Vérification de propriétés d'indistinguabilité pour les protocoles cryptographiques

Antoine Dallon

université PARIS-SACLAY

école normale supérieure paris–saclay

Thèse de Doctorat

NNT : 2018SACLN044

Vérification de propriétés d'indistinguabilité pour les protocoles cryptographiques

Petites attaques et décision efficace avec SAT-Equiv

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'École Normale Supérieure de Paris-Saclay

Ecole doctorale n°580 Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Cachan, le 26 novembre 2018, par

ANTOINE DALLON

Composition du Jury :

Catuscia PALAMIDESSI Directrice de Recherche, INRIA	Présidente
Luca VIGANÒ Professeur, King's College London	Rapporteur
Yannick CHEVALIER Maître de Conférences, Université Toulouse 3	Rapporteur
Luca COMPAGNA Chercheur, SAP	Examinateur
David LUBICZ Chercheur, DGA et Université Rennes 1	Examinateur
Stéphanie DELAUNE Directrice de Recherche, CNRS	Directrice de thèse
Véronique CORTIER Directrice de Recherche, CNRS	Co-encadrante de thèse

Vérification de propriétés d'indistinguabilité pour les protocoles cryptographiques

Petites attaques et décision efficace avec SAT-Equiv

Thèse de doctorat effectuée sous la direction de Stéphanie Delaune et Véronique Cortier

Antoine Dallon

26 novembre 2018

Remerciements

Je voudrais commencer par adresser ma reconnaissance la plus sincère à mes deux directrices de thèse, Stéphanie et Véronique, pour leur aide constante et leurs conseils pertinents, depuis le début de mon stage jusqu'à l'achèvement de ma thèse. Vos idées, à commencer par le sujet de thèse lui-même, ainsi que vos remarques, et votre participation active jusque dans les détails de certaines preuves ont permis à mon travail de porter ses fruits, si bien que la qualité de cette thèse aurait été considérablement inférieure, tant dans son contenu scientifique que dans sa présentation, sans votre engagement continuel. Je mesure la chance que j'ai eue de travailler avec vous, et il m'est difficile d'imaginer ce que vous auriez pu faire de plus.

Je souhaiterais également remercier tous les membres du jury, Catuscia Palamidessi, Yannick Chevalier, Luca Viganò, Luca Compagna et David Lubicz pour avoir accepté d'évaluer mon travail, et en particulier les deux rapporteurs, qui se sont plongés dans les détails parfois très techniques de cette thèse. J'en profite également pour remercier la DGA d'avoir financé ces trois années.

Je tiens aussi à signaler ma gratitude envers tous les membres du LSV, qui m'ont accueilli avec gentillesse depuis le début. Je pense particulièrement, bien sûr, aux membres de l'axe SecSI, qu'il s'agisse de David et Hubert, pour leur passion et leur science communicatives, ou bien des doctorants passés et présents, et surtout Adrien, pour nos discussions techniques ou non ; et Rémy, dont les travaux ont profondément irrigué cette thèse. Ces trois années auraient été bien tristes sans la bonne humeur persistante de tous les doctorants, et singulièrement d'Anthony, Samy et Simon, au cours de ces nombreux mois à partager un bureau commun. Le laboratoire ne fonctionnerait pas si bien (voire pas du tout) sans les services administratifs et techniques, et je tiens à remercier tout particulièrement Virginie, Imane, Hugues et Francis, pour l'aide qu'ils m'ont apportée tout au long de mon séjour au LSV. Ma gratitude s'adresse également à tous ceux que j'ai pu croiser à Nancy ou à Rennes, et surtout à Vincent, parce qu'il m'a accueilli pendant la fermeture estivale de l'ENS, mais aussi pour son efficacité dans la correction de Deepsec à un moment crucial.

Au cours de ces dix dernières années, j'ai eu la chance de tisser plusieurs amitiés qui me sont chères. Qu'elles se soient poursuivies à travers des promenades vespérales, des péripéties partagées dans l'aventureuse colocation de la rue Marguerin, ou de simples, mais sympathiques, conversations, je veux remercier, à cette occasion, tous ceux qui continuent d'y prendre part, et en particulier Rémi et Gabriel, qui m'ont fait le plaisir de se déplacer à l'occasion de ma soutenance.

Les mots me manquent pour dire mon immense dette envers toute ma famille et son soutien infaillible dans toutes les épreuves que j'ai pu traverser ces trois dernières années. Il ne fait aucun doute que les travaux présentés ici en ont grandement bénéficié. Je voudrais également mentionner l'importance de l'aide plus ponctuelle, mais néanmoins cruciale, qui m'a été apportée par Fabien, Claire et ma mère dans l'organisation de la soutenance.

Enfin, je voudrais terminer par quelques mots pour Marie. En me suggérant, avec perspicacité, de ne pas numéroter l'introduction, tu étais loin d'apporter ta contribution la plus précieuse à la qualité de mon travail et plus généralement à mon bonheur, si bien qu'il serait vain de tenter d'énoncer tous les titres auxquels tu mérites de figurer dans ces remerciements, à commencer par ton aide et ta compréhension pendant la phase de rédaction, ainsi que ta présence dans tous les moments où elle était nécessaire. Je préfère donc te dire, tout simplement : merci pour tout.

Résumé

Cette thèse s'inscrit dans le domaine de la vérification de protocoles cryptographiques dans le modèle symbolique. Plus précisément, il s'agit de s'assurer, à l'aide de méthodes formelles, que de petits programmes distribués satisfont à des propriétés d'indistinguabilité, c'est-à-dire qu'un attaquant n'est pas capable de deviner quelle situation (parmi deux) il observe. Ce formalisme permet d'exprimer des propriétés de sécurité comme le secret fort, l'intraçabilité ou l'anonymat. De plus, les protocoles sont exécutés simultanément par un grand nombre d'agents, à plusieurs reprises, si bien que nous nous heurtons très rapidement à des résultats d'indécidabilité. Dès lors, il faut ou bien tenir compte du nombre arbitraire de sessions, et rechercher des méthodes de semi-décision ou identifier des classes décidables ; ou bien établir des procédures de décision pour un nombre fini de sessions.

Au moment où nous avons commencé les travaux présentés dans cette thèse, les outils de vérification de propriétés d'indistinguabilité pour un nombre borné de sessions ne permettaient de traiter que très peu de sessions : dans certains cas il était tout juste possible de modéliser un échange complet. Cette thèse présente des procédures de décision efficaces dans ce cadre. Dans un premier temps, nous établissons des résultats de petite attaque. Pour des protocoles déterministes, nous démontrons qu'il existe une attaque si, et seulement si, il existe une attaque bien typée, lorsque toute confusion entre les types des variables est évitée. De plus, nous prouvons que, lorqu'il existe une attaque, l'attaquant peut la trouver en utilisant au plus trois constantes. Dans un second temps, nous traduisons le problème d'indistinguabilité en termes d'accessibilité dans un système de planification, qui est résolu par l'algorithme du graphe de planification associé à un codage SAT. Nous terminons en confimant l'efficacité de la démarche, à travers l'implémentation de l'outil SAT-Equiv et sa comparaison vis-à-vis des outils analogues.

Abstract

This thesis presents methods to verify cryptographic protocols in the symbolic model: formal methods allow to verify that small distributed programs satisfy equivalence properties. Those properties state that an attacker cannot decide what scenario is beeing played. Strong secrecy, and privacy type properties, like anonymity and unlinkeability, can be modelled through this formalism. Moreover, protocols are executed simultaneously by an unbounded number of agents, for an unbounded number of sessions, which leads to indecidability results. So, we have either to consider an arbitrary number of sessions, and search for semi-decision procedures and decidable classes; or to establish decision procedures for a finite number of sessions.

When we started the work presented in this thesis, the existing equivalence checkers in the bounded model were highly limited. They could only handle a very small number of sessions (sometimes no more than three). This thesis presents efficient decision procedures for bounded verification of equivalence properties. Our first step is to provide small attack results. First, for deterministic processes, there exists an attack if, and ony if, there is a well-typed attack, assuming that there is no confusion between variable types. Second, when there exists a flaw, the attacker needs at most three constants to find it. Then, our second step is to translate the indistinguishability problem as a reachability problem in a planning system. We solve this second problem through planning graph algorithm and SAT encoding. In a final step, we present the implementation of the SAT-Equiv tool, which allows us to evaluate our approach. In particular, a benchmark with comparable tools proves the efficiency of SAT-Equiv.

Table des matières

Introduction

L'utilisation de la cryptographie pour sécuriser tout type de communications remonte à la plus haute antiquité [1]. Toutefois, avec le développement constant des moyens de télécommunication depuis le XIX^{e} siècle, l'essor de l'informatique vers le milieu du XX^{e} siècle et l'avènement d'Internet dans les années 1990, les enjeux liés à la confidentialité des échanges se sont accrus de manière considérable, puisque les moyens de la sécurité informatique ont envahi nos vies quotidiennes à travers la carte bancaire, le passeport et le vote électroniques, la téléphonie mobile et le commerce en ligne.

En première instance, la sécurité repose sur l'utilisation de *primitives cryptographiques*, c'est-à-dire de fonctions mathématiques possédant des propriétés remarquables. Par exemple, une fonction de chiffrement permet de rendre un message inintelligible à quiconque ne connaît pas la clé de déchiffrement. Une fois définies des primitives cryptographiques adéquates, il reste encore, pour chacune de ces applications, à spécifier comment chaque agent doit les utiliser et quels messages il doit transmettre : c'est le rôle des *protocoles cryptographiques*. Ainsi, le protocole *Transport Layer Security* (TLS) sécurise le trafic web, tandis que les protocoles *Single Sign On* permettent à un utilisateur d'un service comme Gmail de s'identifier auprès de tiers sans révéler son mot de passe.

Ces protocoles visent à garantir des *propriétés de sécurité*, y compris en présence d'un attaquant particulièrement puissant, capable d'intercepter, de modifier et d'émettre des messages, ou d'entrer en communication avec l'un des participants. La requête de sécurité la plus naturelle consiste à exiger le *secret* de certaines informations (messages ou clés utiles ultérieurement), c'est-à-dire que l'attaquant ne peut pas les déduire à partir des messages qu'il connaît. Il est rarement suffisant que les échanges restent secrets si nous ne pouvons pas savoir avec qui nous communiquons : la propriété d'*authentification* permet justement de nous assurer de l'identité de notre interlocuteur. Parfois, l'authentification constitue l'objectif unique du protocole, comme lorsque nous utilisons un badge pour ouvrir un garage ou une voiture ; mais le plus souvent, par exemple dans TLS, les deux objectifs sont liés. Ces deux propriétés constituent généralement la fonction principale du protocole : si le badge ou la carte bleue ne garantissaient plus l'authentification, ces objets deviendraient purement inutiles. Cependant, ces protocoles doivent également satisfaire à d'autres exigences de sécurité, en particulier le respect de la vie privée. Plus précisément, un protocole garantit l'*anonymat* s'il ne révèle pas d'informations permettant d'identifier son utilisateur (comme son nom ou sa position). Il garantit l'*intraçabilité* s'il ne permet pas de savoir si un utilisateur a utilisé le même protocole plusieurs fois. Une autre propriété consiste à demander la *confidentialité du vote* pour un système de vote électronique.

Il n'est pas facile de déterminer si un protocole garantit les propriétés souhaitées. En effet, au cours du temps, de nombreuses attaques ont été découvertes sur les protocoles, y compris lorsque la cryptographie est parfaite : de telles erreurs sont appelées *failles logiques*. Ainsi, en 2008, Armando *et al.* [18] ont démontré que l'une des variantes déployées du protocole *OASIS SAML 2.0* (utilisée pour le *Single Sign On* de Google) ne respectait pas l'authentification de son utilisateur, puisqu'un service tiers pouvait usurper l'identité de cet utilisateur auprès d'un autre fournisseur de service. L'importance de ce type de failles, affectant des millions d'utilisateurs, et la difficulté de les découvrir, nous incitent à entreprendre une vérification méthodique des protocoles cryptographiques.

1. D'après Kahn [80], le plus vieux document chiffré connu est une recette secrète datant du XVI^{e} siècle av. J.-C. trouvée en Irak.

Primitives

La cryptographie constitue l'un des plus anciens domaines de l'informatique, mais les schémas utilisés sont restés relativement faibles jusqu'au XVI^e siècle. En effet, la sécurité du chiffrement de César, de la méthode Atbash [2] ou du carré de Polybe [3] reposaient principalement sur le secret de l'algorithme. Au contraire, l'algorithme utilisé par Bellaso et Vigenère, qui consiste à utiliser pour chaque caractère un décalage spécifique paramétré par une clé, est plus robuste. Il ne sera cassé, sous certaines hypothèses, qu'au XIX^e siècle. En 1883, Kerckhoffs énonce le principe que la sécurité ne doit pas reposer sur le secret de l'algorithme. Au XX^e siècle, la cryptanalyse joue un rôle important dans les deux guerres mondiales. Vernam invente en 1917 le système de chiffrement qui consiste à ajouter (modulo 26) une clé aléatoire, aussi longue que le message, et à usage unique (le *masque jetable*) au texte à chiffrer. Il s'agit donc d'un perfectionnement du chiffrement de Vigenère, où la clé n'est plus une phrase à répéter mais une longue séquence n'ayant aucune propriété statistique particulière. À cette époque, les techniques cryptographiques utilisent de plus en plus de moyens automatiques, qui culminent avec les attaques contre Enigma grâce aux « bombes » de Bletchley Park sous la direction de Turing. Après la guerre, en 1949, Shannon publie son résultat fondamental [97] : le masque jetable est le seul système qui ne révèle aucune information sur le message échangé. Le niveau de sécurité de ce système se paie cependant par la longueur des clés (à usage unique) à échanger. À partir de cette date, la cryptographie évolue considérablement. Dans les années 1970, l'algorithme de chiffrement de référence DES (pour *Data Encryption Standard*) est défini. Mais l'avancée la plus considérable provient de l'apparition de la cryptographie à clé publique, grâce au protocole d'échange de clés de Diffie-Hellman [67] (1976) et la description par Rivest, Shamir et Adleman [93] de l'algorithme RSA en 1978. Ces travaux permettent d'envisager l'existence d'une multitude de primitives.

Chiffrement symétrique. Cette primitive est la plus classique : il s'agit d'utiliser une clé sur un texte clair pour obtenir un message chiffré. La personne qui reçoit ce message chiffré doit utiliser la clé de chiffrement pour récupérer le message d'origine : il s'agit d'une *clé partagée*. Tous les algorithmes antérieurs à 1975 sont des implémentations, plus ou moins sécurisées, du chiffrement symétrique. Les implémentations modernes utilisent majoritairement le principe du *chiffrement par blocs* où chaque bloc d'un nombre arbitraire de bits est chiffré à part. C'est par exemple le cas d'AES, (pour *American Encryption Standard*) qui est le chiffrement symétrique le plus utilisé. Une autre méthode, plus rare, appelée *chiffrement par flot*, consiste à générer une suite pseudo-aléatoire à partir d'une clé et à s'en servir comme d'un masque. C'est le cas dans RC4, utilisé dans le chiffrement WEP. Dans tous les cas, la sécurité repose sur l'expérience : un système est sûr parce qu'il résiste à une classe d'attaques connues.

Le chiffrement symétrique possède deux inconvénients majeurs et intrinsèques : il est nécessaire d'échanger les clés avant la communication, et chaque agent doit nécessairement disposer d'une clé différente par interlocuteur. En particulier, si Alice et Bob ne partagent pas de secret commun, il est impossible pour eux de communiquer sans appel à un tiers de confiance. Cependant, comme le chiffrement symétrique est la primitive dont le calcul est le plus efficace, elle est souvent utilisée avec une clé communiquée par un autre moyen.

2. Le chiffrement de César, utilisé par Jules César pendant sa campagne gauloise, consiste à décaler les lettres de 3 rangs dans l'alphabet, et ainsi à remplacer le A par le D. Dans le même esprit, la méthode Atbash remplace chaque lettre par la lettre de même rang dans l'ordre alphabétique inverse, et donc le A par le Z.

3. Dans le carré de Polybe, les lettres sont disposées en carré et chacune est remplacée par le numéro de sa ligne et de sa colonne. Ainsi, 1,1 remplace la lettre A.

Chiffrement asymétrique. Cette primitive permet de pallier aux inconvénients du chiffrement symétrique. En particulier, la clé de chiffrement et la clé de déchiffrement sont distinctes. Ainsi, chaque agent dispose d'une unique clé de déchiffrement et peut, en toute sécurité, révéler la clé de chiffrement associée. Historiquement, le premier algorithme de chiffrement asymétrique est RSA, mais plus récemment d'autres méthodes, qui reposent sur la difficulté du logarithme discret, ont été développées, par exemple dans les courbes elliptiques. Ces primitives utilisent des opérations bien plus coûteuses que le chiffrement symétrique et leur temps de calcul est bien supérieur. Pour des raisons d'efficacité, la plupart des échanges consistent donc à chiffrer le message par une clé symétrique fraîche, et à utiliser la cryptographie asymétrique pour transmettre la clé symétrique. Remarquons que dans un protocole d'établissement de clé partagée, le chiffrement symétrique complète le chiffrement asymétrique de deux manières différentes : d'une part, il permet de chiffrer chaque message avec une clé chaque fois fraîche, et d'autre part, il succède au chiffrement asymétrique une fois que la clé partagée a été établie.

Signature numérique. Cette primitive consiste à prouver qu'un message a bien été émis par un certain agent. Comme pour le chiffrement asymétrique, l'agent possède une clé secrète et révèle une clé publique associée. Lorsqu'il veut signer un message, il applique la primitive avec la clé secrète sur ce message. Ensuite, toute personne qui connaît la clé publique peut vérifier la signature. Cette primitive est donc en quelque sorte l'opération inverse du chiffrement asymétrique, puisque la clé de signature est privée et la clé de vérification est publique. Pour cette raison, un système de chiffrement asymétrique peut souvent être utilisé comme système de signature numérique. RSA, par exemple, fournit également un système de signature. Un tel système doit respecter l'*inforgeabilité* : un attaquant ne doit pas pouvoir forger une signature valide sans en connaître la clé.

Fonction de hachage. Cette primitive accepte en entrée un message de longueur arbitraire et en calcule un de taille fixée possédant principalement deux propriétés. D'une part, la fonction n'est pas inversible, c'est-à-dire que l'on ne peut obtenir aucune information sur le message d'origine à partir de son haché : il s'agit d'une fonction *à sens unique*. D'autre part, la fonction résiste aux collisions, c'est-à-dire qu'il est calculatoirement impossible de trouver deux messages qui donnent le même résultat. Souvent, un système de signature électronique fait appel à une fonction de hachage : en particulier, au lieu d'appliquer les opérations arithmétiques du schéma de signature à un document en entier, la meilleure solution consiste à commencer par le hacher, puis à signer le résultat. De cette manière, l'intégrité de la signature numérique est assurée par construction, et le processus complet est plus efficace (les fonctions de hachage sont encore moins coûteuses que le chiffrement symétrique). Les fonctions de hachage permettent également de construire des MAC (pour *Message Authentication Code*), c'est-à-dire des fonctions qui ont les mêmes propriétés que le hachage, mais qui requièrent en plus une clé secrète partagée. Ainsi, si Alice et Bob partagent une clé secrète k, ils sont les seuls à pouvoir créer $\mathsf{mac}(m, k)$ pour un message m arbitraire. La vérification du mac consiste à le reconstruire à partir de m et de k, puis à effectuer un test d'égalité. Pour construire une fonction de MAC à partir d'une fonction de hachage, il suffit par exemple de concaténer la clé au message avant de hacher. Les algorithmes de hachage les plus utilisés sont les algorithmes de référence SHA-2 et SHA-3 (pour *Secure Hash Algorithm* 2 et 3).

Plus récemment, de nouvelles primitives plus exotiques ont été construites. En particulier, les *preuves sans divulgation de connaissance* permettent de démontrer que l'on possède une information secrète sans rien révéler que l'attaquant ne puisse calculer lui-même. D'autres développement concernent le *chiffrement homomorphe* qui permet à un tiers de réaliser des calculs sur des données auxquelles il n'a pas accès.

Protocoles

Pour obtenir des garanties de sécurité, l'inviolabilité des primitives cryptographiques n'est pas suffisante. En effet, même pour de simples communications, une utilisation naïve de la signature numérique ou du chiffrement asymétrique permet le plus souvent des attaques, par exemple par rejeu. Si Charlie reçoit les messages « Oui » et « Non » signés par Alice, il peut facilement se faire passer pour elle en répondant à n'importe quelle question posée par Bob. Pour se protéger de ce genre d'attaques, Alice doit respecter une certaine procédure : par exemple ajouter la date du message auquel elle répond, ou son haché. Si elle veut en plus être sûre qu'elle communique bien avec Bob, elle doit le contacter et lui demander de respecter également certaines instructions (par exemple, signer un certain message pour prouver son identité). Alice et Bob doivent donc mettre en œuvre un *protocole cryptographique*, c'est-à-dire un programme distribué faisant appel aux primitives cryptographiques. Dans la vie quotidienne, ces protocoles sont exécutés directement par les systèmes informatiques.

Une première classe de protocoles est constituée par les protocoles de communications, c'est-à-dire ceux qui servent exclusivement à établir des communications fiables entre plusieurs agents. Nous avons déjà cité à ce titre TLS, qui permet d'accéder au web de manière sécurisée, c'est-à-dire d'une part d'authentifier le site avec lequel nous communiquons et d'autre part de protéger nos échanges avec le serveur vis-à-vis de tiers, et en particulier de notre fournisseur d'accès. Pour prouver son identité, chaque site doit envoyer un certificat signé par une autorité reconnue et dont la clé publique est connue par le navigateur. Les protocoles SSH (pour *Secure SHell*), qui permet d'accéder à une machine distante pour exécuter des instructions, et AKA (pour *Authentication and Key Agreement*), qui sécurise les communications mobiles, représentent deux autres exemples de protocoles de communication. Un schéma classique consiste à prouver son identité avant d'établir un canal chiffré.

D'autres protocoles utilisent la communication et l'interactivité pour mettre en place certains services. Nous avons déjà évoqué l'exemple d'un badge permettant d'ouvrir une porte, les protocoles *Single Sign On* qui permettent de s'identifier, mais aussi le vote électronique pour lequel plusieurs protocoles ont été proposés, comme Helios [10] ou le système norvégien [66]. Ces deux systèmes ont été déployés pour des élections réelles.

Le passeport électronique permet à son détenteur de démontrer son identité aux autorités compétentes grâce à plusieurs sous-protocoles. Le premier d'entre eux est le protocole BAC (pour *Basic Access Control*), décrit à la figure 1, qui permet d'établir une clé partagée pour la suite. Le protocole est joué par la puce RFID du passeport et le lecteur de l'autorité de vérification. Une clé de chiffrement k_s et une clé de MAC k_m, inscrites sur le passeport, sont transmises au lecteur par lecture optique avant l'exécution du protocole. Le lecteur commence par envoyer un message d'initialisation getC à la puce, qui crée alors un nombre aléatoire n_p qu'elle adresse au lecteur. Celui-ci crée alors un nombre aléatoire jetable (aussi appelé *nonce*, pour *Number only used once*) n_ℓ et une clé fraîche k_ℓ de son côté, et répond en envoyant n_ℓ, n_p et k_ℓ chiffrés par k_s, ainsi qu'un MAC de ce message avec la clé k_m afin de garantir l'intégrité de l'échange. La puce vérifie le MAC, puis ouvre le chiffrement. Il peut alors vérifier le nonce n_p et récupérer n_ℓ et k_ℓ. Il crée une clé fraîche k_p et répond en envoyant k_p, n_ℓ et n_p chiffrés par k_s, ainsi que le MAC, avec la clé k_m, de ce chiffrement, pour en assurer l'intégrité. Le lecteur peut alors vérifier le MAC, ouvrir le chiffrement, vérifier le nonce n_p et obtenir la clé k_p. Le lecteur et la puce RFID possèdent k_p et k_ℓ et peuvent en déduire une clé partagée $k_0 = k_p \oplus k_\ell$ (où $\oplus$ représente le XOR bit-à-bit). Les données transmises par le passeport ne changent pas d'une fois sur l'autre. Utiliser la clé k_0 au lieu de la clé k_s permet d'éviter les attaques par rejeu et de garantir que les messages seront différents d'une fois sur l'autre, afin qu'un adversaire qui ne possède pas la clé k_s ne puisse pas savoir que le protocole est rejoué.

1. $L \to P$: getC
2. $P \to L$: n_p
3. $L \to P$: $\langle x_{senc}, \mathsf{mac}(x_{senc}, k_m)\rangle$ $\quad [x_{senc} = \mathsf{senc}(\langle n_\ell, n_p, k_\ell\rangle, k_s)]$
4. $P \to L$: $\langle y_{senc}, \mathsf{mac}(y_{senc}, k_m)\rangle$ $\quad [y_{senc} = \mathsf{senc}(\langle n_p, n_\ell, k_p\rangle, k_s)]$

FIGURE 1 – Protocole BAC

Propriétés

Les protocoles servent à garantir la sécurité des échanges, c'est-à-dire certaines propriétés comme l'authentification ou le secret. Bien entendu, il ne s'agit pas seulement de montrer que les protocoles sont sécurisés en face d'un attaquant *passif*, capable seulement d'espionner les conversations : nous devons également supposer que l'attaquant peut intercepter les messages et qu'il peut en construire de nouveaux, en réalisant des calculs à partir des primitives, des données publiques et des messages qu'il a reçus. Nous disons alors que l'attaquant est *actif*. De plus, si Alice communique avec Bob et Charlie, la sécurité de sa communication avec Bob ne devrait pas être affectée par la malhonnêteté de Charlie.

Certaines propriétés s'expriment simplement en considérant toutes les exécutions d'un protocole : nous disons que ce sont des propriétés d'*accessibilité*, ou parfois des propriété *de trace*. Dans d'autres cas, la propriété stipule que le protocole doit être *indistinguable* d'un autre protocole. Ce second protocole représente souvent la spécification d'une propriété de sécurité : par exemple, une information importante a été remplacée par un aléa, et l'attaquant est incapable de la retrouver. Nous parlons alors de propriétés d'*équivalence* (ou d'*indistinguabilité*). La vérification des propriétés d'accessibilité a atteint un certain degré de maturité. Dans la suite, nous nous intéresserons particulièrement aux propriétés d'équivalence.

Secret. Le secret d'une donnée (par exemple une clé partagée établie au cours du protocole) est sans doute la propriété la plus naturelle. À l'origine, cette propriété a été modélisée comme une propriété d'accessibilité : il ne doit exister aucune exécution du protocole qui permette à l'attaquant de déduire la clé. Cependant, une telle propriété est parfois bien trop faible. Si l'attaquant dispose d'un haché d'un mot de passe court, il lui est impossible de déduire directement ce mot de passe puisque la fonction de hachage n'est pas inversible. Cependant, il peut facilement monter une attaque hors ligne par force brute. Il lui suffit d'essayer de hacher chaque mot de passe jusqu'à obtenir le bon résultat. Une solution consiste à exiger une notion de secret plus forte. Supposons que l'on veuille démontrer le secret fort de s dans P_0. Nous définissons le protocole P comme le protocole qui exécute P_0 puis révèle s, et Q comme le protocole qui exécute P_0 puis révèle un nonce frais k. Si P et Q sont indistinguables du point de vue de l'attaquant, alors il est incapable de faire la différence entre s et un nom frais qu'il n'a jamais vu. Autrement dit, l'attaquant n'a aucun moyen de reconnaître la valeur de s. Il lui est donc impossible de réaliser l'attaque hors ligne sur s que nous avons évoquée. Le *secret fort* est respecté si les protocoles P et Q sont équivalents.

Authentification. L'authentification permet à un utilisateur de prouver son identité. Dans certains cas, par exemple lorsque nous nous connectons sur le site de notre banque, chaque agent doit identifier l'autre : la banque ne doit pas révéler des informations concernant un client à des tiers, mais un client n'a pas non plus intérêt à confier ses identifiants à un site usurpant l'identité de la banque. Lorsque les participants aux protocoles s'authentifient l'un vis-à-vis de l'autre, nous parlons d'*authentification mutuelle*. L'authentification de Bob par Alice s'exprime en affirmant que, pour toute exécution où Alice parvient jusqu'au bout du protocole en pensant qu'elle communique avec Bob, Bob a bien engagé une communication avec elle. Il s'agit donc d'une propriété d'accessibilité.

Anonymat. Lorsqu'un protocole manipule des données sensibles (par exemple des données médicales) ou tout simplement personnelles, un tiers ne doit pas pouvoir y accéder. L'anonymat consiste précisément à affirmer que l'exécution du protocole ne peut pas permettre à un attaquant d'identifier l'un des agents. Cette propriété peut se voir comme une variante du secret, mais elle nécessite des mesures adaptées. En particulier, il faut considérer que l'attaquant connaît les identités *a priori*. Par exemple, une attaque permettrait à un employeur potentiel de vérifier si un candidat, dont il connaît le nom, a récemment communiqué avec un hôpital. L'anonymat doit donc se voir comme une variante du secret fort qui porte sur les informations personnelles : il s'agit d'une propriété d'équivalence.

Intraçabilité. Même lorsque le protocole respecte l'anonymat, certaines menaces peuvent continuer de peser sur la vie privée de l'utilisateur. En particulier, lorsque le protocole est joué plusieurs fois, un attaquant pourrait être capable de savoir si un même agent a participé plusieurs fois. Cette information peut ensuite être recoupée avec d'autres pour suivre quelqu'un. Par exemple, si un employeur sait qu'une même personne a utilisé son téléphone portable dans la salle d'attente de son entreprise et à l'hôpital, il est facile pour lui d'en déduire l'identité de cette personne. Lorsque le protocole ne permet pas ce genre d'attaques, il respecte l'intraçabilité. Nous pouvons l'exprimer en disant que le scénario où le protocole est exécuté plusieurs fois par plusieurs personnes est indistinguable du scénario où le protocole est exécuté chaque fois par une personne différente. Il n'existe pas d'exemple réel [4] de protocole qui serait anonyme sans être indistinguable, mais l'inverse est assez facile à concevoir : il suffit de penser à un protocole qui n'échange aucune information sensible mais qui s'exécute toujours de la même manière si les interlocuteurs sont les mêmes.

Secret du vote. La dernière propriété que nous envisageons concerne le vote électronique. À première vue, il pourrait sembler qu'il s'agit simplement d'une variante du secret. Cependant, lors d'un vote, les noms des candidats sont publics, et il est impossible de demander le secret d'une information publique. La propriété adaptée paraît donc être le secret fort : par exemple, voter pour Alice est indistinguable de voter pour Bob. Pourtant, un protocole de vote finit nécessairement par annoncer un candidat gagnant. Or, si le protocole de vote déclare Alice gagnante à une seule voix près, la situation où le vote d'un seul électeur d'Alice change n'est pas indistinguable de la situation d'origine. La propriété de secret du vote consiste donc à énoncer que dans tous les cas de figure, le scénario où Arthur vote pour Alice et Béatrice vote pour Bob est indistinguable de celui où Arthur vote pour Bob et Béatrice vote pour Alice. Il s'agit donc d'une propriété d'équivalence.

4. Il existe cependant des exemples artificiels, selon le modèle choisi. Pour une comparaison plus complète de ces deux propriétés, il sera utile de consulter le travail d'Arapinis, Clothia, Ritter et Ryan [14].

1. $Att. \to P$: getC
2. $P \to Att.$: n_p
3. $Att. \to P$: $\langle x^0_{senc}, \mathsf{mac}(x^0_{senc}, k_m)\rangle$ [observés lors d'un échange précédent.]

FIGURE 2 – Début de l'attaque sur BAC

Attaques

Au cours des années, de nombreuses attaques ont été trouvées. En 1995, Lowe [83] a découvert une faille très simple sur le protocole de Needham-Schroeder [87] publié en 1978. Mais celles qui portent sur des protocoles déployés sont les plus spectaculaires. Ainsi, Armando, Carbone et Compagna [18] ont découvert en 2008 que l'une des versions du protocole *Single Sign On* utilisé par Google permettait à un service d'usurper l'identité d'un utilisateur auprès d'un tiers. De leur côté, Bhargavan *et al.* [33] ont trouvé plusieurs attaques sur TLS, dont FREAK : une faille logique permettait à un attaquant de forcer l'utilisation de *RSA Export*, qu'il est possible de casser aujourd'hui.

Nous présentons une autre faille qui porte sur le protocole BAC, et qui a été découverte par Chothia et Smirnov [50]. Rappelons que sa description se trouve en figure 1. Avant de présenter l'attaque, il faut néanmoins préciser le comportement du protocole, et surtout celui de la puce RFID. À l'étape 3, lorsque la puce reçoit le message chiffré et le MAC, elle commence par vérifier le MAC, puis, si le MAC est correct, elle déchiffre le message et vérifie le nonce n_p. Dans l'implémentation originale du passeport français, si le MAC n'était pas correct, la puce envoyait un message d'erreur $\mathsf{erreurMAC}$. Si c'est le nonce qui ne correspondait pas, la puce envoyait un autre message $\mathsf{erreurNonce}$. En exploitant de tels messages d'erreurs, un attaquant pouvait violer l'intraçabilité du passeport, et c'est pourquoi ce comportement a été corrigé : dans l'implémentation actuelle, les erreurs $\mathsf{erreurMAC}$ et $\mathsf{erreurNonce}$ sont remplacées par une seule et même erreur erreur. Remarquons que les implémentations sont réalisées indépendamment dans chaque pays : cette faille, par exemple, n'a jamais existé sur le passeport britannique.

Pour monter l'attaque, l'adversaire doit commencer par observer un échange normal entre le passeport d'Alice et un lecteur. En particulier, il doit obtenir une paire $\langle x^0_{senc}, \mathsf{mac}(x^0_{senc}, k_m)\rangle$. C'est sans doute la partie la plus difficile en pratique : il faut se trouver à proximité du lecteur et de la puce au moment de l'échange. La suite de l'attaque est très simple : lorsqu'une puce est à proximité, il suffit d'initier un échange comme décrit à la figure 2 : l'attaquant envoie getC ; la puce répond donc en envoyant un nonce frais n'_p, et l'attaquant lui répond en rejouant le message $\langle x^0_{senc}, \mathsf{mac}(x^0_{senc}, k_m)\rangle$ qu'il a enregistré lors de la session observée.

Comme le message x^0_{senc} n'est pas correct (il n'utilise pas le nonce n'_p qui vient d'être créé par la puce), la puce ne poursuit pas le protocole, ce qui protège le détenteur du passeport : l'attaquant ne peut pas accéder aux informations qui seront révélées dans la suite, après l'exécution de BAC. Cependant, si le passeport est celui d'Alice, le MAC est correct, et l'attaquant observe donc le message $\mathsf{erreurNonce}$; tandis que si le passeport n'est pas celui d'Alice, la vérification du MAC échoue et l'attaquant observe $\mathsf{erreurMAC}$. C'est donc un moyen pour lui de savoir si Alice est passée à proximité du lecteur de l'attaquant.

Même après la correction du protocole, l'intraçabilité n'est pas garantie dans la pratique. En effet, les deux cas d'erreurs ne se produisent pas au même moment : après la vérification du MAC, il faut ajouter le temps de déchiffrement pour obtenir l'erreur qui correspond à un mauvais nonce. Il s'agit d'un temps court, mais non nul, et il est possible à un attaquant de mesurer le temps avant de recevoir le message d'erreur. De cette manière, l'ambiguïté sur la nature de l'erreur est levée, ce qui permet de retrouver l'attaque. Dans ce cas, nous parlons d'attaques par *canaux cachés*

(*side channel*) et plus précisément d'attaques *chronométrique* (*timing attack*). Pour s'en protéger, il suffit d'établir un délai commun pour l'émission des messages d'erreurs. Cependant, d'autres attaques physiques peuvent subsister, en s'appuyant sur des données telles que la consommation énergétique ou les émissions électromagnétiques.

De plus, les clés k_s et k_m sont inscrites sur la page qui contient toutes les informations personnelles, ce qui remet fortement en cause le modèle d'un attaquant ne connaissant pas ces clés. Comme les informations écrites sur cette page sont les mêmes que celles qui sont conservées dans la puce, cette situation peut sembler tout à fait légitime. Cependant, il est clair qu'un attaquant en possession des clés peut facilement casser l'intraçabilité du passeport. Par exemple, il est légitime qu'un employeur ait accès aux informations générales concernant un candidat, telles que sa date et son lieu de naissance, ou sa photographie d'identité. Mais en donnant une copie de son passeport (et donc les clés), le candidat donne à son employeur le moyen de le suivre à peu de frais.

Vérification

L'existence de failles dans des protocoles déployés, y compris lorsqu'une attention particulière a été portée à la sécurité, montre la nécessité de développer des méthodes pour vérifier les protocoles. Le fait que des attaques relativement simples sont découvertes seulement après plusieurs années, y compris quand les protocoles ont été prouvés à la main, incite fortement à automatiser les preuves et la recherche d'attaques à l'aide des méthodes formelles. L'une des difficultés principales repose sur le nombre d'exécutions possibles : un nombre arbitraire d'agents exécute le protocole un nombre arbitraire de fois. De plus, le comportement de l'attaquant n'est pas spécifié, et en particulier il n'est pas forcé de respecter le protocole. Le rôle de la vérification formelle des protocoles n'est pas seulement de trouver des attaques sur des protocoles existants, mais aussi, si possible, de participer à la conception des protocoles, le plus tôt possible, afin d'éviter de déployer des protocoles contenant des failles. Deux approches principales ont été envisagées : l'approche *calculatoire* et l'approche *symbolique*.

Modèle calculatoire. Cette première méthode consiste à se rapprocher le plus possible des primitives réelles. Ainsi, chaque primitive respecte des hypothèses précises, qui sont formulées par des jeux cryptographiques. Les messages sont des suites de bits, et l'attaquant est une machine de Turing probabiliste polynomiale. Il s'agit donc de montrer que la probabilité que l'attaquant distingue deux protocoles décroît (asymptotiquement) très vite avec le paramètre de sécurité. Une preuve dans ce modèle garantit donc qu'il n'y aura pas d'attaque sur le protocole sans attaque sur les primitives. Cependant, ces preuves sont très difficiles, et sont souvent réalisées à la main. Or, une telle démarche est assez difficilement généralisable : chaque variante de chaque protocole nécessite des heures pour être vérifiée, tandis que l'automatisation permet de traiter chaque exemple assez rapidement, et avec des compétences bien moindres. À partir des travaux de Blanchet [35] en 2008, un effort d'automatisation a été entrepris, mais la complexité des protocoles et des scénarios qu'il est possible de vérifier reste limitée.

Modèles symboliques. Une deuxième méthode, qui est celle que nous retenons, consiste à sacrifier un peu de précision pour gagner beaucoup en automatisation. Les primitives cryptographiques sont complètement abstraites, et sont supposées parfaites : par exemple, l'attaquant ne peut rien apprendre sur le contenu d'un texte chiffré sans en connaître la clé. Les messages échangés sur le réseau sont représentés par une algèbre de termes, et les relations entre primitives sont décrites par des théories équationnelles. Pour donner une surapproximation réaliste de l'attaquant, nous supposons que l'attaquant *est* le réseau. En particulier, il récupère et peut intercepter chaque message émis ; en fonction de ce qu'il a appris, il peut construire de nouveaux messages. Pour obtenir automatiquement des preuves calculatoires, la correction calculatoire d'un modèle symbolique sous certaines hypothèses a été démontrée par Abadi et Rogaway [9]. Une autre approche intéressante, due à Bana et Comon [29], consiste à utiliser un modèle symbolique calculatoirement correct par construction.

Le plus ancien modèle symbolique a été défini par Dolev et Yao [68] en 1983. Puisque les modèles symboliques reposent sur une abstraction et un équilibre entre l'exactitude et l'automatisation, plusieurs compromis ont été envisagés. D'une part, les messages sont généralement modélisés par une algèbre de termes, dont les atomes représentent les données, comme les nonces, les clés ou les constantes. Les symboles de fonctions représentent les opérations comme la concaténation ou la dérivation de clé, et les primitives cryptographiques. Les relations entre primitives sont représentées par des théories équationnelles plus ou moins restreintes, qui se ramènent souvent à un système de réécriture convergent. D'autre part, diverses modélisations ont été envisagées pour les protocoles à la fin des années 1990. Le modèle des *Strand spaces* [74], qui permet de représenter les exécutions sous forme d'un graphe. La réécriture de multi-ensemble [42] utilise les quantificateurs existentiels pour représenter les instanciations possibles des variables. Cependant, comme les protocoles sont des programmes distribués, l'une des idées les plus naturelles consiste à les représenter par une algèbre de processus. Dans cette optique, Abadi et Gordon [8] ont défini le spi-calcul. Plus tard, le pi-calcul appliqué a été proposé par Abadi et Fournet [7]. Le principal avantage du pi-calcul appliqué par rapport au spi-calcul consiste à rendre l'algèbre de processus largement indépendante des termes qui sont échangés, et en particulier à permettre l'utilisation de théories équationnelles arbitraires. Une présentation plus complète du pi-calcul appliqué a été proposée récemment par Abadi, Blanchet et Fournet [5].

Décidabilité et indécidabilité

Plusieurs résultats d'indécidabilité existent pour des propriétés d'accessibilité. Amadio, Lugiez et Vanackère [12] codent l'arrêt d'une machine à deux compteurs comme propriété de secret dans un protocole cryptographique (le secret est vérifié si, et seulement si, la machine termine), tandis que Heintze et Tygar [77] utilisent le problème de correspondance de Post [90]. Durgin, Lincoln, Mitchell et Scedrov [69] ont démontré que, même pour une classe de protocoles très restreinte, la question de savoir si un protocole respecte le secret d'un nonce est indécidable. En particulier, borner la profondeur des messages ne suffit pas à obtenir la décidabilité. L'indécidabilité a été étendue assez naturellement aux propriétés d'équivalences, notamment par Chrétien, Cortier et Delaune [54].

La première approche consiste donc à établir des procédure de semi-décision. Cette approche a d'abord été entreprise pour les propriétés d'accessibilité, et les outils existants ont ensuite été étendus à l'équivalence. Par exemple, l'outil ProVerif [34] prouve des propriétés d'accessibilité pour un nombre arbitraire de sessions grâce à une surapproximation des comportements de l'attaquant, ce qui signifie que ProVerif découvre parfois de fausses attaques. ProVerif ne termine pas toujours, mais lorsqu'il termine sans trouver d'attaque, il est démontré que le protocole est sûr, y compris pour un nombre arbitraire de sessions. ProVerif a été étendu aux propriétés d'équivalences [37]

pour des protocoles qui diffèrent seulement sur les termes (*diff-équivalence*). Maude-NPA [73] et Tamarin [86] ont été étendus de la même manière [95, 30]. Tamarin procède par une recherche vers l'arrière. Son automatisation n'est pas complète : dans certains cas, l'utilisateur doit intervenir pour compléter la preuve. Cette caractéristique permet de réaliser des preuves dans des cas où l'automatisation serait impossible. Maude-NPA tient compte de propriétés algébriques élaborées mais le paie par des temps de calculs bien supérieurs, qui rendent impossible l'analyse de protocoles complexes. Une approche complètement différente, qui a mené au développement de l'outil TypeEquiv [64, 65], permet de démontrer l'équivalence pour un nombre borné ou non de sessions, en prouvant la sécurité à travers un système de types. L'outil ne conclut pas toujours, mais il répond très vite et donne des preuves correctes. De plus, il ne force pas à choisir entre un modèle borné et un modèle non-borné : il est possible de choisir les protocoles qui s'exécutent un nombre arbitraire de fois. Il est également envisageable de ramener le problème à des propriétés plus simples. En particulier, l'intraçabilité, qui ne peut pas être formulée directement en termes de diff-équivalence, peut être démontrée indirectement dans ProVerif par des méthodes correctes, mais incomplètes, à travers l'identification de conditions suffisantes [79].

Une autre approche consiste à identifier des classes de protocoles pour lequelles l'équivalence est décidable. L'une des méthodes consiste à ne considérer qu'un nombre borné de sessions. Cette méthode risque de manquer des failles, puisqu'il peut exister des attaques pour un nombre plus grand de sessions. Néanmoins, elle permet d'obtenir des résultats de décision : la terminaison est souvent garantie et toute attaque découverte correspond à une attaque réelle, au moins dans le modèle de l'outil en question. Pour un nombre borné de sessions, et dans le cadre des propriétés d'accessibilité avec une théorie de réécriture fixée, le problème de vérification est Co-NP-complet [94]. Ce résultat a été étendu à des opérateurs possédant des propriétés algébriques, comme le *ou exclusif* [48] ou l'exponentiation [47]. Pour les théories convergentes dont toutes les règles ne réécrivent que vers des sous-termes, la vérification de l'équivalence pour un attaquant passif est décidable en temps polynomial si la théorie n'est pas un paramètre du problème [6, 56, 59]. Cependant, Cheval, Kremer et Rakotonirina [46] établissent que le problème d'équivalence statique est co-NP-complet si l'on tient compte du nombre de règles de la théorie. Dans les mêmes conditions, ils démontrent également que le problème de l'équivalence de trace est co-NEXP-complet.

Il est alors possible d'établir des procédures et des outils pour un nombre borné de sessions. En la matière, un résultat précurseur, qui portait sur la décidabilité de l'équivalence pour un nombre borné de sessions, a été montré par Baudet [31]. La preuve a été simplifiée ultérieurement par Chevalier et Rusinowitch [49]. SPEC [99] est le premier outil à vérifier l'indistinguabilité, pour une signature fixée. La propriété vérifiée n'est pas exactement l'équivalence de trace, mais la bisimulation ouverte qui est une propriété plus forte, ce qui peut mener à de fausses attaques. L'outil APTE [45] permet de vérifier l'équivalence de trace, toujours pour une signature fixée. L'algèbre de processus d'APTE contient des constructions comme les branches *else* et les canaux privés. Akiss [44] utilise les clauses de Horn pour modéliser les protocoles. L'algèbre de termes est paramétrique, et la terminaison est démontrée pour toute une classe de théories [43]. Les possibilités de modélisation d'Akiss ont permis d'étendre l'outil pour représenter l'opérateur *ou exclusif* (XOR) [26]. De plus, Akiss se parallélise facilement, et l'utilisation de plusieurs processeurs en parallèle permet d'augmenter notablement ses performances. Plus récemment, et simultanément avec le travail présenté ici, l'outil Deepsec [46] a été développé. Cheval, Kremer et Rakotonirina s'appuient sur les résultats de complexité présentés pour établir une procédure de décision. En particulier, Deepsec construit en pratique les arbres de partition qui sont utilisés pour démontrer la borne supérieure de complexité. L'outil repose sur le formalisme des clauses de Horn et permet de modéliser une large classe de primitives. Deepsec permet également de bénéficier du parallélisme et subsume APTE. En particulier, son algèbre de processus contient les canaux privés et les branches *else*.

Une autre manière d'envisager le problème consiste à démontrer la décidabilité pour un nombre arbitraire de sessions en restreignant la classe de protocoles. Ainsi, Chrétien, Cortier et Delaune [54] ont démontré qu'à l'intérieur d'une classe restreinte de protocoles, l'équivalence se réduisait à l'équivalence de langages dans les automates à pile. Mais le plus souvent, la décidabilité est obtenue grâce à des résultats de petites attaques valables également dans un cadre plus général : Chrétien, Cortier et Delaune [51] ont démontré que pour une classe de protocoles contenant les protocoles marqués, il existe une attaque si et seulement si il existe une attaque bien typée. Ils en déduisent un résultat de décidabilité pour des protocoles sans nonces, pour un nombre arbitraire de sessions. Ils montrent également [52] qu'il est possible d'abstraire les nonces, c'est-à-dire que, pour obtenir la correction, il n'est pas nécessaire de faire appel à un nom frais à chaque génération de nonce. En d'autres termes, si un protocole est prouvé sûr avec un nombre fini de nonce, il est également sûr dans le scénario sans approximation. Il est donc possible, d'après le résultat de décidabilité précédent, de prouver l'équivalence pour un nombre arbitraire de sessions. Mais une application plus fine du résultat de typage [53] permet de montrer la décidabilité pour des protocoles avec nonces, en imposant une condition supplémentaire (le graphe de dépendance doit être acyclique).

Il est également nécessaire d'optimiser l'efficacité des procédures de décisions. La réduction d'ordre partiel [27] constitue probablement la plus décisive en la matière. Ce résultat permet de réduire drastiquement le nombre d'exécutions à considérer pour des protocoles déterministes. Par exemple, il est souvent possible d'exiger que tous les messages disponibles soient émis avant d'exécuter une réception supplémentaire.

Contributions

Cette thèse présente une nouvelle procédure de décision pour l'équivalence pour un nombre borné de sessions. La procédure repose sur le codage du problème sous forme de planification, puis comme formule SAT. Cette démarche est inspirée par l'outil SATMC [21] pour l'accessibilité. Elle repose sur deux résultats de réduction :

— Un résultat de typage : s'il existe une attaque, il existe une attaque bien typée.

— Une borne sur le nombre de constantes : s'il existe une attaque bien typée, il en existe une dans laquelle l'attaquant n'utilise que trois constantes.

Le reste de cette section décrit plus précisément ces contributions.

Résultats de typage. Une manière de restreindre l'espace de recherche pour les attaques consiste à supposer que les exécutions respectent un système de type. Cependant, cette hypothèse est bien trop forte en général, puisqu'un attaquant n'hésitera pas à envoyer des messages du mauvais type. Cependant, dans certains cas, il est possible de montrer que cette hypothèse est correcte, comme le montrent plusieurs résultats [76, 91, 15, 51, 11] de la littérature. Les hypothèses sur les protocoles et les primitives considérés sont diverses, mais à l'exception des travaux de Chrétien, Cortier et Delaune [51], aucun ne porte sur l'équivalence. Nous étendons ce résultat. La démonstration reposait sur un algorithme de décision pour un nombre de sessions borné. Cette preuve a été intégralement reformulée et simplifiée. De plus, le résultat précédent n'était valable que pour le chiffrement symétrique et la paire, ce qui pesait sur tous les résultats construits dessus, et en particulier le résultat de décision [53]. Nous démontrons que le résultat reste valable pour une classe paramétrique de primitives, qui contient les chiffrements symétrique et asymétrique, les tuples, la signature numérique et les fonctions de hachage. Les protocoles considérés sont *déterministes*, c'est-à-dire que l'attaquant doit être capable de savoir avec quel processus il communique à tout moment. De plus, l'algèbre de processus est restreinte et ne contient pas de branches *else*. Chaque résultat de typage repose sur un système de marquage plus ou moins contraignant. Dans notre

cas, la notion de *conformité* d'un protocole à un système de types permet de s'appuyer sur la structure du protocole pour se passer de *marques* (*tags*) explicites. Cette contribution nous permet en particulier de borner la profondeur des termes impliqués dans une attaque. De plus, des contre-exemples sont présentés pour justifier les hypothèses qui restreignent les primitives, ce qui démontre que toute extension nécessiterait des développements techniques supplémentaires. Nous obtenons également un résultat de typage pour l'accessibilité, pour une classe de primitives incomparable avec celle d'Almousa *et al.* [11]. Tous les autres résultats se limitent à des primitives fixées.

Une borne sur le nombre de constantes. Comme nous considérons un nombre fini de sessions et comme la profondeur des termes a été bornée, il est facile de voir que le nombre de constantes nécessaires à l'attaquant est borné. Cependant, ce nombre peut croître très vite avec la taille du protocole, et ralentir fortement toute procédure reposant sur l'énumération des exécutions concrètes d'un protocole. Pour cette raison, nous démontrons que pour notre algèbre de processus et pour une théorie sans paires critiques, l'attaquant n'a besoin que de trois constantes en plus de celles qui apparaissent explicitement dans le protocole. Pour obtenir un tel résultat, nous devons relâcher l'hypothèse de typage sur les traces. Cette contribution découle directement de nos travaux sur le nombre d'agents [60]. Nous avons démontré que, pour un ensemble de primitives bien plus larges, et pour des processus déterministes dont les branches *else* sont contrôlées, il est toujours possible de calculer une borne sur le nombre d'agents. Plus précisément, il existe une attaque si et seulement si il existe une attaque faisant appel à un nombre d'agents inférieurs à la borne. De plus, nous avons montré que pour des protocoles non déterministes, ou contenant des branches *else* arbitraires, ou pour des théories qui sortent de notre cadre, une telle borne n'est plus calculable (calculer une telle borne sur les agents revenait à calculer une borne sur la longueur des solutions du problème de correspondance de Post). Ces contre-exemples sont également valables dans le cas des constantes, ce qui impose des limites à notre démarche. Pour préserver la cohérence du formalisme, nous avons choisi de ne pas présenter le résultat de réduction du nombre d'agents [60] dans cette thèse. Au contraire, le résultat qui porte sur les constantes, et qui s'appuie sur des arguments très similaires, ainsi que les contre-exemples sont exposés.

Planification : traduction et résolution. Armando et Compagna [21] utilisent un problème de bas niveau, appelé planification, pour obtenir une surapproximation finie et close de l'exécution des protocoles dans le cadre de l'accessibilité. Lorsqu'il n'y a pas d'attaque, cette surapproximation suffit le plus souvent à le démontrer. Si toutefois une attaque est détectée, le problème est codé comme formule SAT. Un solveur permet alors de confirmer l'attaque ou de prouver le protocole. Cette manière de procéder a été fructueuse : non seulement SATMC est relativement rapide, mais il a permis de découvrir des attaques sur des protocoles réellement déployés, en particulier sur le *Single Sign On* de Google [18]. Pour cette raison, nous suivons la méthode de SATMC et nous codons une surapproximation du problème d'équivalence sous forme de planification, puis nous éliminons les fausses attaques grâce au solveur SAT. Le cas de l'équivalence est notoirement plus complexe. En particulier, il ne s'agit pas seulement de modéliser l'exécution d'un protocole, mais aussi de rendre compte des échecs d'exécution, puisqu'ils peuvent démontrer la non-équivalence. Par ailleurs, dans le cas de l'équivalence, un système de types ne borne directement l'exécution que d'un seul protocole. Il faut donc contrôler le second de manière indirecte.

Notre procédure permet de décider l'équivalence pour les primitives classiques (chiffrements symétrique et asymétrique, tuples, signature numérique et fonction de hachage). La procédure complète comprend la traduction de l'équivalence, la résolution du problème d'équivalence grâce à l'algorithme du graphe de planification [39], ainsi que la traduction, si nécessaire, comme formule SAT, en suivant la méthode d'Ernst, Millstein et Weld [72]. La résolution du problème SAT n'est pas un élément critique, et n'importe quel solveur récent répond en moins d'une seconde à toutes

les requêtes. Pour l'efficacité, les enjeux se trouvent plutôt du côté de la planification. Cependant, la manière dont nous utilisons le problème ne permet pas d'adapter directement l'état de l'art. En effet, la planification ne sert pas uniquement à résoudre le problème, mais surtout à en donner une représentation close. Dans le cas de l'équivalence, nous ne connaissons pas les règles du problème avant la fin du calcul.

Implémentation. Ces travaux ont permis la mise au point de l'outil SAT-Equiv [3] qui utilise le solveur Minisat [71]. La terminaison de minisat n'est pas garantie, comme pour tous les SAT-solveurs les plus performants, mais pour tous nos exemples, une réponse est obtenue en moins d'une seconde. SAT-Equiv a été implémenté en OCaml, et contient environ 5000 lignes de code. Nous effectuons une comparaison avec les autres outils pour l'équivalence bornée, sur huit protocoles de la littérature qui entrent dans notre classe relativement restreinte (protocoles déterministes sans branches *else*). En 24 heures, SAT-Equiv analyse au moins deux à trois fois plus de sessions que les autres outils, et parfois beaucoup plus.

Plan de la thèse

Le chapitre 1 présente le modèle général. Nous ajoutons quelques restrictions aux représentations classiques des messages en termes d'algèbre de termes et des protocoles comme algèbre de processus. En particulier, les termes dont les clés ne sont pas atomiques ne sont pas transmis sur le réseau. Cette hypothèse permet de simplifier les calculs que doit réaliser l'attaquant : il lui suffit de procéder par analyse-synthèse. Elle repose sur l'idée que les termes utilisés comme clés ne peuvent pas être arbitraires. Le chapitre 2 présente les résultats de typage pour l'accessibilité et l'équivalence. Dans un premier temps, nous énonçons les hypothèses nécessaires pour nos résultats. Ensuite, nous présentons le théorème de typage pour l'accessibilité, qui est une conséquence de développements techniques nécessaires pour la preuve du théorème de typage pour l'équivalence exposé ensuite. Dans le chapitre 3 nous démontrons que trois constantes suffisent à l'attaquant, tout en préservant l'essentiel du résultat de typage. Comme la procédure de décision s'appuie sur les problèmes de planification, nous les présentons dans le chapitre 4, qui contient également l'algorithme du graphe de planification, ainsi que la traduction du problème comme formule SAT. Le chapitre 5 expose la méthode de traduction de l'équivalence sous forme de planification, d'abord pour un attaquant passif, puis pour un attaquant actif. Enfin, le chapitre 6 présente dans les détails l'algorithme utilisé ainsi que l'implémentation. Diverses études de cas sont réalisées, ainsi qu'une comparaison avec les autres outils pour l'équivalence bornée. Cette thèse se terminera alors par une conclusion générale. Une bibliographie, un index des termes techniques et une annexe contenant les preuves laissées de côté sont fournis en supplément.

Publications

Les travaux présentés dans cette thèse ont fait l'objet de plusieurs publications.

— Les résultats du chapitre 3 sont fortement inspirés par un article [60] présenté en 2016 à la conférence *Principles Of Security and Trust* (POST'16), qui a obtenu le prix EASST du meilleur article présenté à ETAPS.

— Une première version [61] des résulats du chapitre 5 a été exposée à la conférence *Computer Security Foundations* de 2017 (CSF'2017),

— Diverses extensions de ces résultats (primitives asymétriques, phases, terminaison) ont été présentées par un article [62] accepté à la conférence *European Symposium On Research In Computer Science* de 2018 (ESORICS'18).

— Un article contenant les résultats du chapitre 5 a été soumis à une revue et est en cours de relecture.

1 Modèle

Ce chapitre définit le modèle qui sera utilisé tout au long de cette thèse. Dolev et Yao [68] ont introduit le premier modèle symbolique, où l'attaquant contrôle le réseau et où les primitives cryptographiques sont purement abstraites. Plus récemment, Abadi et Fournet [7] ont posé les bases du pi-calcul appliqué, particulièrement adapté à l'étude des protocoles cryptographiques, et ont transcrit les équivalences dans ce contexte. Dans les fondements théoriques de ProVerif [34], Blanchet introduit la séparation entre symboles constructeurs et destructeurs ; seuls les termes constructeurs sont échangés sur le réseau.

Le modèle présenté ici s'appuie sur ces travaux. D'abord, la section 1.1 expose l'algèbre de termes, qui formalise les messages et les calculs de l'attaquant. De plus, des restrictions supplémentaires sur les clés et les opérations cryptographiques. Ensuite, l'algèbre de processus de la section 1.2 constitue une variante de celle du pi-calcul appliqué où les déchiffrements sont effectués par filtrage à la réception du message. Cette variante provient des travaux de Chrétien, Cortier et Delaune [52, 51, 53]. Dans la section 1.3, l'indistinguabilité entre deux protocoles est formalisée à travers la définition de l'équivalence de trace et de l'approximation que nous utiliserons. Enfin, la section 1.4 introduit des hypothèses qui seront vérifiées dans la suite, et qui garantissent que l'équivalence de trace coïncide avec son approximation.

1.1 Algèbre de termes

Pour représenter les communications entre les protocoles, la première étape consiste à modéliser les messages, qui correspondent à des données sur lesquelles des opérations cryptographiques sont appliquées. Intuitivement, les messages sont construits à partir de textes en clair, de nonces et de clés, modélisés par les données exposées en section 1.1.1 et d'opérations cryptographiques représentées par des symboles de fonctions présentés en section 1.1.2. Une algèbre de termes construite sur ces données et ces symboles de fonctions modélise les messages, mais aussi les calculs de l'attaquant.

Souvent, un processus attend un message respectant un certain motif. L'adéquation entre le motif et le message réellement reçu est formalisée, en section 1.1.3, par l'unification, qui permet également d'exprimer les restrictions sur les termes à travers les sortes (section 1.1.4) et les contours (section 1.1.5). Ces restrictions portent sur les messages échangés sur le réseau, qui sont décrits formellement en section 1.1.6, en même temps que les recettes qui représentent les opérations réalisées par l'attaquant. Les relations entre les primitives cryptographiques représentées par les symboles de fonction sont traduites sous forme de règles de réécriture dans la section 1.1.7, où plusieurs exemples illustrent les possibilités du modèle. Son étendue et ses limites sont discutées en section 1.1.8.

1.1.1 Données

Les ensembles de variables $\mathcal{X}$ et $\mathcal{W}$, infinis et disjoints, permettent de représenter des messages. Plus précisément, les variables de $\mathcal{X}$ seront utilisées pour modéliser un message arbitraire qui doit être reçu par un processus tandis que les variables de $\mathcal{W}$ symboliseront les messages connus par l'attaquant. Les *noms* de l'ensemble infini $\mathcal{N}$ symboliseront les informations inconnues de l'attaquant avant l'exécution du programme, comme les clés privées et les nonces.

Les informations connues par l'attaquant (telles que les clés de l'attaquant ou les informations publiques) seront représentées par Σ_0^-. À ces constantes, nous devons ajouter celles de Σ_{fresh}, qui servent à abstraire les messages dans le chapitre 2. Σ_{fresh} se décompose en $\Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ et $\Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$, qui sont deux ensembles infinis de constantes, soit $\Sigma_{\mathsf{fresh}} = \Sigma_{\mathsf{fresh}}^{\mathsf{atom}} \uplus \Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$. Dans les développements techniques du chapitre 2, les constantes de $\Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ seront utilisées pour remplacer des données tandis que les constantes de $\Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$ remplaceront des termes construits. Ces constantes seront donc considérées comme des constantes fraîches qui n'apparaissent ni dans le protocole, ni dans l'exécution, et nous devrons donc toujours distinguer entre les exécutions qui peuvent contenir ces constantes (après remplacement) et celles qui ne peuvent pas les contenir et rapport auxquelles elles sont fraîches (avant remplacement). C'est pourquoi nous définissons l'ensemble de constantes publiques $\Sigma_0^+ = \Sigma_0^- \uplus \Sigma_{\mathsf{fresh}}$. Les exécutions normales n'utilisent que les constantes de Σ_0^-, tandis que les exécutions de remplacement utilisent les constantes de Σ_0^+ ; et dans la suite de ce chapitre nous nous servirons d'un ensemble paramétrique de constantes Σ_0. Nous devrons également démontrer dans le chapitre 2 que ces deux modèles coïncident.

Nous dirons que d est une *donnée* si c'est une variable, un nom ou une constante publique, c'est-à-dire un élément de $\mathcal{X} \uplus \mathcal{W} \uplus \Sigma_0^+ \uplus \mathcal{N}$. L'exemple suivant présente le rôle des données de manière générale.

Exemple 1.1. *Supposons qu'Alice envoie un nonce frais à Bob, tandis qu'Ève observe l'échange. Du côté de Bob, le message attendu sera représenté par la variable $x \in \mathcal{X}$. Du côté d'Alice, le message envoyé sera représenté par le nom n, puisque le nonce n'est pas supposé être connu par Ève au départ. Lorsque le message sera émis par Alice, Ève pourra le lire et la variable $\mathsf{w} \in \mathcal{W}$ représentera ce message. Ensuite, Ève aura le choix entre transmettre le nonce à Bob (pour qui la variable x correspondra alors au nonce n) ou lui envoyer une autre donnée $a \in \Sigma_0^-$ (et dans ce cas, la variable x correspondra alors à a).*

1.1.2 Symboles de fonctions

Les opérations cryptographiques réalisées par l'attaquant sont modélisées par des symboles de fonction. Un ensemble Σ de symboles de fonctions, avec leur arité, est appelé *signature*. Une signature est divisée en symboles *constructeurs* (dont l'ensemble est Σ_c) et symboles *destructeurs* (Σ_d). On note donc $\Sigma = \Sigma_d \uplus \Sigma_c$, et $\mathcal{T}(\Sigma, D)$ l'ensemble des termes construits sur un ensemble de données D et une signature Σ. Les *termes constructeurs* sur D sont les termes de $\mathcal{T}(\Sigma_c, D)$. Les termes de $\mathcal{T}(\Sigma, D) \setminus D$ sont appelés *termes composés*.

L'exemple suivant présente une signature très simple pour illustrer quelques concepts. D'autres symboles de fonction plus complexes seront ajoutés dans la suite.

Exemple 1.2. *Posons $\Sigma_c = \{\mathsf{senc}/2; \mathsf{hash}/1; \langle\cdot,\cdot\rangle/2\}$ et $\Sigma_d = \{\mathsf{sdec}/2; \mathsf{proj}_1/1; \mathsf{proj}_2/1\}$. senc modélise le chiffrement symétrique, tandis que sdec représente le déchiffrement symétrique. hash désigne une fonction de hachage, et $\langle\cdot,\cdot\rangle$ la construction de paires. proj_1 et proj_2 sont les deux opérations de projection des paires.*

$\Sigma = \Sigma_c \uplus \Sigma_d$ est une signature, et, avec $n, k, k' \in \mathcal{N}$, $t = \mathsf{senc}(\mathsf{hash}(n), \mathsf{proj}_1(\langle k, k'\rangle))$ est un terme de $\mathcal{T}(\Sigma, \mathcal{N})$. $t' = \mathsf{senc}(\mathsf{hash}(x), a)$, avec $a \in \Sigma_0^-$ et $x \in \mathcal{X}$, est un terme constructeur de

$\mathcal{T}(\Sigma_c, \Sigma_0^- \uplus \mathcal{X})$.

Étant donné un terme t, nous définissons récursivement les *positions* dans t : la position ϵ désigne t ; et si $t = \mathsf{f}(t_1, \ldots, t_n)$ et, pour $1 \leqslant i \leqslant n$, la position p_i désigne un terme t'_i dans t_i, alors la position $i.p_i$ désigne t'_i dans t_i. Dans ce cas, on dit que $i.p_i$ est une position de t.

Les sous-termes $St(t)$ d'un terme t sont définis par récurrence : si t est une donnée, alors $St(t) = \{t\}$ et, pour $t = \mathsf{f}(t_1, \ldots, t_n)$, $St(t) = \{t\} \cup (\cup_i St(t_i))$. Chaque position p de t désigne un sous-terme de t noté $t|_p$. $t[t']_p$ désigne le terme t où $t|_p$ a été remplacé par t'. Formellement, on définit $t[t']_p$ par récurrence. Si $p = \epsilon$ alors $t[t']_p = t'$. S'il existe un entier i tel que $p = i.p_i$ est une position de t, alors il existe $t_1, \ldots, t_n$ tels que $t = \mathsf{f}(t_1, \ldots, t_n)$. Dans ce cas, $t[t']_p = \mathsf{f}(t_1, \ldots, t_i[t']_{p_i}, \ldots, t_n)$.

Soient $t_1, \ldots, t_n$ des termes, et f un symbole de fonction. Posons $t = \mathsf{f}(t_1, \ldots, t_n)$. La *racine* de t, notée $\mathsf{root}(t)$, est son symbole de fonction en position ϵ, c'est-à-dire f.

Exemple 1.3. *Reprenons la signature Σ de l'exemple 1.2 et les termes t, t'. Les positions de t sont $\{\epsilon, 1, 1.1, 2, 2.1, 2.2\}$ et celles de t' sont $\{\epsilon, 1, 1.1, 2\}$. On a $St(t) = \{t = t|_\epsilon, \mathsf{hash}(n) = t|_1, n = t|_{1.1}, \mathsf{proj}_1(\langle k, k' \rangle) = t|_2, k = t_{2.1}, k' = t|_{2.2}\}$ et $St(t') = \{t' = t'|_\epsilon, \mathsf{hash}(x) = t'|_1, x = t'|_{1.1}, a = t'|_2\}$. On a également $t[t']_1 = \mathsf{senc}(\mathsf{senc}(\mathsf{hash}(x), a), \mathsf{proj}_1(\langle k, k' \rangle))$ et $\mathsf{root}(t|_1) = \mathsf{hash}$.*

1.1.3 Substitutions et unification

Soit u un terme. u est *clos* s'il ne contient aucune variable. Les variables qui apparaissent dans u sont notées $vars(u)$, c'est-à-dire $vars(u) = (\mathcal{X} \uplus \mathcal{W}) \cap St(u)$. Pour un terme clos, on a donc $vars(u) = \emptyset$. Une *substitution* σ est un ensemble fini de paires $d \triangleright t$ où d est une donnée et t un terme, avec la propriété que pour tout d, il n'existe pas deux termes $t \neq t'$ tels que $d \triangleright t$ et $d \triangleright t'$. On appelle *domaine* de σ l'ensemble $dom(\sigma) = \{d \mid d \triangleright t \in \sigma\}$ et *image* de σ l'ensemble $img(\sigma) = \{t \mid d \triangleright t \in \sigma\}$.

Soit u un terme. On définit récursivement l'application de σ à u : Si $u \in dom(\sigma)$, alors $u\sigma = t$ où $u \triangleright t \in \sigma$. Si u est une donnée, mais $u \notin dom(\sigma)$, alors $u\sigma = u$. S'il existe f et $u_1, \ldots, u_n$ tels que $u = \mathsf{f}(u_1, \ldots, u_n)$, $u\sigma = \mathsf{f}(u_1\sigma, \ldots, u_n\sigma)$.

Deux termes u_1 et u_2 sont *unifiables sur* X (ou simplement *unifiables* quand le contexte est clair) s'il existe une substitution σ avec $dom(\sigma) \subseteq X$ et $u_1\sigma = u_2\sigma$. La substitution σ est un *unificateur* de u_1 et u_2.

Si σ et σ' sont deux substitutions qui unifient u_1 et u_2 sur un même ensemble X, alors on dit que σ est *plus générale* que σ' s'il existe une substitution τ telle que $\sigma' = \sigma \circ \tau$. Quand deux termes u_1 et u_2 sont unifiables, il existe des unificateurs les plus généraux, qui sont toutes égales à renommage des variables près. On note $mgu(u_1, u_2)$ l'une de ces substitutions, choisie arbitrairement. Lorsque u_1 ou u_2 est clos, il n'y a qu'un seul unificateur le plus général, même sans tenir compte des renommages. On dit alors que $mgu(u_1, u_2)$ est un *filtrage*. Lorsque u_2 est un terme clos, le problème de savoir s'il existe une substitution σ telle que $u_1\sigma = u_2$ est appelé *problème de filtrage*.

Exemple 1.4. *Reprenons la signature Σ de l'exemple 1.2 et les termes t et t'. t et t' ne sont pas unifiables sur $\mathcal{X}$ car $t|_2 \neq t'|_2$ et $t|_2$ et $t'|_2$ sont clos. Les termes $u = \mathsf{senc}(\mathsf{hash}(x), \mathsf{proj}_1(\langle k, k' \rangle))$ et $v = \mathsf{senc}(\mathsf{hash}(n), \mathsf{proj}_1(y))$ sont unifiables. La substitution $\{x \triangleright n; y \triangleright \langle k, k' \rangle\}$ est l'unique unificateur le plus général de u et v. $\langle x, y \rangle$ et $\langle y, x \rangle$ ont deux unificateurs les plus généraux : $\sigma_x = \{x \triangleright y\}$ et $\sigma_y = \{y \triangleright x\}$. σ_x et σ_y sont égaux à renommage près.*

1.1.4 Sortes

Dans les développements techniques, nous aurons besoin de restreindre l'algèbre de terme, en particulier pour que les résultats du chapitre 2 soient valables. L'une des restrictions concerne les clés : le résultat n'a été prouvé que si quand le protocole n'accepte que des termes dont les clés sont atomiques.

On diposera donc de deux *sortes*, atom, pour représenter les messages acceptés en position de clé, et $\mathsf{bitstring}$, pour modéliser des messages arbitraires. Un message de sorte atom pourra toujours être utilisé quand on attend un terme de sorte $\mathsf{bitstring}$, mais l'inverse n'est pas vrai. Parmi les données, les noms de $\mathcal{N}$ et les constantes de $\Sigma_0^- \uplus \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ sont de sorte atom. Ainsi, les constantes de $\Sigma_0^- \uplus \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ sont appelées *constantes atomiques*. Au contraire, les constantes de $\Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$ seront de sorte $\mathsf{bitstring}$. Elles serviront à représenter synthétiquement des messages composés. Les variables de $\mathcal{X}$ seront aussi de sorte atom, mais elles pourront être instanciées par des termes arbitraires (les substitutions ne respectent pas les sortes).

Chaque symbole constructeur $\mathsf{f} \in \Sigma_c$ dispose d'une sorte $\mathsf{f} : s_1 \times \cdots \times s_n \to \mathsf{bitstring}$ où n est l'arité de f et $s_i \in \{\mathsf{atom}; \mathsf{bitstring}\}$ pour chaque $1 \leqslant i \leqslant n$. Ainsi, un terme $\mathsf{f}(t_1, \ldots, t_n)$ sera toujours de sorte $\mathsf{bitstring}$. Les *positions de clé* d'un terme t sont les positions $p.i$ de t telles que le symbole $\mathsf{root}(t|_p)$ est de sorte $s_1 \times \cdots \times s_n \to \mathsf{bitstring}$ et $s_i = \mathsf{atom}$. On dira qu'un terme t *respecte les sortes* si, pour chaque position de clé p de t, $t|_p$ est de sorte atom.

L'exemple suivant introduit quelques nouveaux exemples de symboles constructeurs.

Exemple 1.5. *Posons* $\Sigma_c = \{\mathsf{senc}/2; \mathsf{aenc}/2; \mathsf{pub}/1; \mathsf{hash}/1; \langle\cdot,\cdot\rangle/2\}$*, avec les sortes :*

$$\begin{array}{rcl} \mathsf{senc} & : & \mathsf{bitstring} \times \mathsf{atom} \to \mathsf{bitstring}; \\ \mathsf{aenc} & : & \mathsf{bitstring} \times \mathsf{bitstring} \to \mathsf{bitstring}; \\ \mathsf{pub} & : & \mathsf{atom} \to \mathsf{bitstring}; \\ \mathsf{hash} & : & \mathsf{bitstring} \to \mathsf{bitstring}; \\ \langle\cdot,\cdot\rangle & : & \mathsf{bitstring} \times \mathsf{bitstring} \to \mathsf{bitstring} \end{array}$$

aenc *permet de modéliser le chiffrement asymétrique (non randomisé), et* pub *permet d'obtenir la clé publique associée à une clé privée.*

Soient $a \in \Sigma_0^-, n \in \mathcal{N}, x \in \mathcal{X}$ *et* $b \in \Sigma_{\mathsf{bitstring}}^{\mathsf{fresh}}$*. Les termes constructeurs* $\mathsf{senc}(a, x)$, $\mathsf{aenc}(a, \mathsf{pub}(n))$, $\mathsf{aenc}(a, \langle n, n\rangle)$, $\mathsf{hash}(\langle a, n\rangle)$ *et* $\langle\langle x, a\rangle, b\rangle$ *respectent tous les sortes.*

Au contraire, les termes $\mathsf{senc}(a, b)$, $\mathsf{senc}(a, \mathsf{pub}(n))$, $\mathsf{pub}(\langle a, a\rangle)$ *et* $\langle\mathsf{pub}(b), a\rangle$ *ne respectent pas les sortes.*

1.1.5 Contours

Une autre hypothèse sera nécessaire pour démontrer le résultat du chapitre 2. À chaque symbole constructeur f, nous associerons un terme $\mathsf{sh}_\mathsf{f} \in \mathcal{T}(\Sigma_c, \mathcal{X})$ linéaire (c'est-à-dire dans lequel chaque variable n'apparaît qu'au plus un fois), que nous appellerons son *contour*. Les contours seront supposés *compatibles*, c'est-à-dire que tout sous-terme d'un contour sera soit une variable, soit un contour. Plus formellement, pour tout $\mathsf{f} \in \Sigma_c$, pour tout $t \in St(\mathsf{sh}_\mathsf{f})$, $t = \mathsf{sh}_{\mathsf{root}(t)}$.

On dira qu'un terme $t \in \mathcal{T}(\Sigma_c, \Sigma_0^+ \uplus \mathcal{X})$ *respecte les contours* si tous ses sous-termes suivent le contour de leur symbole de tête, c'est-à-dire si pour tout $t' \in St(t)$, quand $t' = \mathsf{g}(t_1, \ldots, t_n)$ pour un certain n et un certain g d'arité n, il existe une substitution σ avec $dom(\sigma) \subseteq \mathcal{X}$ telle que $t' = \mathsf{sh}_\mathsf{g}\sigma$.

Pour un symbole f donné, l'existence de termes t avec $\mathsf{f} = \mathsf{root}(t)$ qui respectent les contours impose certaines conditions sur sh_f, en particulier que $\mathsf{root}(\mathsf{sh}_\mathsf{f}) = \mathsf{f}$ et que f n'apparaisse pas ailleurs dans le contour. Seuls des termes qui respectent les contours seront échangés sur le réseau, donc nous supposerons que tous les contours vérifient ces deux hypothèses.

Dans le cas où le contour d'un symbole f est réduit à $\mathsf{f}(x_1, \ldots, x_n)$ pour $x_1, \ldots, x_n$ des variables, il n'apporte aucune restriction supplémentaire. On dira qu'il est *trivial*.

L'exemple 1.6 prolonge l'exemple 1.5 en donnant des exemples de contours et de termes qui respectent ou non les contours, tout en rappelant quels termes respectent ou non les sortes.

Exemple 1.6. *Reprenons la signature Σ_c de l'exemple 1.5, avec les contours :*

$$\begin{array}{rcl} \mathsf{senc} & : & \mathsf{senc}(y, x); \\ \mathsf{aenc} & : & \mathsf{aenc}(y, \mathsf{pub}(x)); \\ \mathsf{pub} & : & \mathsf{pub}(x); \\ \mathsf{hash} & : & \mathsf{hash}(y); \\ \langle \cdot, \cdot \rangle & : & \langle y, y' \rangle \end{array}$$

où $x, y, y' \in \mathcal{X}$. La variable x représente les positions de clé, tandis que les autres positions contiennent y, y'. Pour $a \in \Sigma_0^-$, $\mathsf{aenc}(\mathsf{senc}(y, a), \mathsf{pub}(\langle a, a \rangle))$ respecte les contours (mais pas les sortes). Les termes $t = \mathsf{aenc}(\mathsf{senc}(y, a), \mathsf{pub}(a))$ et $t' = \mathsf{aenc}(\mathsf{senc}(y, a), a)$ respectent les sortes, mais seul t respecte les contours. Tous les termes ne contenant pas aenc *respectent les contours car $\mathsf{sh}_{\mathsf{aenc}}$ est le seul contour non trivial de notre exemple.*

1.1.6 Messages et recettes

Soient A un ensemble de données et Σ_c un ensemble de symboles constructeurs. On notera $\mathcal{T}_0(\Sigma_c, A)$ l'ensemble des termes constructeurs construits sur A qui respectent les sortes et les contours.

Soit Σ_0 un ensemble de constantes. Les Σ_0-messages sont les termes constructeurs clos qui respectent les sortes et les contours, c'est-à-dire $\mathcal{M}_{\Sigma_0} = \mathcal{T}_0(\Sigma_c, \Sigma_0 \uplus \mathcal{N})$. Seuls les messages seront échangés sur le réseau. En particulier, les agents modélisés par les processus n'accepteront pas en entrée des termes qui ne seront pas des messages, et ils n'émettront pas non plus de termes qui ne sont pas des messages.

Lorsque l'attaquant dispose d'informations émises sur le réseau, représentées par les variables de $\mathcal{W}$, il peut lui être utile d'effectuer des calculs, qui sont représentés par des *recettes*. Plus précisément, on dira que R est une Σ_0-recette si $R \in \mathcal{T}(\Sigma, \mathcal{W} \cup \Sigma_0)$. Ainsi, si $s \in \Sigma_0^-$ représente une clé de l'attaquant et w un message qu'il a appris, le terme $\mathsf{sdec}(\mathsf{w}, s)$ représente le résultat de la tentative de déchiffrement de w par s.

1.1.7 Théories

Nous avons modélisé les données, les opérations cryptographiques et les messages échangés sur le réseau. Il reste à définir les relations entre les primitives, qui seront représentées par des règles de réécriture. Plus précisément, soit $\Sigma = \Sigma_c \uplus \Sigma_d$ une signature. Une règle de réécriture est de la forme $\ell \to r$ avec $\ell = \mathsf{des}(t_1, \ldots, t_n)$ et $r = t_0$, où $\mathsf{des} \in \Sigma_d$ est d'arité n, et pour chaque i, $t_i \in \mathcal{T}_0(\Sigma_c, \mathcal{N} \uplus \Sigma_0^- \uplus \mathcal{X})$. Soit $\mathcal{R}$ un ensemble de règles de réécriture. $\mathcal{R}$ définit un système de réécriture : pour tout terme u, on note $u \to v$ s'il existe p une position de u, σ une substitution avec $dom(\sigma) = vars(\ell)$, $\ell \to r$ dans $\mathcal{R}$ tels que $u|_p = \mathsf{des}(t_1, \ldots, t_n)\sigma$ avec $t_i\sigma$ un Σ_0^+-message pour chaque i, $r\sigma$ est un Σ_0^+-message et $v = u[r\sigma]_p$. Ainsi, si un terme t s'unifie avec ℓ sans que chaque $t_i\sigma$ soit un message, il ne se réduit pas.

On étend $\to$ à sa clôture réflexive et transitive. On dit que $(\Sigma, \mathcal{R})$ est une *théorie à constructeurs*. Le système de réécriture associé à chaque théorie à constructeurs termine, puisque chaque réécriture fait disparaître un destructeur. Ainsi, un système de réécriture est *convergent* si, et seulement si, il est *confluent*, c'est-à-dire si et seulement si, chaque fois que $t \to u$ et $t \to v$, il existe w tel

que $u \to w$ et $v \to w$. On dit qu'une théorie à constructeurs $(\Sigma, \mathcal{R})$ admet une *paire critique* s'il existe deux règles $\ell \to r$ et $\ell' \to r'$ telles que ℓ et ℓ' sont unifiables. Il est clair qu'une théorie à constructeurs qui n'admet pas de paire critique est convergente. Pour une théorie à constructeurs $(\Sigma, \mathcal{R})$ convergente, et pour tout terme t, on notera $t{\downarrow}$ sa *forme normale*.

Dans toute la suite, tous les résultats positifs vaudront uniquement pour des théories à constructeurs convergentes. Les exemples suivants présentent quelques primitives qui peuvent être facilement modélisées par des théories à constructeurs pour lesquelles nous pouvons donner un sens aux hypothèses sur les sortes et les contours. La liste n'est pas exhaustive.

Exemple 1.7. *Reprenons l'ensemble de symboles constructeurs Σ_c de l'exemple 1.5, et ajoutons les destructeurs $\Sigma_d = \{\mathsf{adec}, \mathsf{sdec}, \mathsf{proj}_1, \mathsf{proj}_2\}$. Les règles de réécriture associées sont :*

$$\begin{array}{rcl} \mathsf{adec}(\mathsf{aenc}(y, \mathsf{pub}(x)), x) & \to & y \\ \mathsf{sdec}(\mathsf{senc}(y, x), x) & \to & y \\ \mathsf{proj}_i(\langle x_1, x_2 \rangle) & \to & x_i \text{ pour } i \in \{1, 2\} \end{array}$$

avec $x, x_1, x_2, y \in \mathcal{X}$.

Posons $t = \mathsf{adec}(\mathsf{aenc}(\mathsf{proj}_1(\langle n, m \rangle), \mathsf{pub}(s)), s)$ avec $n, m, s \in \mathcal{N}$. Le terme t n'est pas un Σ_0^+-message car ce n'est pas un terme constructeur, mais sa forme normale $t{\downarrow} = n$ est un message. Dans certains cas, il peut être intéressant de donner à l'attaquant la possibilité de savoir si deux textes chiffrés ont la même clé ou non, ou si un texte chiffré correspond bien à une clé publique donnée. Pour ce faire, nous pouvons compléter Σ_d par $\Sigma'_d = \{\mathsf{samekey}/2, \mathsf{rightkey}/2\}$, ajouter un constructeur $\mathsf{ok}/0$ supplémentaire et considérer les règles de réécriture suivantes, avec $x, y, y' \in \mathcal{X}$.

$$\begin{array}{rcl} \mathsf{samekey}(\mathsf{aenc}(y, \mathsf{pub}(x)), \mathsf{aenc}(y', \mathsf{pub}(x))) & \to & \mathsf{ok} \\ \mathsf{rightkey}(\mathsf{aenc}(y, \mathsf{pub}(x)), \mathsf{pub}(x)) & \to & \mathsf{ok} \end{array}$$

La règle de samekey *peut également être formulée pour le chiffrement symétrique (avec $x, y, y' \in \mathcal{X}$)*

$$\mathsf{samekey}(\mathsf{senc}(y, x), \mathsf{senc}(y', x)) \to \mathsf{ok}$$

Dans certains développements techniques, nous supposerons que toutes les règles ont des destructeurs distincts. Dans ce cas nous devrons distinguer entre, par exemple, $\mathsf{samekey}_{\mathsf{aenc}}$ *et* $\mathsf{samekey}_{\mathsf{senc}}$*. Le reste du temps, ces deux règles ne sont pas exclusives l'une de l'autre, d'autant qu'elles ne forment pas de paire critique car* $\mathsf{senc}(x, y)$ *et* $\mathsf{aenc}(x, y)$ *ne sont pas unifiables. La règle de* rightkey *pourrait aussi se formuler pour le chiffrement symétrique, mais comme la connaissance de la clé symétrique permet d'ouvrir le chiffrement, une telle règle n'apporterait rien.*

Il est clair que l'on peut aussi modéliser des tuples de taille arbitraire.

Exemple 1.8. *Soit $n \geqslant 2$ un entier. Les tuples de taille n sont modélisés par le symbole $\langle \cdot, \ldots, \cdot \rangle^n / n$ et l'ensemble de destructeurs $\Sigma_d = \{\mathsf{proj}_1, \ldots, \mathsf{proj}_n\}$. La sorte de $\langle\ \rangle^n$ est* $\mathsf{bitstring} \times \cdots \times \mathsf{bitstring} \to \mathsf{bitstring}$ *et son contour est trivial. Pour chaque $1 \leqslant i \leqslant n$, on a la règle*

$$\mathsf{proj}_i(\langle x_1, \ldots, x_n \rangle^n) \to x_i$$

avec $x_i \in \mathcal{X}$ pour chaque i. On obtient $\Sigma_n = \Sigma_c \cup \Sigma_d$. Comme Σ_n a été défini pour chaque n, on peut poser

$$\Sigma_n^{\mathsf{tuple}} = \cup_{i=2}^n \Sigma_i$$

L'exemple suivant montre comment modéliser les chiffrements symétrique et asymétrique randomisés.

Exemple 1.9. *Posons* $\Sigma_c = \{\mathsf{rsenc}/3, \mathsf{raenc}/3, \mathsf{pub}/2\}$ *et* $\Sigma_d = \{\mathsf{rsdec}/2, \mathsf{radec}/2\}$. rsenc *désigne le chiffrement symétrique randomisé et* raenc *le chiffrement asymétrique randomisé.* rsdec *est le déchiffrement de* rsenc *et* radec *celui de* raenc. *Nous pouvons proposer ici les sortes suivantes.*

$$\begin{array}{rcl} \mathsf{raenc} & : & \mathsf{bitstring} \times \mathsf{atom} \times \mathsf{bitstring} \to \mathsf{bitstring} \\ \mathsf{rsenc} & : & \mathsf{bitstring} \times \mathsf{atom} \times \mathsf{atom} \to \mathsf{bitstring} \\ \mathsf{pub} & : & \mathsf{atom} \to \mathsf{bitstring} \end{array}$$

Une autre possibilité aurait consisté à donner la sorte bitstring *à l'aléa. Seul le contour* $\mathsf{sh}_{\mathsf{raenc}} = \mathsf{raenc}(y, r, \mathsf{pub}(x))$ *(avec* $x, r, y \in \mathcal{X}$*) de* raenc *n'est pas trivial. Les règles de réécriture associées sont comme suit, avec* $x, r, y \in \mathcal{X}$.

$$\begin{array}{rcl} \mathsf{radec}(\mathsf{raenc}(y, r, \mathsf{pub}(x)), x) & \to & y \\ \mathsf{rsdec}(\mathsf{rsenc}(y, r, x), x) & \to & y \end{array}$$

Nous pouvons également modéliser la signature, comme détaillé dans l'exemple suivant.

Exemple 1.10. *Considérons* $\Sigma_c^{\mathsf{sign}} = \{\mathsf{sign}/2, \mathsf{vk}/1, \mathsf{ok}/0\}$. *Les sortes de ces constructeurs sont*

$$\begin{array}{rcl} \mathsf{sign} & : & \mathsf{bitstring} \times \mathsf{atom} \to \mathsf{bitstring} \\ \mathsf{vk} & : & \mathsf{atom} \to \mathsf{bitstring} \\ \mathsf{ok} & : & \to \mathsf{bitstring} \end{array}$$

et leurs contours sont tous triviaux. L'ensemble de destructeurs associés est :

$$\Sigma_d^{\mathsf{sign}} = \{\mathsf{check}/2, \mathsf{getmsg}/1\}$$

check *permet de vérifier la signature et* getmsg *est une fonction de récupération du message signé. Les règles associées sont, avec* $x, y \in \mathcal{X}$ *:*

$$\begin{array}{rcl} \mathsf{check}(\mathsf{sign}(y, x), \mathsf{vk}(x)) & \to & \mathsf{ok} \\ \mathsf{getmsg}(\mathsf{sign}(y, x)) & \to & y \end{array}$$

Enfin, les théories à constructeurs permettent aussi de modéliser certaines primitives plus exotiques. L'exemple suivant s'inspire du partage de secret de Shamir [96].

Exemple 1.11. *Posons* $\Sigma_c = \{\mathsf{onekey}/2, \mathsf{k}_1/1, \mathsf{k}_2/1\}$, *et* $\Sigma_d = \{\mathsf{get}/2\}$. onekey *est la fonction de chiffrement, et* k_i *(*$i \in \{1, 2\}$*) représente les fonctions de clé.* get *permet de révéler le message en possédant seulement une clé. Les sortes sont :*

$$\begin{array}{rcl} \mathsf{onekey} & : & \mathsf{bitstring} \times \mathsf{atom} \to \mathsf{bitstring} \\ \mathsf{k}_i & : & \mathsf{atom} \to \mathsf{bitstring} \ \textit{pour}\ i \in \{1; 2\} \end{array}$$

Les contours sont tous triviaux, et l'on pose la règle de réduction suivante pour chaque $i \in \{1, 2\}$ *(avec* $x, y \in \mathcal{X}$*)*

$$\mathsf{get}(\mathsf{onekey}(y, x), \mathsf{k}_i(x)) \to y$$

Comme get *apparaît dans deux règles, il peut être intéressant de distinguer entre deux destructeurs* get_1 *et* get_2. *Ce n'est pas nécessaire pour le moment, car les deux règles ne forment pas de paire critique.*

Il est possible de généraliser cette construction pour ne révéler le secret que si l'on possède k *clés parmi* n, *avec* $k \leqslant n$, *par exemple pour* $k = 2$ *:*

$$\mathsf{get}(\mathsf{twokeys}(y, x), \mathsf{k}_i(x), \mathsf{k}_j(x)) \to y$$

Une autre variante notable repose sur l'utilisation, pour chaque i, de la règle

$$\mathsf{get}(\mathsf{onekey}(y, x_1, \ldots, x_n), x_i) \to y$$

où onekey *est un constructeur d'arité $n+1$, et $y, x_1, \ldots, x_n \in \mathcal{X}$.*

Les primitives présentées dans les différents exemples de cette section peuvent être utilisées simultanément. En particulier, on peut confondre les constructeurs qui ont le même nom (c'est le cas de ok et pub). Lorsque les destructeurs ont le même nom, comme c'est le cas des projections associées aux différents tuples, il peut être utile de les distinguer, et on appellera proj_i^n la $i^{\text{ème}}$ projection du tuple de taille n. La confusion ne créerait pas de paires critiques, puisque les tuples de taille différente ne sont pas unifiables, mais nous retenons la variante la plus claire.

Dans la suite, nous utiliserons à plusieurs reprises la signature classique Σ_n^{std} (avec $n \in \mathbb{N}$) définie comme suit.

$$\Sigma_n^{\mathsf{std}} = \{\mathsf{senc}, \mathsf{sdec}, \mathsf{aenc}, \mathsf{adec}, \mathsf{pub}, \mathsf{check}, \mathsf{getmsg}, \mathsf{ok}, \mathsf{vk}, \mathsf{hash}\} \cup \Sigma_n^{\mathsf{tuple}}$$

$\Sigma_n^{\mathsf{tuple}}$ a été défini dans l'exemple 1.8, et les autres primitives, ainsi que leurs contours, leurs sortes et les règles de réécriture associées, notées $\mathcal{R}_n^{\mathsf{std}}$, ont été exposées dans les exemples 1.7 et 1.10. Comme chaque tuple est d'arité différente, il n'y a pas de confusion possible entre les différents symboles constructeurs $\langle\rangle^2, \ldots, \langle\rangle^n$. Parfois, pour alléger la présentation, nous noterons simplement $\langle\rangle$ pour $\langle\rangle^k$ avec $1 \leqslant k \leqslant n$.

1.1.8 Étendue et limites du modèle

Les restrictions imposées aux messages échangés sur le réseau constituent autant de limites de l'approche, pour deux raisons : d'une part, elles restreignent les théories qui peuvent être utilisées, d'autre part elles éloignent partiellement le modèle de la réalité. La section 1.1.7 a permis de montrer que les possibilités de modélisation restent importantes, et en particulier que toutes les théories classiques entrent dans ce cadre. Néanmoins, la notion de théories à constructeurs impose déjà certaines limites. Par exemple, la théorie $\mathsf{f}(\mathsf{f}(x)) \to x$ ne peut pas se modéliser comme une théorie à constructeurs. Une fonction comme le ou exclusif (XOR) ne peut donc pas entrer dans le cadre des théories à constructeurs, sauf en perdant beaucoup de ses propriétés.

La limitation aux clés atomiques s'ajoute à la précédente. Cependant, la plupart des algorithmes de cryptographie n'acceptent pas des messages arbitraires en position de clé : le plus souvent, dans les algorithmes symétriques, leur longueur est fixée ; et dans les algorithmes à clé publique, il faut souvent utiliser des nombres avec des propriétés particulières (par exemple des nombres premiers). Il semble donc acceptable de limiter l'ensemble des clés possibles. Pour autant, ces clés ne sont pas forcément atomiques à notre sens, puisqu'elles peuvent être générées par morceaux (par exemple, l'un des agents génère un nonce k, tandis que l'autre agent génère un autre nonce k' afin d'établir une clé $\langle k, k'\rangle$ qui ne sera pas atomique). De telles situations sont exclues si nous interdisons les clés non-atomiques. Il s'agit donc d'une limite de cette approche.

Nous avons également limité l'ensemble des messages en exigeant qu'ils respectent les contours. Dans tous les exemples, seul le chiffrement asymétrique (et ses variantes, comme le chiffrement asymétrique randomisé) aura un contour non trivial, et la condition ne semble donc pas très restrictive : il s'agit seulement de supposer que $\mathsf{aenc}(a, b)$ n'a aucun sens, sauf si b est une clé publique $\mathsf{pub}(b')$.

1.2 Algèbre de processus

Maintenant que les messages ont été définis, nous pouvons nous intéresser à la modélisation des protocoles cryptographiques par une algèbre de processus. La section 1.2.1 présente la syntaxe de cette algèbre de processus. La sémantique associée est exposée en section 1.2.2. Elle stipule que les seules termes échangés sur le réseau doivent être des messages, au sens de la section 1.1.

1.2.1 Syntaxe

Nous disposons d'un ensemble infini $\mathcal{C}h$ de *canaux*. Les processus sont construits sur la grammaire suivante, avec $u \in \mathcal{T}_0(\Sigma_c, \Sigma_0^- \uplus \mathcal{N} \uplus \mathcal{X})$ et $c \in \mathcal{C}h$:

$$\begin{array}{rcll} P, Q & := & 0 & \text{processus nul} \\ & | & \mathsf{in}(c,u).P & \text{entrée} \\ & | & \mathsf{out}(c,u).P & \text{sortie} \\ & | & (P \mid Q) & \text{composition parallèle} \\ & | & i : P & \text{phase} \end{array}$$

Le processus nul (0) ne fait rien : il représente les processus qui ont fini de s'exécuter. Aussi, nous l'omettrons souvent. $\mathsf{in}(c,u).P$ représente un processus qui attend un Σ_0^+-message m correspondant au motif u sur le canal c. Une fois qu'il a reçu un tel message, il continuera à s'exécuter comme $P\sigma$ où $\sigma = mgu(m,u)$ (comme m est clos, σ est défini de manière unique). Le processus $\mathsf{out}(c,u).P$ émet u sur le canal c, à condition que u soit un message ; puis le processus P s'exécute. $(P|Q)$ représente les deux processus P et Q en parallèle, c'est-à-dire le processus qui exécute simultanément P et Q. Enfin, le processus $i : P$ permet d'exiger que l'on ait atteint la phase i avant de commencer l'exécution de P. Cette construction est particulièrement utile pour modéliser certaines propriétés. De cette manière, il est possible d'exprimer que l'attaquant ne doit pas pouvoir obtenir certaines informations, même s'il apprend certaines clés après l'exécution du protocole.

L'absence d'opération interne de déchiffrement (let), remplacée par le filtrage de motifs à la réception, en constitue la principale limite. Cependant, rien ne porte à croire que cette restriction ne puisse pas être levée.

Les processus seront supposés avoir des variables disjointes, c'est-à-dire que chaque variable n'est liée qu'une seule fois. En particulier, avec $x, y \in \mathcal{X}$ et $c \in \mathcal{C}h$, les processus $\mathsf{in}(c,x).\mathsf{in}(c,x)$ et $\mathsf{in}(c,x).\mathsf{in}(c,y)$ auront des sémantiques différentes : le premier attend deux fois le même message, alors que le second attend deux messages arbitraires. Une variable d'un processus P est dite *libre* si elle n'est dans le champ d'aucune entrée. L'ensemble des variables libres d'un processus P sera noté $fv(P)$, et l'ensemble des sous-termes de P, noté $St(P)$, est l'ensemble des sous-termes apparaissant dans P. Plus formellement, $St(0) = \emptyset$, $St(i : P) = St(P)$, $St(P|Q) = St(P) \cup St(Q)$ et $St(\mathsf{in}(c,u).P) = St(\mathsf{out}(c,u).P) = St(u) \cup St(P)$.

Nous supposerons aussi que les phases des processus sont *croissantes* , c'est-à-dire que, si $i : P$ est un processus, pour tout suffixe $j : Q$ de P, on a $j > i$.

Définition 1.1. *On dit que le processus P est un* protocole *si c'est un processus* clos*, c'est-à-dire $fv(P) = \emptyset$.*

Nous pouvons introduire l'exemple du protocole d'Otway-Rees qui nous servira de référence pour la suite.

Exemple 1.12. *Le protocole d'Otway-Rees [89] est un protocole d'établissement de clé (K_{ab}) entre deux agents grâce à un tiers de confiance joué par un serveur. Il se décrit informellement comme suit.*

1. $A \to B : M, A, B, \{N_a, M, A, B\}_{K_{as}}$
2. $B \to S : M, A, B, \{N_a, M, A, B\}_{K_{as}}, \{N_b, M, A, B\}_{K_{bs}}$
3. $S \to B : M, \{N_a, K_{ab}\}_{K_{as}}, \{N_b, K_{ab}\}_{K_{bs}}$
4. $B \to A : M, \{N_a, K_{ab}\}_{K_{as}}$

Les agents A et B cherchent à établir une clé, et S est le serveur. K_{as} (resp. K_{bs}) est la clé partagée par A (resp. B) avec S. M est un identifiant de session. N_a et N_b sont des nonces créés par A et B, et K_{ab} est une clé créée aléatoirement par S. Pour représenter le protocole d'Otway-Rees, nous utilisons Σ_5^{std}, comme définie à la fin de la section 1.1.7, et le processus P_{OR} avec

$$P_{OR} = P_A \mid P_B \mid P_S$$

où P_A, P_B et P_S représentent respectivement les rôles de A, B et S.

Soient $c_A, c_B, c_S \in \mathcal{C}h$ et les données suivantes :

$$x, y_a, y_b, y_m, z_m, z^1_{\mathsf{senc}}, z^2_{\mathsf{senc}}, z_{ab} \in \mathcal{X}$$
$$a, b \in \Sigma_0^-, \quad m, n_a, n_b, k_{as}, k_{bs}, k_{ab} \in \mathcal{N}$$

Les rôles sont définis comme suit.

$$\begin{aligned}
P_A &= \mathsf{out}(c_A, \langle m, a, b, \mathsf{senc}(\langle n_a, m, a, b\rangle^4, k_{as})\rangle^4).\\
&\quad\ \mathsf{in}(c_A, \langle m, \mathsf{senc}(\langle n_a, x\rangle^2, k_{as})\rangle^2)\\[1ex]
P_B &= \mathsf{in}(c_B, \langle z_m, a, b, z^1_{\mathsf{senc}}\rangle^4).\\
&\quad\ \mathsf{out}(c_B, \langle z_m, a, b, z^1_{\mathsf{senc}}, \mathsf{senc}(\langle n_b, z_m, a, b\rangle^4, k_{bs})\rangle^5).\\
&\quad\ \mathsf{in}(c_B, \langle z_m, z^2_{\mathsf{senc}}, \mathsf{senc}(\langle n_b, z_{ab}\rangle^2, k_{bs})\rangle^3).\\
&\quad\ \mathsf{out}(c_B, \langle z_m, z^2_{\mathsf{senc}}\rangle^2)\\[1ex]
P_S &= \mathsf{in}(c_S, \langle y_m, a, b, \mathsf{senc}(\langle y_a, y_m, a, b\rangle^4, k_{as}), \mathsf{senc}(\langle y_b, y_m, a, b\rangle^4, k_{bs})\rangle^5).\\
&\quad\ \mathsf{out}(c_S, \langle y_m, \mathsf{senc}(\langle y_a, k_{ab}\rangle^2, k_{as}), \mathsf{senc}(\langle y_b, k_{ab}\rangle^2, k_{bs})\rangle^3))
\end{aligned}$$

1.2.2 Sémantique

Avant d'expliciter la sémantique des processus, nous devons définir le contexte dans lequel ils s'exécutent, et qui tient compte des processus qui restent à exécuter, de la connaissance de l'attaquant, de l'instanciation des variables du protocole et de la phase courante. Soit Σ_0 un ensemble de constantes. On dit que le quadruplet $(\mathcal{P}; \phi; \sigma; i)$ est une Σ_0-*configuration* si :

— $\mathcal{P}$ est un multi-ensemble de processus (qui peuvent avoir des variables libres)

— ϕ est une substitution avec $dom(\phi) \subset \mathcal{W}$ et $img(\phi) \subset \mathcal{M}_{\Sigma_0}$, appelée Σ_0-*trame*.

— σ est une substitution avec $fv(\mathcal{P}) \subset dom(\sigma) \subset \mathcal{X}$.

— $i \in \mathbb{N}$.

Une Σ_0-configuration est *initiale* si $\sigma = \emptyset$. On notera souvent P à la place de $0{:}P$ ou de $(0{:}P; \emptyset; \emptyset; 0)$ et $P \uplus \mathcal{P}$ pour $\{P\} \uplus \mathcal{P}$. Le multi-ensemble $\mathcal{P}$ représente ce qui reste à exécuter des processus. ϕ modélise les messages qui ont été diffusés sur le réseau et qui sont connus de l'attaquant, et σ permet de conserver l'affectation des variables libres. L'entier i désigne la phase courante.

La sémantique des processus relativement à Σ_0 est décrite par la relation $\xrightarrow{\alpha}$ définie à la figure 1.1, avec la convention que lorsque P n'est de la forme $i{:}P$ pour aucun entier i, il s'exécute comme $0{:}P$. La règle de réception décrit la réception par un processus d'un message construit

RÉCEPTION $(i{:}\mathsf{in}(c,u).P \cup \mathcal{P}; \phi; \sigma; i) \xrightarrow{\mathsf{in}(c,R)} (i{:}P \cup \mathcal{P}; \phi; \sigma \uplus \sigma_0; i)$ où R est une Σ_0-recette tel que $R\phi\downarrow$ est un Σ_0-message, et $R\phi\downarrow = (u\sigma)\sigma_0$ pour σ_0 avec $dom(\sigma_0) = vars(u\sigma)$.

ÉMISSION $(i{:}\mathsf{out}(c,u).P \cup \mathcal{P}; \phi; \sigma; i) \xrightarrow{\mathsf{out}(c,\mathsf{w})} (i{:}P \cup \mathcal{P}; \phi \cup \{\mathsf{w} \triangleright u\sigma\}; \sigma; i)$ avec w une variable fraîche de $\mathcal{W}$, et $u\sigma$ un Σ_0-message.

CHANGEMENT DE PHASE	$(\mathcal{P}; \phi; \sigma; i)$	$\xrightarrow{\mathsf{phase}\ i'}$	$(\mathcal{P}; \phi; \sigma; i')$	avec $i' > i$.
PHASE	$(i{:}i'{:}P \cup \mathcal{P}; \phi; \sigma; i)$	$\xrightarrow{\tau}$	$(i'{:}P \cup \mathcal{P}; \phi; \sigma; i)$	
PARALLÉLISME	$(i{:}(P \mid Q) \cup \mathcal{P}; \phi; \sigma; i)$	$\xrightarrow{\tau}$	$(i{:}P \cup i{:}Q \cup \mathcal{P}; \phi; \sigma; i)$	

FIGURE 1.1 – Sémantique des processus relative à Σ_0

par l'attaquant. La règle d'émission représente la diffusion d'un message par un processus. Ce message est ajouté à la connaissance de l'attaquant, représentée par la trame ϕ. Le parallélisme rend chaque processus disponible dans le multi-ensemble de processus $\mathcal{P}$; cette action n'est pas observable. Grâce à la règle de changement de phase, l'attaquant peut augmenter la phase à tout moment, tandis que la règle de phase permet à un processus de passer à l'étape suivante.

La relation $\xrightarrow{\alpha_1 \dots \alpha_n}$ (relativement à Σ_0) est la clôture réflexive transitive de $\xrightarrow{\alpha}$ (relativement à Σ_0). Étant donnée une séquence d'actions $\alpha_1 \dots \alpha_n$, la *trace* tr correspondante est la séquence $\alpha_1 \dots \alpha_n$ sans ses τ-actions. Si $\mathcal{K} \xrightarrow{\alpha_1 \dots \alpha_n} \mathcal{K}'$ et tr est la trace correspondante à $\alpha_1 \dots \alpha_n$, on note $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}'$. L'ensemble des traces d'une configurations est l'ensemble des séquences d'actions que l'attaquant peut observer ainsi que la connaissance qu'il obtient en les observant.

Les règles sont classiques, à l'exception de celles de la phase. Par exemple, dans ProVerif [37], la phase n'est pas visible, mais une fausse attaque est levée chaque fois que les processus ne se trouvent pas dans la même phase. Pour simplifier, dans notre sémantique, les phases et les changements de phases sont visibles. Ces phases servent surtout à modéliser les propriétés.

Définition 1.2. *Soit Σ_0 un ensemble de constantes, $\mathcal{K}$ une Σ_0-configuration. On note $\mathsf{trace}_{\Sigma_0}(\mathcal{K})$ l'ensemble défini par*

$$\mathsf{trace}_{\Sigma_0}(\mathcal{K}) = \{(\mathsf{tr}, \phi) \mid \exists \mathcal{P}, \sigma, i \text{ tel que } \mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i) \text{ avec } (\mathcal{P}; \phi; \sigma; i) \text{ une } \Sigma_0\text{-configuration.}\}$$

On parlera parfois de *trace concrète* d'une Σ_0-configuration $\mathcal{K}$ pour désigner la trace $\mathsf{tr}\phi\downarrow$ où $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K})$. L'exemple suivant analyse une trace du protocole d'Otway-Rees.

Exemple 1.13. *Reprenons P_{OR} comme défini dans l'exemple 1.12. Considérons la trace tr suivante.*

$$\mathsf{tr} = \mathsf{out}(c_A, \mathsf{w}_1).\mathsf{in}(c_B, \mathsf{w}_1).\mathsf{out}(c_B, \mathsf{w}_2).\mathsf{in}(c_S, \mathsf{w}_2).\mathsf{out}(c_S, \mathsf{w}_3).\mathsf{in}(c_A, R)$$

où $R = \langle \mathsf{proj}_1^4(\mathsf{w}_1), \mathsf{proj}_2^3(\mathsf{w}_3)\rangle^2$. Avec cette séquence d'actions, l'attaquant obtient la connaissance suivante, avec $t_A = \mathsf{senc}(\langle n_a, m, a, b\rangle^4, k_{as})$, $t_B = \mathsf{senc}(\langle n_b, m, a, b\rangle^4, k_{bs})$:

$$\phi = \{\mathsf{w}_1 \triangleright \langle m, a, b, t_A\rangle^4,\ \mathsf{w}_2 \triangleright \langle m, a, b, t_A, t_B\rangle^5,\ \mathsf{w}_3 \triangleright \langle m,\ \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}),\ \mathsf{senc}(\langle n_b, k_{ab}\rangle^2, k_{bs})\rangle^3\}$$

Nous avons $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}((P_{OR}; \emptyset; \emptyset; 0))$. A l'exception de la dernière action $\mathsf{in}(c_A, R)$, la trace tr correspond à une exécution normale du protocole. Pour la dernière action, l'attaquant remplace P_B : il récupère directement l'identifiant de session m et le chiffrement $\mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as})$ contenant la clé k_{ab} et les passe au processus P_A. $R\phi\downarrow = \langle m, \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as})\rangle^2$ et la trace

concrète correspondante à (tr, ϕ) *est :*

$$\begin{aligned}\mathsf{tr}\phi\downarrow =& \mathsf{out}(c_A, \langle m, a, b, t_A\rangle^4).\mathsf{in}(c_B, \langle m, a, b, t_A\rangle^4).\mathsf{out}(c_B, \langle m, a, b, t_A, t_B\rangle^5).\\ & \mathsf{in}(c_S, \langle m, a, b, t_A, t_B\rangle^5).\mathsf{out}(c_S, \langle m, \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}), \mathsf{senc}(\langle n_b, k_{ab}\rangle^2, k_{bs})\rangle^3).\\ & \mathsf{in}(c_A, \langle m, \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as})\rangle^2)\end{aligned}$$

La substitution σ *telle que* $(P_{OR}; \emptyset; \emptyset; 0)) \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; 0)$ *est*

$$\sigma = \{z_m \triangleright m \ ; \ z^1_{\mathsf{senc}} \triangleright t_A \ ; \ y_m \triangleright m \ ; \ y_a \triangleright n_a \ ; \ y_b \triangleright n_b \ ; \ x \triangleright k_{ab}\}$$

Quant au multi-ensemble $\mathcal{P}$ *des processus qui restent à exécuter, il vaut :*

$$\mathcal{P} = \{0 \ ; \ 0 \ ; \ \mathsf{in}(c_B, \langle z_m, z^2_{\mathsf{senc}}, \mathsf{senc}(\langle n_b, z_{ab}\rangle^2, k_{bs})\rangle^3).\mathsf{out}(c_B, \langle z_m, z^2_{\mathsf{senc}}\rangle^2)\}$$

1.3 Équivalences

De nombreuses propriétés de sécurité, comme le secret fort, l'intraçabilité, l'anonymat ou la confidentialité d'un protocole de vote électronique, s'expriment en affirmant que le protocole réel est indistinguable d'un modèle idéal ou que le protocole reste équivalent si on modifie les données sensibles. Par exemple, dans le modèle d'Arapinis, Chothia, Ritter et Ryan [14], dire qu'un protocole est intraçable consiste à affirmer que du point de vue de l'attaquant, il est indistingable de jouer plusieurs fois le protocole avec la même clé ou d'utiliser chaque fois des clés différentes. D'autre part, un protocole est anonyme si l'attaquant ne peut pas faire la différence entre un protocole joué par Alice et le même protocole joué par Bob. Ces propriétés s'expriment à travers la notion d'équivalence de protocoles. L'équivalence entre deux trames, c'est-à-dire l'équivalence en face d'un attaquant passif, sera définie dans la prochaine section, et la suivante présentera l'équivalence de protocoles, c'est-à-dire contre un attaquant qui contrôle le réseau.

1.3.1 Équivalence statique

Supposons que l'attaquant a enregistré les communications émises au cours de l'exécution d'un protocole, dont nous voulons montrer l'anonymat. Il s'agit donc de démontrer que l'attaquant n'est pas capable de savoir si c'est Alice ou Bob qui a participé à l'exécution de ce protocole. Or, la seule façon pour l'attaquant de distinguer entre ces deux scénarii est d'effectuer des tests sur les messages qu'il a reçus. Si les tests donnent des résultats indépendants de l'agent (Alice ou Bob) qui s'en sert, l'attaquant ne peut extraire aucune information de ces messages. Cette situation est formalisée par la notion d'équivalence statique.

Définition 1.3. *Soit* Σ_0 *un ensemble de constantes. Soient* ϕ *et* ψ *deux* Σ_0*-trames avec* $dom(\phi) = dom(\psi)$*. On dit que* ϕ *est* statiquement incluse *dans* ϕ *par rapport à* Σ_0*, noté* $\phi \sqsubseteq_s \psi$*, si :*

- *Pour toute* Σ_0*-recette* R *telle que* $R\phi\downarrow$ *est un* Σ_0*-message,* $R\psi\downarrow$ *est un* Σ_0^+*-message.*
- *Pour toutes* Σ_0*-recettes* R, R' *telles que* $R\phi\downarrow = R'\phi\downarrow$ *est un* Σ_0*-message,* $R\psi\downarrow = R'\psi\downarrow$*.*

Lorsque l'équivalence statique est réciproque, c'est-à-dire lorsque $\phi \sqsubseteq_s \psi$ *et* $\psi \sqsubseteq_s \phi$ *par rapport à* Σ_0*, on dit que* ϕ *et* ψ *sont* statiquement équivalentes *par rapport à* Σ_0*, et on note* $\phi \sim_s \psi$*.*

Certains tests sont vrais dans toutes les trames de même domaine, par exemple $\mathsf{w}_1 = \mathsf{w}_1$ est vrai dans ϕ dès que $\mathsf{w}_1 \in dom(\phi)$. On dira parfois que ces tests sont *triviaux*.

L'exemple suivant illustre la notion d'équivalence statique.

Exemple 1.14. *Considérons à nouveau la trame ϕ de l'exemple 1.13, et posons $\phi_1 = \phi \uplus \{\mathsf{w}_4 \triangleright k_{as}\}$ et $\phi_2 = \phi \uplus \{\mathsf{w}_4 \triangleright k\}$ avec $k \in \mathcal{N}$. Les domaines de ϕ_1 et ϕ_2 sont identiques, mais pour $R = \mathsf{sdec}(\mathsf{proj}_4^4(\mathsf{w}_1), \mathsf{w}_4)$, $R\phi_1\downarrow \in \mathcal{M}_{\Sigma_0^-}$ tandis que $R\phi_2\downarrow$ n'est pas un message. Donc $\phi_1 \not\sqsubseteq_s \phi_2$ et $\phi_1 \not\sim_s \phi_2$. Au contraire, comme k est un nom frais, $\phi_2 \sqsubseteq_s \phi_1$.*

L'inclusion statique seule ne garantit aucune sécurité : si une trame ϕ est incluse dans une autre trame ψ, sans que ψ soit incluse dans ϕ, l'attaquant peut faire la différence entre les deux simplement parce qu'il existe un test vrai dans ψ et faux dans ϕ. Cependant, cette notion est utile en pratique et permet de définir une approximation utile de l'équivalence de trace.

1.3.2 Équivalences de trace

Maintenant que l'équivalence contre un attaquant passif a été définie, il reste à traiter le cas de l'attaquant actif. Un tel attaquant peut interagir avec le protocole, mais, tant qu'il ne sait pas s'il a affaire à Alice ou à Bob, il ne peut pas utiliser cette information : il doit donc effectuer les mêmes opérations sur P et sur Q. Il dispose alors de deux manières de distinguer entre les deux scenarii. Soit les messages échangés lui permettent de savoir s'il interagit avec Alice ou avec Bob, et il s'agit à nouveau d'un problème d'équivalence statique ; soit l'un des protocoles réagit différemment de l'autre. La définition ci-dessous formalise cette description.

Définition 1.4. *Soit Σ_0 un ensemble de constantes. Soient $\mathcal{K}$ et $\mathcal{K}'$ deux Σ_0-configurations. On dit que $\mathcal{K}$ est en* inclusion de trace *dans $\mathcal{K}'$ par rapport à Σ_0, noté $\mathcal{K} \sqsubseteq_t \mathcal{K}'$, si pour tout $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K})$, il existe ψ tel que $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K}')$ tel que $\phi \sim_s \psi$ par rapport à Σ_0. Lorsque $\mathcal{K} \sqsubseteq_t \mathcal{K}'$ et $\mathcal{K}' \sqsubseteq_t \mathcal{K}$, on dira que $\mathcal{K}$ et $\mathcal{K}'$ sont en* équivalence de trace *par rapport à Σ_0, et on notera $\mathcal{K} \approx_t \mathcal{K}'$.*

Cette définition est la définition classique de l'équivalence de trace, mais les développements techniques nécessiteront une notion plus faible qui a été introduite par Chadha, Ciobâcă et Kremer (voir [44]).

Définition 1.5. *Soit Σ_0 un ensemble de constantes. Soient $\mathcal{K}$ et $\mathcal{K}'$ deux Σ_0-configurations. On dit que $\mathcal{K}$ est en* inclusion approximative de trace *dans $\mathcal{K}'$ par rapport à Σ_0 (ou plus simplement que $\mathcal{K}$ est Σ_0-incluse dans $\mathcal{K}'$), noté $\mathcal{K} \sqsubseteq_{at} \mathcal{K}'$, si pour tout $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K})$, il existe ψ tel que $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K}')$ tel que $\phi \sqsubseteq_s \psi$ par rapport à Σ_0. Lorsque $\mathcal{K} \sqsubseteq_{at} \mathcal{K}'$ et $\mathcal{K}' \sqsubseteq_{at} \mathcal{K}$ par rapport à Σ_0, on dira que $\mathcal{K}$ et $\mathcal{K}'$ sont Σ_0-équivalentes.*

Une trace tr telle qu'il existe une Σ_0-trame ϕ avec $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K})$ mais aucune trame ψ avec $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K}')$ et $\phi \sqsubseteq_s \psi$ est appelée *témoin* ou *trace de non-inclusion* $\mathcal{K} \sqsubseteq_{at} \mathcal{K}'$. L'exemple ci-dessous, adapté de [44], illustre la différence entre l'équivalence de trace et son approximation.

Exemple 1.15. *Soient $n, m, k \in \mathcal{N}$, $c, c' \in \mathcal{C}h$ et $x \in \mathcal{X}$. Soient P, Q les processus*

$$P = \mathsf{out}(c, \mathsf{senc}(n, k)) \mid \mathsf{out}(c, \mathsf{senc}(m, k)) \mid \mathsf{in}(c', \mathsf{senc}(x, k)).\mathsf{out}(c', n).\mathsf{out}(c', n)$$
$$Q = \mathsf{out}(c, \mathsf{senc}(n, k)) \mid \mathsf{out}(c, \mathsf{senc}(m, k)) \mid \mathsf{in}(c', \mathsf{senc}(x, k)).\mathsf{out}(c', n).\mathsf{out}(c', x)$$

Considérons la trace $\mathsf{tr} = \mathsf{out}(c, \mathsf{w}_1).\mathsf{in}(c', \mathsf{w}_1).\mathsf{out}(c', \mathsf{w}_2).\mathsf{out}(c', \mathsf{w}_3)$. Comme l'adversaire ne peut passer $\mathsf{in}(c, \mathsf{senc}(x, k))$ qu'après l'exécution de l'une des instructions sur le canal c, et comme il ne peut pas choisir lequel des messages $\{n\}_k$ et $\{m\}_k$ correspond à w_1, il suffit d'examiner cette trace pour savoir si les configurations sont en équivalence ou non.

Les trames ϕ telles que $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(P)$ sont $\phi_1 = \{\mathsf{w}_1 \triangleright \mathsf{senc}(n, k)\ ;\ \mathsf{w}_2 \triangleright n\ ;\ \mathsf{w}_3 \triangleright n\}$ et $\phi_2 = \{\mathsf{w}_1 \triangleright \mathsf{senc}(m, k)\ ;\ \mathsf{w}_2 \triangleright n\ ;\ \mathsf{w}_3 \triangleright n\}$. Les trames ψ telles que $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^-}(Q)$ sont $\psi_1 = \phi_1$ et $\psi_2 = \{\mathsf{w}_1 \triangleright \mathsf{senc}(m, k)\ ;\ \mathsf{w}_2 \triangleright n\ ;\ \mathsf{w}_3 \triangleright m\}$. Dans ψ_2, les seuls tests qui tiennent sont les

tests triviaux car l'attaquant ne connaît pas k. Donc $Q \not\sqsubseteq_t P$ *puisque* $(\mathsf{tr}, \psi_2) \in \mathsf{trace}_{\Sigma_0^-}(Q)$ *mais* $\phi_1 \not\sqsubseteq_s \psi_2$ *et* $\phi_2 \not\sqsubseteq_s \psi_2$ *(le test* $\mathsf{w}_2 = \mathsf{w}_3$ *est vrai dans* ϕ_1 *et* ϕ_2 *mais pas dans* ψ_2*). On en déduit que* $P \not\approx_t Q$*.*

Cependant, $P \sqsubseteq_t Q$ *car* $\phi_1 \sqsubseteq_s \psi_1$ *et* $\phi_2 \sqsubseteq_s \psi_1$*. On en déduit que* $P \sqsubseteq_{at} Q$*. De plus,* $\psi_1 = \phi_1$ *et* $\psi_1 \sqsubseteq_s \phi_2$ *car le test* $\mathsf{w}_2 = \mathsf{w}_3$ *est vrai dans* ψ_1 *et* ϕ_2 *et tous les autres tests qui tiennent dans* ψ_1 *se ramènent à ce test ou sont triviaux. Enfin,* $\psi_2 \sqsubseteq_s \phi_1$ *et* $\psi_2 \sqsubseteq_s \phi_2$ *car les seuls tests vrais dans* ψ_2 *sont les tests triviaux (autrement dit,* ψ_2 *est statiquement incluse dans toute trame ayant le même domaine). Donc* $Q \sqsubseteq_{at} P$*. Nous avons donc* $P \not\approx_t Q$ *mais* $P \approx_{at} Q$*.*

Dans la section suivante, nous démontrerons que sous une hypothèse de déterminisme, l'équivalence de trace et l'équivalence approximative de trace coïncident ; nous en profiterons pour développer l'exemple plus réaliste du protocole d'Otway-Rees. Dans toute la suite, nous nous concentrerons sur l'étude de protocoles déterministes, et nous nous intéresserons exclusivement à la démonstration de l'inclusion approximative de trace.

1.4 Déterminismes

Diverses notions de déterminisme ont été introduites. Il suffit que les protocoles vérifient la moins restrictive, introduite par Chadha, Ciobâcă et Kremer [44], pour que les deux équivalences de trace présentées dans la section précédente soient identiques. Selon cette définition, un protocole peut avancer de plusieurs manières différentes, mais les configurations résultantes doivent rester en équivalence statique. Nous l'introduisons pour démontrer le lemme 1.1, mais c'est principalement la définition 1.7, plus forte, qui sera utilisée dans la suite.

Définition 1.6. *Soit* Σ_0 *un ensemble de constantes. Soit* $\mathcal{K}$ *une* Σ_0*-configuration. On dit que* $\mathcal{K}$ *est* déterminée *si, pour toute paire d'exécutions* $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}' = (P', \phi', \sigma', i')$ *et* $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}'' = (P'', \phi'', \sigma'', i'')$ *avec* $\mathcal{K}'$ *et* $\mathcal{K}''$ *deux* Σ_0*-configurations et* tr *une trace, on a* $\phi' \sim_s \phi''$*.*

Autrement dit, la trame est déterminée par la trace, à équivalence statique près. Sous cette hypothèse, les deux équivalences coïncident, comme le démontre le lemme suivant.

Lemme 1.1. *Soit* Σ_0 *un ensemble de constantes. Soient* $\mathcal{K}$ *et* $\mathcal{K}'$ *deux* Σ_0*-configurations déterminées. On a* $\mathcal{K} \approx_{at} \mathcal{K}'$ *par rapport à* Σ_0 *si et seulement si* $\mathcal{K} \approx_t \mathcal{K}'$ *par rapport à* Σ_0*.*

Démonstration. Comme $\mathcal{K} \sqsubseteq_t \mathcal{K}'$ implique $\mathcal{K} \sqsubseteq_{at} \mathcal{K}'$, et par symétrie, il suffit de démontrer que $\mathcal{K} \approx_{at} \mathcal{K}'$ implique $\mathcal{K} \sqsubseteq_t \mathcal{K}'$. Supposons donc que $\mathcal{K} \approx_{at} \mathcal{K}'$ par rapport à Σ_0.

Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K})$. Comme $\mathcal{K} \sqsubseteq_{at} \mathcal{K}'$, il existe ψ tel que $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0}(\mathcal{K}')$ et $\phi \sqsubseteq_s \psi$. Comme $\mathcal{K}' \sqsubseteq_{at} \mathcal{K}$, il existe ϕ' tel que $(\mathsf{tr}, \phi') \in \mathsf{trace}_{\Sigma_0}(\mathcal{K})$ et $\psi \sqsubseteq_s \phi'$. Comme $\mathcal{K}$ est déterminée, $\phi \sim_s \phi'$.

Tout test vrai dans ϕ est vrai dans ψ par $\phi \sqsubseteq_s \psi$. Réciproquement, tout test vrai dans ψ est vrai dans ϕ' par $\psi \sqsubseteq_s \phi'$ et donc dans ϕ par $\phi' \sqsubseteq_s \phi$. On a donc $\phi \sim_s \psi$, d'où $\mathcal{K} \sqsubseteq_t \mathcal{K}'$. □

Dans de nombreux cas, la notion de configuration déterminée est trop faible, et il faut que l'attaquant sâche exactement où il en est de l'exécution du protocole. Cette forme de déterminisme correspond à la définition introduite par Baelde, Delaune et Hirschi [27].

Définition 1.7. *Soit* Σ_0 *un ensemble de constantes. Soit* $\mathcal{K}$ *une* Σ_0*-configuration.* $\mathcal{K}$ *est* déterministe par action *(ou simplement* déterministe*) si chaque fois que* $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ *est une exécution et* $i{:}\alpha.P, j{:}\beta.P \in \mathcal{P}$ *avec* α, β *des instructions de la forme* $\mathsf{in}(c, u)$*,* $\mathsf{out}(c', u)$*, alors au moins l'une des conditions suivantes est vérifiée :*

— $i \neq j$

— *Les canaux de α et β sont différents*

— *α et β sont des instructions de nature différente (si α est de la forme $\mathsf{in}(c,u)$, alors β est de la forme $\mathsf{out}(c,u)$ ou inversement).*

Une conséquence importante de cette forme de déterminisme est que la trame est entièrement déterminée par la trace de l'exécution. Il en découle en particulier que toute configuration déterministe est déterminée.

Lemme 1.2. *Soit Σ_0 un ensemble de constantes. Soit $\mathcal{K}$ une Σ_0-configuration déterministe par action. Soient tr une trace, $\mathcal{K}_1 = (\mathcal{P}_1;\phi_1;\sigma_1;i_1)$, et $\mathcal{K}_2 = (\mathcal{P}_2;\phi_2;\sigma_2;i_2)$, telles que $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}_1$ et $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}_2$. Alors $\phi_1 = \phi_2$, $\sigma_1 = \sigma_2$ et $i_1 = i_2$.*

Démonstration. Le résultat se démontre par récurrence. La seule difficulté notable consiste à remarquer que l'on n'a pas nécessairement $\mathcal{P}_1 = \mathcal{P}_2$, car il peut y avoir une différence entre les τ-actions qui ont été exécutées d'un côté et de l'autre. Toutefois, si l'on exécute toutes les τ-actions, qui sont toujours en nombre fini, $\mathcal{P}_1$ et $\mathcal{P}_2$ sont identiques. Ensuite, il y a trois cas :

1. $\mathsf{tr} = \mathsf{tr}_0.\mathsf{in}(c,R)$. Par hypothèse de récurrence, $\phi_1 = \phi_2$ après tr_0 donc $R\phi_1{\downarrow} = R\phi_2{\downarrow}$. Comme il n'y a qu'un seul $\mathsf{in}(c,u).P$ accessible dans $\mathcal{P}_1$ et $\mathcal{P}_2$, on en déduit que $\sigma_1 = \sigma_2$ après tr.
2. $\mathsf{tr} = \mathsf{tr}_0.\mathsf{out}(c,\mathsf{w})$. Par hypothèse de récurrence, $\sigma_1 = \sigma_2$ après tr_0. Il n'y a qu'un seul $\mathsf{out}(c,u).P$ accessible dans $\mathcal{P}_1$ et $\mathcal{P}_2$, donc les messages émis sont $u\sigma_1$ et $u\sigma_2$. On a donc $\phi_1 \uplus \{\mathsf{w} \triangleright u\sigma_1\} = \phi_2 \uplus \{\mathsf{w} \triangleright u\sigma_2\}$ car $\sigma_1 = \sigma_2$ par hypothèse de récurrence.
3. $\mathsf{tr} = \mathsf{tr}_0.\mathsf{phase}\ i$. La phase évolue de la même manière des deux côtés, donc $i_1 = i_2 = i$.

□

Il est possible de garantir que les processus sont déterministes à l'aide d'une condition syntaxique. On dit qu'un processus P est *simple* s'il existe un entier k tel que

$$P = (P_1 \mid P_2 \cdots \mid P_k)$$

et de plus, pour chaque $1 \leqslant i \leqslant k$, le processus P_i est *séquentiel* utilisant un seul canal c_i, c'est-à-dire qu'il est construit sur la grammaire

$$P := 0 \mid \mathsf{in}(c_i,u).P \mid \mathsf{out}(c_i,u).P \mid i : P$$

et de plus $c_i \neq c_j$ si $i \neq j$. Il est clair qu'un processus simple est déterministe par action. De plus, la plupart des protocoles déterministes peuvent se voir comme des protocoles simples où chaque canal représente les actions d'un agent donné. Cette hypothèse est souvent réalisée en pratique : par exemple, les adresses IP des participants peuvent servir de canaux qui les identifient.

L'exemple suivant illustre ces notions à travers le protocole d'Otway-Rees.

Exemple 1.16. *Revenons au processus P_{OR} de l'exemple 1.12 et codons une propriété particulièrement forte : à la fin de l'exécution, l'attaquant ne doit pas être capable de distinguer entre la clé k_{as} de A et une clé fraîche k. Dans ce but, on définit $P'_A = P_A.1{:}\mathsf{out}(c_A,k_{as})$ et $P''_A = P_A.1{:}\mathsf{out}(c_A,k)$ avec $k \in \mathcal{N}$, et on obtient $P'_{OR} = P'_A \mid P_B \mid P_S$ et $P''_{OR} = P''_A \mid P_B \mid P_S$. Les processus P_{OR}, P'_{OR} et P''_{OR} sont simples et donc déterministes et déterminés. On considère la trace tr de l'exemple 1.13, et on lui ajoute $\mathsf{phase}\ 1.\mathsf{out}(c,\mathsf{w}_4)$ pour obtenir la trace $\mathsf{tr}' = \mathsf{tr}.\mathsf{phase}\ 1.\mathsf{out}(c,\mathsf{w}_4)$. On obtient $(\mathsf{tr}',\phi_1) \in \mathsf{trace}_{\Sigma_0^-}(P'_{OR})$ et $(\mathsf{tr}',\phi_2) \in \mathsf{trace}_{\Sigma_0^-}(P''_{OR})$ où ϕ_1 et ϕ_2 ont été définies à l'exemple 1.14. Nous avions montré que $\phi_1 \not\sim_s \phi_2$, et donc, comme ces trames sont les seules qui puissent être obtenues après tr (d'après le lemme 1.2), on a $P'_{OR} \not\approx_t P_{OR}$. Dans l'exemple 1.14, nous avons démontré que $\phi_1 \not\sqsubseteq_s \phi_2$, et donc $P'_{OR} \not\approx_{at} P_{OR}$, ce qui est cohérent avec le lemme 1.1.*

Dans ce cas particulier, l'intérêt de la phase n'est pas clair, car la propriété est tout aussi fausse si l'on supprime la restriction de phase. Toutefois, l'idée est simplement de savoir si l'agent est capable de distinguer la clé après l'exécution, mais sans pouvoir l'utiliser pour interagir avec le protocole.

L'hypothèse de déterminisme amène quelques restrictions. En effet, certains protocoles, comme par exemple le système Norvégien [66] de vote électronique, utilisent le non-déterminisme, et leur version déterminisée n'est pas sécurisée. En effet, le système utilise un réseau de mélangeurs mixnet modélisé par le non-déterminisme. Une autre limitation provient de ce que la propriété d'intraçabilité formalisée par Arapinis *et al.* [14] repose sur l'utilisation du non-déterminisme : la version déterministe de la propriété $!\,\mathsf{new}\ n\,!\,\mathsf{new}\ k.\,P(n,k) \approx_t !\,\mathsf{new}\ n.\mathsf{new}\ k.\,P(n,k)$ est

$$!\,\mathsf{new}\ c'.\mathsf{out}(c_0,c').\mathsf{new}\ n\,!\,\mathsf{new}\ c''.\mathsf{out}(c',c'').\mathsf{new}\ k.\,P(n,k) \approx_t !\,\mathsf{new}\ c'.\mathsf{out}(c_0,c').\mathsf{new}\ n.\,\mathsf{new}\ k.\,P(n,k)$$

et elle n'est jamais vérifiée simplement parce que le nombre de réplications est différent de chaque côté, et que les réplications doivent être visibles pour conserver le déterminisme. Toutefois, cette propriété utilise la réplication, tandis que nous travaillons dans un modèle qui n'en contient pas, et il ne s'agit donc pas d'une restriction supplémentaire.

1.5 Conclusion

Ce chapitre a présenté le modèle qui sera utilisé dans la suite. Les messages qui transitent sur le réseau sont modélisés par une algèbre de termes à laquelle des restrictions ont été apportées à travers les sortes et les contours. Pour le moment, ces restrictions ne semblent pas importantes, puisque les sortes et les contours que nous pouvons affecter à chaque symbole constructeur ne sont pas contraints. De nouvelles limites seront données dans le chapitre suivant, sous forme d'hypothèses sur les règles de réécriture.

Les relations entre les primitives sont représentées par des théories de réécriture particulières, appelées théories à constructeurs, qui supposent que l'échec d'un déchiffrement est visible. Ainsi, un processus n'acceptera pas un message déchiffré avec une mauvaise clé. Ce modèle empêche la modélisation de certaines primitives, comme le XOR, pour lesquelles la notion de déchiffrement (et donc d'échec de ce déchiffrement) n'a pas de sens.

Ces messages sont manipulés par des programmes distribués, représentés formellement par une algèbre de processus. Cette thèse étudie en particulier l'équivalence entre de tels processus, c'est-à-dire leur indistinguabilité du point de vue de l'attaquant qui contrôle le réseau. Nous avons donné la définition de référence de cette équivalence, appelée équivalence de trace, ainsi qu'une approximation dont nous avons démontré qu'elle est exacte sous une hypothèse de déterminisme.

2 Résultats de typage

Afin de coder le problème de l'équivalence de protocoles sous forme de problème de planification et de formule SAT, il est nécessaire de borner la taille des messages utilisés par l'attaquant. Une solution consiste à typer les messages, et à se contenter de considérer les messages suivant le système de types, comme dans SATMC [21]. Cependant, rien ne garantit que l'attaquant respecte le système de type. Il semble donc naturel de chercher à déterminer une classe de protocoles et de systèmes de types pour laquelle ce n'est pas le cas, c'est-à-dire pour laquelle il existe une attaque bien typée chaque fois qu'il existe une attaque arbitraire.

Heather, Lowe et Schneider [76] ont démontré un résultat de ce genre pour le secret et l'authentification, mais avec des hypothèses très fortes. En particulier, chaque type doit être indiqué par une étiquette contenue dans le message, et les processus ne doivent pas transmettre de messages qu'ils ne peuvent pas ouvrir. Ramanujam et Suresh établissent [91] établissent un résultat semblable, avec un système de marquage encore plus contraignant. Un résultat plus fin a été établi par Almousa, Mödersheim, Modesti et Viganò [11] et formalisé en Isabelle/HOL par Hess et Mödersheim [78]. Ce résultat, semblable à celui qui est exposé ici, porte exclusivement sur l'accessibilité, avec une classe paramétrique de primitives incomparable à la nôtre.

Nous étendons le résultat de Chrétien, Cortier et Delaune [51], qui porte exclusivement sur le chiffrement symétrique pour les propriétés d'équivalence. La technique de preuve de [51] (et donc la nôtre) s'inspire des travaux d'Arapinis et Duflot [15], où il est montré que, dans le cas de l'accessibilité et pour une classe de protocoles marqués, il suffit de considérer des attaques bien typées pour un système de types particulier. Le présent chapitre démontre que l'on peut se contenter de considérer des attaques bien typées pour une classe paramétrique de primitives cryptographiques, et pour les propriétés d'accessibilité ou d'équivalence. Ces extensions permettent en particulier de considérer les primitives classiques, comme les chiffrements symétrique ou asymétrique, la signature électronique et les fonctions de hachage, qu'elles soient randomisées ou non. De plus, la preuve a été considérablement simplifiée. Comme dans [51], l'hypothèse qui porte sur les marques est relâchée sous une forme plus faible, à savoir que deux sous-termes chiffrés unifiables ont le même type, et qui peut être obtenue en ajoutant des marques.

Le modèle général a été présenté dans le chapitre précédent, mais nous devons ajouter des hypothèses supplémentaires dans la section 2.1, qui apporteront des contraintes sur les contours et les sortes, afin de limiter la non-linéarité des règles. Puis la section 2.2 présentera les systèmes de types et la conformité d'un protocole à un tel système de type, ce qui permettra d'énoncer les deux théorèmes du chapitre : pour l'accessibilité et pour l'équivalence, il existe une trace d'attaque si et seulement si il existe une trace d'attaque bien typée. Les résultats techniques introduits dans la section 2.3 nous aideront à démontrer le résultat de typage pour l'accessibilité dans la section 2.4 et le théorème qui porte sur l'équivalence dans la section 2.5. Enfin, la section 2.6 reviendra sur les hypothèses pour les justifier à travers des exemples qui mettent en défaut les théorèmes et la section 2.7 comparera ce résultat de typage aux autres résultats existants.

2.1 Théories

Le chapitre précédent exposait le modèle en toute généralité. Cependant, les résultats du présent chapitre nécessitent des hypothèses plus fortes sur les primitives cryptographiques, qui s'expriment à l'aide des sortes et des contours. La nécessité de ces restrictions supplémentaires sera discutée en section 2.6. Nous disons qu'un terme t est *linéaire* si chacune de ses variables n'apparaît qu'une seule fois dans t, c'est-à-dire si pour tout $x \in vars(t)$, il existe exactement une position de t tel que $t|_p = x$. En dehors de la propriété classique des sous-termes, l'objet des hypothèses qui suivent est de limiter la non-linéarité des règles.

Soit $(\Sigma, \mathcal{R})$ une théorie à constructeurs. On dit que $(\Sigma, \mathcal{R})$ est une théorie *quasi-linéaire* si, pour chaque destructeur des d'arité n, il existe exactement une règle $\ell_{\mathsf{des}} \to r_{\mathsf{des}} \in \mathcal{R}$, avec

1. $\ell_{\mathsf{des}} = \mathsf{des}(t_1, \dots, t_n)$ où chaque t_i est soit une variable, soit $\mathsf{sh}_{\mathsf{root}(t_i)}$ (à un renommage des variables près).
2. $r_{\mathsf{des}} \in \mathcal{T}_0(\Sigma_c, \emptyset) \cup St(t_1)$.
3. Ou bien ℓ_{des} est un terme linéaire, ou bien il existe une unique variable $x \in \mathcal{X}$ qui apparaît plusieurs fois dans ℓ_{des}, dont exactement une fois en position de clé de t_1.

Nous supposerons également que $\mathcal{R}$ contient au moins une règle non-linéaire.

Le premier item contraint les contours. Par exemple, la règle de déchiffrement asymétrique

$$\mathsf{adec}(\mathsf{aenc}(y, \mathsf{pub}(x)), x) \to y$$

impose que $\mathsf{sh}_{\mathsf{aenc}} = \mathsf{aenc}(y, \mathsf{pub}(x))$ (à un renommage près des variables). Comme les contours sont linéaires (voir section 1.1.5), chaque terme t_i est également linéaire. Le second item est une définition formelle de la propriété des sous-termes dans le cadre des théories à constructeurs. La propriété classique impose seulement que $r_{\mathsf{des}} \in St(\ell_{\mathsf{des}})$, mais comme r_{des} est un terme constructeur, nécessairement $r_{\mathsf{des}} \in \cup_{i=1}^{n} St(t_i)$. Le choix de t_1 est arbitraire et lève seulement la symétrie. Associé à l'item 2, le premier item garantit que si r_{des} contient une variable $y \in \mathcal{X}$, c'est-à-dire si $r_{\mathsf{des}} \in St(t_1) \setminus \mathcal{T}_0(\Sigma_c, \emptyset)$, alors la position p telle que $r_{\mathsf{des}} = t_1|_p$ est unique (sinon la variable y empêcherait t_1 d'être linéaire). Les sortes sont contraintes par le dernier item. En effet, les positions de clés sont définies comme les positions où un terme de sorte atom est attendu (voir section 1.1.4). Ce troisième item requiert que la position de clé à laquelle apparaît x se trouve dans t_1, c'est-à-dire dans le terme tel que $r_{\mathsf{des}} \in St(t_1)$. Rien n'indique que cette restriction soit importante, mais elle simplifie le contenu technique de ce chapitre, et n'élimine aucun exemple notable de primitive.

Exemple 2.1. *Soit n un entier. Considérons la signature standard Σ_n^{std} et les règles de réécriture associées $\mathcal{R}_n^{\mathsf{std}}$ introduites dans la section 1.1.7. $(\Sigma_n^{\mathsf{std}}, \mathcal{R}_n^{\mathsf{std}})$ est une théorie quasi-linéaire, comme les théories des exemples 1.9 et 1.11.*

Des exemples de théories qui n'entrent pas dans notre modèle seront donnés en section 2.6.

2.2 Typage

L'objet de ce chapitre est de limiter l'espace de recherche pour les traces de non-inclusion en se limitant aux traces bien typées. Un tel résultat est toujours vrai pour le système de type trivial où tous les termes ont le même type ; la section 2.2.1 définira les systèmes de types en général, ainsi que les systèmes structurés qui nous intéresseront plus particulièrement, puis la section 2.2.2 énoncera les conditions qu'un système de types doit vérifier, par rapport à un protocole donné, pour démontrer le résultat. La section 2.2.3 énoncera les théorèmes qui seront démontrés dans ce chapitre.

2.2.1 Système de types

Définissons d'abord un système de types en toute généralité.

Définition 2.1. *Un système de types est la donnée d'une paire* $(\mathcal{T}, \delta)$ *où* $\mathcal{T}$ *est un ensemble de* types, δ *une fonction qui envoie les termes de* $t \in \mathcal{T}_0(\Sigma_c, \Sigma_0^+ \cup \mathcal{N} \cup \mathcal{X})$ *sur des types de* $\mathcal{T}$, *et* $(\mathcal{T}, \delta)$ *vérifie les propriétés suivantes :*

- *Si* t *est un terme et* σ *une substitution bien typée, alors* $\delta(t\sigma) = \delta(t)$.
- *Pour tous sous-termes unifiables* t *et* t' *de même type, c'est-à-dire* $\delta(t) = \delta(t')$, *leur unificateur le plus général* $(mgu(t, t'))$ *est bien typé.*

où une substitution σ *est* bien typée *si pour tout* $d \in dom(\sigma)$, $\delta(d\sigma) = \delta(d)$.

Une exécution $\mathcal{K} \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ est *bien typée* si σ est une substitution bien typée, et lorsqu'il existe une telle exécution, on dit également que tr est bien typée. L'un des avantages à considérer seulement les exécutions bien typées consiste à pouvoir borner la taille des messages en termes de nombre d'opérations cryptographiques, c'est-à-dire leur profondeur comme termes. Les systèmes de types structurés donnent naturellement cette propriété.

Définition 2.2. *On appelle* système de types structuré *la donnée d'une paire* $(\mathcal{T}_0, \delta_0)$ *où* $\mathcal{T}_0$ *est un ensemble de* types initiaux *et* δ_0 *une fonction qui envoie les données de* $\Sigma_0^+ \uplus \mathcal{N} \uplus \mathcal{X}$ *vers les types* τ *engendrés par la grammaire suivante :*

$$\tau, \tau_1, \tau_2 = \tau_0 \mid \mathsf{f}(\tau_1, \ldots, \tau_n) \text{ avec } \mathsf{f} \in \Sigma_c \text{ et } \tau_0 \in \mathcal{T}_0$$

Ensuite, δ_0 *est étendue aux termes constructeurs de la façon suivante :*

$$\delta_0(\mathsf{f}(t_1, \ldots, t_n)) = \mathsf{f}(\delta_0(t_1), \ldots, \delta_0(t_n)) \text{ avec } f \in \Sigma_c$$

Comme la définition 2.2 ne l'indique pas directement, le lemme suivant montre que les systèmes de types structurés sont bien des systèmes de types.

Lemme 2.1. *Soit* $(\mathcal{T}_0, \delta)$ *un système de types structuré. Alors* $(\mathcal{T}(\Sigma_c, \mathcal{T}_0), \delta)$ *est un système de types comme dans la définition 2.1.*

Démonstration. Nous devons prouver les deux énoncés suivants :

- Si t est un terme et σ une substitution bien typée, alors $\delta(t\sigma) = \delta(t)$.
- Pour tous termes unifiables t et t' de même type, leur unificateur le plus général $mgu(t, t')$ est bien typé.

Le premier item se prouve par récurrence sur t. Si t est une donnée et $t \notin dom(\sigma)$, alors $t\sigma = t$ d'où $\delta(t\sigma) = \delta(t)$. Si $t \in dom(\sigma)$, alors $\delta(t\sigma) = \delta(t)$ car σ est bien typée. Si $t = \mathsf{f}(t_1, \ldots, t_n)$, alors $\delta(\mathsf{f}(t_1, \ldots, t_n)) = \mathsf{f}(\delta(t_1), \ldots, \delta(t_n))$ par définition de δ pour les termes composés. De même :

$$\begin{aligned}\delta(t\sigma) &= \delta(\mathsf{f}(t_1, \ldots, t_n)\sigma) \\ &= \delta(\mathsf{f}(t_1\sigma, \ldots, t_n\sigma)) \\ &= \mathsf{f}(\delta(t_1\sigma), \ldots, \delta(t_n\sigma))\end{aligned}$$

Par hypothèse de récurrence, pour chaque i, $\delta(t_i\sigma) = \delta(t_i)$ et donc $\delta(t\sigma) = \delta(t)$, ce qui conclut la preuve de cet item.

Prouvons maintenant le second item. Étant donné un ensemble Γ d'équations bien typées, on note $\#vars(\Gamma)$ le nombre de variables de Γ, et $|\Gamma|$ sa taille, à savoir

$$\sum_{t=t' \in \Gamma} (|t| + |t'|)$$

où $|t|$ est le nombre de symboles dans t. Notre mesure $\|\Gamma\|$ est déterminée par ces deux éléments dans l'ordre lexicographique. Prouvons que $mgu(\Gamma)$ est bien typé par récurrence sur $\|\Gamma\|$, en nous appuyant sur cette mesure.

Initialisation. $\|\Gamma\| = (0, 0)$, c'est-à-dire $\Gamma = \emptyset$, et donc le résultat est évident.

Hérédité. $\Gamma = \Gamma' \uplus \{t = t'\}$. Plusieurs cas se présentent :

— t **ou** t' **est une variable.** Alors on suppose, sans perte de généralité, que t est une variable x, et on pose $\sigma = \{x \triangleright t'\}$. σ est bien typée car les équations de Γ sont bien typées. De plus, en appliquant l'hypothèse de récurrence sur Γ', on déduit que $mgu(\Gamma')$ est bien typé, et donc la substitution $\{x \triangleright t' mgu(\Gamma')\}$ est bien typée. Il en découle que $mgu(\Gamma) = mgu(\Gamma') \uplus \{x \triangleright t' mgu(\Gamma')\}$ est bien typé.

— t **est une donnée, mais pas une variable.** Dans ce cas, t' est aussi une variable puisque $\delta(t) = \delta(t')$, et t' n'est pas une variable (ou bien on est ramené au cas précédent). Comme t et t' sont unifiables, $t = t'$ et $mgu(\Gamma) = mgu(\Gamma')$, avec $\|\Gamma'\| < \|\Gamma\|$. Par hypothèse de récurrence, $mgu(\Gamma')$ est bien typé, donc $mgu(\Gamma)$ est bien typé.

— t **est un terme composé :** $t = \mathsf{f}(t_1, \ldots, t_k)$**.** Dans ce cas, $t' = \mathsf{f}(t'_1, \ldots, t'_k)$ car t' n'est pas une variable et t et t' sont unifiables. On en déduit que $mgu(\Gamma) = mgu(\Gamma'')$ avec $\Gamma'' = \Gamma \uplus \{t_1 = t'_1, \ldots, t_k = t'_k\}$. Γ'' est un ensemble d'équations bien typées et $\|\Gamma''\| < \|\Gamma\|$ donc $mgu(\Gamma'')$ est bien typé par hypothèse de récurrence.

□

Dans la suite, nous supposerons l'existence d'un nombre infini de constantes de chaque type dans chacun des ensembles Σ_0^-, $\Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ et $\Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$. L'exemple suivant donne un exemple de système de types structuré, à partir du protocole d'Otway-Rees introduit dans le chapitre précédent.

Exemple 2.2. *La modélisation du protocole d'Otway-Rees, dans l'exemple 1.12, s'appuyait sur les données*

$$\begin{aligned} x, y_a, y_b, y_m, z_m, z^1_{\mathsf{senc}}, z^2_{\mathsf{senc}}, z_{ab} &\in \mathcal{X} \\ a, b \in \Sigma_0^-, \quad m, n_a, n_b, k_{as}, k_{bs}, k_{ab} &\in \mathcal{N} \end{aligned}$$

Considérons l'ensemble de types initiaux $\mathcal{T}_0 = \{\tau_a, \tau_b, \tau_m, \tau_{na}, \tau_{nb}, \tau_{as}, \tau_{bs}, \tau_{ab}\}$*. La fonction de typage* δ *se définit naturellement sur les noms et les constantes, par exemple* $\delta(a) = \tau_a$*,* $\delta(m) = \tau_m$*,*

$\delta(n_a) = \tau_{na}$, $\delta(k_{as}) = \tau_{as}$. *Sur les variables, on peut donner le type attendu pour une exécution honnête :*

$$\delta(x) = \delta(z_{ab}) = \tau_{ab} \quad \delta(y_a) = \tau_{na} \quad \delta(y_b) = \tau_{nb} \quad \delta(y_m) = \delta(z_m) = \tau_m$$
$$\delta(z^1_{\mathsf{senc}}) = \mathsf{senc}(\langle \tau_{na}, \tau_m, \tau_a, \tau_b \rangle^4, \tau_{as}) \qquad \delta(z^2_{\mathsf{senc}}) = \mathsf{senc}(\langle \tau_{na}, \tau_{ab} \rangle^2, \tau_{as})$$

La substitution σ de l'exemple 1.13 est bien typée pour ce système de type, et le témoin tr' de l'exemple 1.16 est donc un témoin bien typé. Si l'on avait donné des types différents à k_{ab} et x, alors il n'y aurait eu aucun témoin de non-inclusion bien typé, puisque $\mathsf{senc}(\langle n_a, k_{ab} \rangle^2, k_{as})$ est le seul message connu de l'attaquant qui peut s'unifier avec $\mathsf{senc}(\langle n_a, k_{ab} \rangle^2, k_{as})$, et que cette unification serait interdite par le système de types.

Un autre système de types $(\mathcal{T}'_0, \delta')$ intéressant consiste à prendre $\mathcal{T}'_0 = \mathcal{T}_0 \uplus \{\tau_{fwd}\}$ et δ' qui coïncide avec δ sauf pour $\delta'(z^1_{\mathsf{senc}}) = \delta'(z^2_{\mathsf{senc}}) = \tau_{fwd}$, De cette manière, le témoin tr' est mal typé, mais il existe un autre témoin (par rapport à Σ_0^+)

$$\mathsf{tr}'' = \mathsf{out}(c_A, \mathsf{w}_1).\mathsf{in}(c_B, R_1).\mathsf{out}(c_B, \mathsf{w}_2).\mathsf{in}(c_S, R_2).\mathsf{out}(c_S, \mathsf{w}_3).\mathsf{in}(c_A, R).\mathsf{phase}\ 1.\mathsf{out}(c, \mathsf{w}_4)$$

où R a été définie à l'exemple 1.13, et :

$$R_1 = \langle \mathsf{proj}^4_1(\mathsf{w}_1), a, b, c_{fwd} \rangle^4$$
$$R_2 = \langle \mathsf{proj}^4_1(\mathsf{w}_1), a, b, \mathsf{proj}^4_4(\mathsf{w}_1), \mathsf{proj}^4_4(\mathsf{w}_2) \rangle^5$$

avec $c_{fwd} \in \Sigma_0^+$ une constante de l'attaquant de type $\delta(c_{fwd}) = \tau_{cfw}$. Il est facile de transformer tr'' en un témoin par rapport à Σ_0^- en appliquant le renommage $\rho = \{c_{fwd} \triangleright c\}$ avec $c \in \Sigma_0^-$: $\mathsf{tr}''\rho$ est alors un témoin par rapport à Σ_0^-. L'utilité des constantes de Σ_{fresh} sera plus claire à partir de l'exemple 2.8. Les exécutions bien typées pour $(\mathcal{T}'_0, \delta')$ ont l'avantage de ne pas affecter de messages composés aux variables z^1_{senc} et z^2_{senc}.

2.2.2 Conformité

Nous avons vu dans l'exemple 2.2 qu'il n'était pas possible d'exiger des témoins bien typés pour certains systèmes de types trop contraignants. Plus précisément, lorsque le système de types empêche certaines unifications, une partie des processus peut se transformer en code mort. Une restriction naturelle consisterait à imposer que tous les sous-termes unifiables aient le même type, mais alors seuls des systèmes de types triviaux seraient autorisés. Il faut donc choisir une hypothèse plus fine, et qui concerne uniquement les sous-termes chiffrés. Ces sous-termes sont définis par opposition aux symboles transparents, c'est-à-dire inversibles.

Formellement, soit n un entier et $\mathsf{f} \in \Sigma_c$ un symbole d'arité n. Nous disons que f est *transparent* s'il existe un terme $\mathsf{f}(R^{\mathsf{f}}_1, \ldots, R^{\mathsf{f}}_n) \in \mathcal{T}(\Sigma, \Box)$, noté C_{f}, tel que pour tout $t \in \mathcal{T}_0(\Sigma, \Sigma_0^+ \uplus \mathcal{N} \uplus \mathcal{X})$ avec $\mathsf{root}(t) = \mathsf{f}$, on a $\mathsf{f}(R^{\mathsf{f}}_1, \ldots, R^{\mathsf{f}}_n)\{\Box \triangleright t\}{\downarrow} = t$. Pour tout terme t, on note $C_{\mathsf{f}}[t] = C_{\mathsf{f}}\{\Box \triangleright t\}$. Intuitivement, les symboles transparents correspondent à des termes qui peuvent toujours être ouverts par l'attaquant, comme les paires ou les tuples. Les sous-termes chiffrés $ESt(t)$ d'un terme t sont les sous-termes de t dont la racine n'est pas transparente, c'est-à-dire

$$ESt(t) = \{u \in St(t) \mid u \text{ est un terme composé et } \mathsf{root}(u) \text{ n'est pas transparent.}\}$$

Exemple 2.3. *Considérons les symboles $\langle\rangle^2$ et senc de la signature Σ^{std}_n définie en section 1.1.7. Pour tout message t avec $\mathsf{root}(t) = \langle \cdot, \cdot \rangle^2$, $t = \langle \mathsf{proj}^2_1(t), \mathsf{proj}^2_2(t) \rangle^2{\downarrow}$, et donc $C_{\langle\rangle^2} = \langle \mathsf{proj}^2_1(\Box), \mathsf{proj}^2_2(\Box) \rangle^2$. Au contraire, le symbole de fonction senc n'est pas transparent car la seule règle qui contient senc ne permet pas d'en déduire la clé.*

Les sous-termes chiffrés d'une configuration initiale $(\mathcal{P}; \phi; \emptyset; i)$ sont les sous-termes chiffrés de $\mathcal{P}$ et ceux de ϕ. Nous pouvons maintenant exprimer notre hypothèse sur les systèmes de types, qui suit celles de Chrétien, Cortier et Delaune [51] et Blanchet et Podelski [38].

Définition 2.3. *Soit Σ_0 un ensemble de constantes. Soit $\mathcal{K}$ une Σ_0-configuration initiale. On dit que $\mathcal{K}$ est* conforme *au système de types $(\mathcal{T}, \delta)$ si tous les sous-termes chiffrés unifiables ont le même type, c'est-à-dire si pour tous $t, t' \in ESt(\mathcal{K})$, $\delta(t) = \delta(t')$ chaque fois que t et t' sont unifiables.*

Dans les cas usuels, il est facile de savoir quels sont les symboles transparents, et les contraintes sur nos règles de réécriture, comme la propriété des sous-termes pour les théories à constructeurs, doivent permettre de décider si un symbole est transparent ou non. Cependant, comme la conformité porte uniquement sur les sous-termes chiffrés, il est superflu de savoir exactement quels sont les symboles transparents : simplement, lorsque nous savons qu'un symbole est transparent, l'hypothèse nous autorise à ne pas prendre en compte certaines unifications.

Revenons à notre exemple de référence.

Exemple 2.4. *Les processus P'_{OR} et P''_{OR} introduits par l'exemple 1.16 sont conformes aux systèmes de types $(\mathcal{T}_0, \delta)$ et $(\mathcal{T}'_0, \delta')$ décrit dans l'exemple 2.2. En effet, les sous-termes chiffrés de P'_{OR} et P''_{OR} sonti ceux de P_{OR}, c'est-à-dire :*

$$\mathsf{senc}(\langle n_a, m, a, b\rangle^4, k_{as})\ \mathsf{senc}(\langle n_a, x\rangle^2, k_{as})\ \ \mathsf{senc}(\langle y_a, y_m, a, b\rangle^4, k_{as})\ \mathsf{senc}(\langle y_a, k_{ab}\rangle^2, k_{as})$$
$$\mathsf{senc}(\langle y_b, y_m, a, b\rangle^4, k_{bs})\ \mathsf{senc}(\langle n_b, z_{ab}\rangle^2, k_{bs})\ \mathsf{senc}(\langle n_b, z_m, a, b\rangle^4, k_{bs})\ \mathsf{senc}(\langle y_b, k_{ab}\rangle^2, k_{bs})$$

Les termes ne sont unifiables que s'ils sont chiffrés par la même clé (k_{as} ou k_{bs}). Une fois la clé fixée, la taille des tuples empêche toutes les unifications, hormis les unifications honnêtes, et donc les protocoles P'_{OR} et P''_{OR} sont conformes à $(\mathcal{T}_0, \delta)$. Comme $(\mathcal{T}'_0, \delta')$ ne diffère de $(\mathcal{T}_0, \delta)$ que sur z^1_{senc} et z^2_{senc}, qui n'apparaissent pas dans les sous-termes chiffrés, les protocoles P'_{OR} et P''_{OR} sont également conformes à $(\mathcal{T}'_0, \delta')$. Pour chacun de ces systèmes de types, l'exemple 2.2 donnait un témoin bien typé.

Au contraire, ces deux protocoles ne sont pas conformes à l'autre système de types envisagé dans l'exemple 2.2, où x et k_{ab} ne sont pas de même type, car alors les sous-termes chiffrés unifiables $\mathsf{senc}(\langle n_a, x\rangle^2, k_{as})$ et $\mathsf{senc}(\langle y_a, k_{ab}\rangle^2, k_{as})$ n'ont pas le même type. Rappelons qu'il n'existe aucun témoin bien typé pour ce système de types.

L'objet de ce chapitre est de démontrer que l'exemple 2.4 se généralise, c'est-à-dire que pour tous protocoles et tout système de types, lorsque les protocoles sont conformes au système de types, il suffit d'explorer les traces bien typées pour démontrer l'équivalence des protocoles.

2.2.3 Énoncé des résultats du chapitre

Dans ce chapitre, nous démontrons que, pour l'accessibilité et l'équivalence, s'il existe une trace tr, il existe une trace tr' bien typée telle que tr' et tr partagent la même séquence d'émissions (out) et de réceptions (in) de messages sur les mêmes canaux. En d'autres termes, tr et tr' ont le même squelette $\overline{\mathsf{tr}}$.

Pour chaque Σ_0-trace tr, la séquence $\overline{\mathsf{tr}}$ est définie récursivement par :

— Si tr est vide, alors $\overline{\mathsf{tr}} = \mathsf{tr}$.

— Si $\mathsf{tr} = \mathsf{tr}'.\mathsf{phase}\ \ i$ pour un certain entier i, alors $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}.\mathsf{phase}\ \ i$.

— Si $\mathsf{tr} = \mathsf{tr}'.\mathsf{out}(c, \mathsf{w})$ pour un certain $\mathsf{w} \in \mathcal{W}$, alors $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}.\mathsf{out}(c, \_)$.

— Si $\mathsf{tr} = \mathsf{tr}'.\mathsf{in}(c, R)$ pour une certaine Σ_0-recette R alors $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}.\mathsf{in}(c, \_)$.

Le résultat constitue une étape intermédiaire pour le théorème principal, qui porte sur l'équivalence, mais possède un intérêt indépendant.

Théorème 2.1. *Soit $\mathcal{K}_P$ une Σ_0^--configuration conforme à un système de types $(\mathcal{T}_0, \delta_0)$. Si $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^- alors il existe une exécution bien typée $\mathcal{K}_P \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}; \phi'; \sigma'; i)$ par rapport à Σ_0^+ telle que $\overline{\mathsf{tr}'} = \overline{\mathsf{tr}}$.*

Réciproquement, si $\mathcal{K}_P \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}; \phi'; \sigma'; i)$ est une exécution bien typée par rapport à Σ_0^+, alors il existe une exécution $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^- telle que $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}$.

Ce résultat est suffisant pour modéliser certaines propriétés d'accessibilité, comme le secret. Par exemple, toute trace de $P|\mathsf{in}(c, s)$ contenant une réception sur le canal c est un témoin d'attaque sur le secret de $s \in \mathcal{N}$, pourvu que c n'apparaisse pas dans le processus P. La démonstration de ce théorème sera exposée dans la section 2.4. Les constantes de Σ_{fresh} permettent de remplacer des termes composés d'une exécution arbitraire dans une exécution bien typée : le terme t sera remplacé par une constante $c \in \Sigma_{\mathsf{fresh}}$ du bon type.

Nous en arrivons au théorème principal de cette section.

Théorème 2.2. *Soient $\mathcal{K}_P$ une Σ_0^--configuration conforme à $(\mathcal{T}_0, \delta_0)$ et $\mathcal{K}_Q$ une Σ_0^--configuration déterministe. On a $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^- si, et seulement si, il existe un témoin bien typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ de cette non-inclusion.*

Seule la configuration $\mathcal{K}_P$ doit être conforme au système de types, puisque la trace n'est bien typée que par rapport à $\mathcal{K}_P$. De plus, comme l'inclusion de trace stipule que le protocole $\mathcal{K}_Q$ doit pouvoir imiter toute exécution de $\mathcal{K}_P$, l'attaquant peut choisir l'exécution exacte de $\mathcal{K}_P$ qui doit être imitée, donc le déterminisme n'est utile que pour $\mathcal{K}_Q$. Ce théorème sera démontré précisément dans la section 2.5.

2.3 Préliminaires techniques

Comme les démonstrations des résultats de typages sont relativement techniques, nous exposons séparément des notions utiles pour les mener à bien. D'abord, la réduction forcée, exposée dans la section 2.3.1, permet de calculer l'effet potentiel des destructeurs sans tenir compte de l'affectation des variables de la règle associée. Ensuite, la mesure présentée en section 2.3.2 indique lorsqu'une recette s'approche de la recette utilisée dans une trace bien typée. Enfin, la section 2.3.3, qui est la plus technique, présente la notion de *substitution au premier ordre* permettant de faire le lien entre une trace arbitraire et la trace bien typée qui la représente.

2.3.1 Réduction forcée

Cette section introduit la réduction forcée $\twoheadrightarrow$ et sa forme normale associée $u{\downarrow}$ pour un terme u quelconque. Il s'agit de la réduction obtenue en tenant compte seulement des destructeurs et des constructeurs, par exemple en déchiffrant un message chiffré, même si la clé utilisée n'est pas la bonne. La définition suivante formalise cette idée, pour toute théorie quasi-linéaire $(\Sigma, \mathcal{R})$.

Définition 2.4. *Soit $(\Sigma, \mathcal{R})$ une théorie quasi-linéaire. Soit $\ell_{\mathsf{des}} \to r_{\mathsf{des}} \in \mathcal{R}$ et $\mathsf{des} = \mathsf{root}(\ell_{\mathsf{des}})$. La règle de réduction forcée associée est $\ell'_{\mathsf{des}} \twoheadrightarrow r_{\mathsf{des}}$, où ℓ'_{des} est obtenue à partir de ℓ_{des} en gardant seulement le chemin vers r dans ℓ_{des}. Formellement, ℓ'_{des} est définie par :*

1. *$\ell'_{\mathsf{des}} = \mathsf{des}(x_1, \ldots, x_n)$ quand r est un terme clos ;*
2. *Sinon, notons p_0 l'unique position de ℓ_{des} telle que $\ell_{\mathsf{des}}|_{p_0} = r$ et $p_0 = 1.p'_0$, ℓ'_{des} est le terme linéaire tel que :*

— *pour chaque préfixe* p' *de* p_0, $\mathsf{root}(\ell'_{\mathsf{des}}|_{p'}) = \mathsf{root}(\ell_{\mathsf{des}}|_{p'})$
— $\ell'_{\mathsf{des}}|_{p_0} = r = \ell_{\mathsf{des}}|_{p_0}$
— *Pour toute autre position* p' *de* ℓ'_{des}, $\ell'_{\mathsf{des}}|_{p'}$ *est une variable.*

Rappelons que l'unicité de la position p_0 telle que $r_{\mathsf{des}} = \ell_{\mathsf{des}}|_{p_0}$ est garantie par la linéarité du terme t_1. De plus, l'unicité de la règle associée à chaque destructeur impose que le système de réécriture forcée soit confluent ; il ne serait pas suffisant d'interdire les paires critiques, car la transformation qui associe les règles forcées aux règles de départ ne préserve pas cette propriété.

Exemple 2.5. *Reprenons la théorie standard* $(\Sigma_n^{\mathsf{std}}, \mathcal{R}_n^{\mathsf{std}})$ *de la section 1.1.7. Pour tous entiers* $i \leqslant n$, *les règles de réécriture forcée associées aux projections des tuples* $\ell_{\mathsf{proj}_i^n} \to r_{\mathsf{proj}_i^n}$ *sont identiques à ces règles :* $\ell'_{\mathsf{proj}_i^n} = \ell_{\mathsf{proj}_i^n}$. *Les autres règles donnent :*

$$\begin{array}{rlrl} \mathsf{adec}(\mathsf{aenc}(x,y),z) & \twoheadrightarrow x & \mathsf{sdec}(\mathsf{senc}(x,y),z) & \twoheadrightarrow x \\ \mathsf{getmsg}(\mathsf{sign}(x,y),z) & \twoheadrightarrow x & \mathsf{check}(x,y) & \twoheadrightarrow \mathsf{ok} \end{array}$$

Étant donné un ensemble $\mathcal{R}_f$ de règles de réécriture forcée, un terme u peut être réécrit par $\mathcal{R}_f$ s'il existe un sous-terme de u qui s'unifie avec le membre gauche de la règle. Formellement, on note $u \twoheadrightarrow v$ s'il existe une règle $\ell \twoheadrightarrow r \in \mathcal{R}_f$, une position p et une substitution σ tels que $u|_p = \ell\sigma$ et $v = u[r\sigma]_p$. En particulier, contrairement à la réduction définie au chapitre précédent, la réduction forcée se produit même lorsque les sous-termes ne sont pas des messages. Comme toujours, nous noterons $\twoheadrightarrow^*$ la clôture réflexive-transitive de $\twoheadrightarrow$. Ce système de réécriture est convergent car il termine (chaque réduction supprime un destructeur) et n'a pas de paires critiques. La forme normale associée est notée $u{\downarrow}$.

Nous l'appliquerons aux recettes pour les simplifier et éviter les détours. Le lemme suivant démontre qu'un terme déduit par la recette R dans une trame ϕ donnée est identique à celui qui serait obtenu par la recette $R{\downarrow}$, pourvu que $R\phi{\downarrow}$ soit un message.

Lemme 2.2. *Soient* ϕ *une* Σ_0*-trame et* R *une* Σ_0*-recette telles que* $R\phi{\downarrow}$ *est un* Σ_0*-message. Si* R' *vérifie* $R \twoheadrightarrow R'$, *alors* R' *est une* Σ_0*-recette, et* $R'\phi{\downarrow} = R\phi{\downarrow}$.

Démonstration. Comme $R \twoheadrightarrow R'$, il existe une position p dans R, une règle de réécriture forcée $\ell' \twoheadrightarrow r$ associée à une règle $\ell \to r \in \mathcal{R}$, et une substitution σ telle que $R|_p = \ell'\sigma$ et $R' = R[r\sigma]_p$. Comme $R\phi{\downarrow}$ est un Σ_0-message, $(\ell'\sigma)\phi{\downarrow}$ est un Σ_0-message. Soient $t'_1, \ldots, t'_k$ et $\mathsf{des} \in \Sigma_d$ tels que $\ell' = \mathsf{des}(t'_1, \ldots, t'_k)$. On a $(\ell'\sigma)\phi = \mathsf{des}((t'_1\sigma)\phi, \ldots, (t'_k\sigma)\phi)$. Comme $(\ell'\sigma)\phi{\downarrow}$ est un Σ_0-message, $\ell \to r$ s'applique, puisqu'il n'existe qu'une seule règle qui réduise des. On en déduit que $\mathsf{des}((t'_1\sigma)\phi{\downarrow}, \ldots, (t'_k\sigma)\phi{\downarrow}) \to (r\sigma)\phi{\downarrow}$ et donc que $(\ell'\sigma)\phi{\downarrow} = (r\sigma)\phi{\downarrow}$, d'où $R\phi{\downarrow} = R'\phi{\downarrow}$. □

Dans les développements techniques, une partie du travail consistera à montrer qu'il est suffisant de considérer des recettes sous une forme simple, où la phase d'analyse (déduction de messages, essentiellement par des destructeurs) est séparée de la synthèse (ajout de termes constructeurs).

Définition 2.5. *Soit* R *une* Σ_0*-recette.* R *est une* recette de sous-terme *si* $R\phi{\downarrow} \in St(\phi)$ *pour toute* Σ_0*-trame* ϕ *telle que* $R\phi{\downarrow}$ *est un* Σ_0*-message.* R *est* simple *si* $R = C[R_1, \ldots, R_k]$, *où* C *est un contexte (c'est-à-dire un terme à trous) construit sur* $\Sigma_c \uplus \Sigma_0$, *et chaque* R_i *est une* Σ_0*-recette de sous-terme telle que* $\mathsf{root}(R_i) \notin \Sigma_c$.

L'exemple suivant présente des recettes simples et des recettes de sous-terme, et illustre la différence entre ces dernières et des recettes composées de destructeurs.

Exemple 2.6. *Si nous considérons la signature standard* $(\Sigma_n^{\mathsf{std}}, \mathcal{R}_n^{\mathsf{std}})$ *définie dans la section 1.1.7,* $R = \mathsf{adec}(\mathsf{w}_1, \mathsf{w}_2)$ *est une* Σ_0^-*-recette de sous-terme, car* $\mathsf{adec}(\mathsf{w}_1, \mathsf{w}_2)\phi{\downarrow}$ *est un sous-terme de* $\mathsf{w}_1\phi$ *pour toute* Σ_0^-*-trame* ϕ *telle que* $R\phi{\downarrow}$ *est un* Σ_0^-*-message. Comme* adec *n'est pas un constructeur,* $\mathsf{senc}(R, \mathsf{w}_3)$ *est une* Σ_0^-*-recette simple. La recette* $\mathsf{check}(\mathsf{w}_1, \mathsf{w}_2)$ *n'est pas non plus une recette de sous-terme, et la recette* ok *est une recette équivalente du point de vue de la déduction de l'attaquant.*

Considérons maintenant la signature $\Sigma = \Sigma_d \uplus \Sigma_c$ *avec* $\Sigma_d = \{\mathsf{des}\}$ *et* $\Sigma_c = \{\mathsf{f}\ ;\ \mathsf{g}\}$ *et la règle*

$$\mathsf{des}(\mathsf{f}(\mathsf{g}(x))) \to x$$

$R' = \mathsf{des}(\mathsf{f}(\mathsf{w}_1))$ *n'est pas composée uniquement de destructeurs, mais pour toute* Σ_0^-*-trame* ϕ *telle que* $R'\phi{\downarrow}$ *est un* Σ_0^-*-message,* $R'\phi{\downarrow}$ *est un sous-terme de* $\mathsf{w}_1\phi$*. Donc* R' *est une recette de sous-terme.*

Il existe un lien entre la réduction forcée et les recettes simples : lorsqu'une recette est la recette d'un message, sa forme normale forcée est simple, comme le montre le lemme suivant, où les constantes de Σ_{fresh} sont remplacées par des Σ_0^--recettes.

Lemme 2.3. *Soit* θ *une substitution avec* $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$ *et dont l'image est constituée de* Σ_0^-*-recettes. Soit* R *une* Σ_0^+*-recette en forme normale forcée telle que* $(R\theta)\phi{\downarrow}$ *est un* Σ_0^+*-message pour une certaine* Σ_0^-*-trame* ϕ*. Toute* Σ_0^+*-recette* $R' \in St(R)$ *est simple.*

Démonstration. Ce résultat se démontre par récurrence sur R.
Initialisation. $R \in \mathcal{W} \uplus \Sigma_0^+$. Le résultat est vrai car R est une Σ_0^+-recette simple.
Hérédité. $R = \mathsf{f}(R_1, \ldots, R_k)$ pour un certain $\mathsf{f} \in \Sigma$. $R_1, \ldots, R_k$ sont en forme normale pour $\twoheadrightarrow$.

— *Cas* $\mathsf{f} \in \Sigma_c$. On a $(R\theta)\phi{\downarrow} = \mathsf{f}((R_1\theta)\phi{\downarrow}, \ldots, (R_k\theta)\phi{\downarrow})$, et $(R\theta)\phi{\downarrow}$ est un Σ_0^+-message pour une certaine Σ_0^--trame ϕ, donc $(R_i\theta)\phi{\downarrow}$ est un Σ_0^+-message pour chaque $i \in \{1; \ldots; k\}$. La conclusion découle directement de l'application de l'hypothèse de récurrence à R_i pour chaque i.

— *Cas* $\mathsf{f} = \mathsf{des} \in \Sigma_d$. Soient $\ell_{\mathsf{des}} \to r_{\mathsf{des}}$ la règle de $\mathcal{R}$ telle que $\mathsf{root}(\ell_{\mathsf{des}}) = \mathsf{des}$, et $\ell'_{\mathsf{des}} \twoheadrightarrow r_{\mathsf{des}}$ la règle de réduction forcée associée. Si $r_{\mathsf{des}} \in \mathcal{T}_0(\Sigma_c, \emptyset)$, alors $R \twoheadrightarrow r_{\mathsf{des}}$, ce qui est impossible car R est en forme normale pour $\twoheadrightarrow$. Donc il existe une unique position p_0 de ℓ_{des} telle que $\ell_{\mathsf{des}}|_{p_0} = r_{\mathsf{des}}$ et $p_0 = 1.p'_0$. Rappelons que chaque R_i est en forme normale pour $\twoheadrightarrow$, et $(R_i\theta)\phi{\downarrow}$ est un Σ_0^+-message (pour $1 \leqslant i \leqslant k$) puisque $(R\theta)\phi{\downarrow}$ est un Σ_0^+-message. L'hypothèse de récurrence s'applique donc : tout sous-terme de R_i est une Σ_0^+-recette simple (pour $1 \leqslant i \leqslant k$). Supposons que R_1 est une Σ_0^+-recette de sous-terme. Soit ψ une Σ_0^+-trame telle que $R\psi{\downarrow}$ est un Σ_0^+-message. Comme $r_{\mathsf{des}} = \ell_{\mathsf{des}}|_{p_0}$, $R\psi{\downarrow} \in St(R_1\psi{\downarrow}) \subseteq St(\psi)$. On en déduit que R est une Σ_0^+-recette de sous-terme, et donc une recette simple.

Nous avons traité le cas où R_1 est une recette de sous-terme. Supposons donc maintenant que $R_1 = C[R'_1, \ldots, R'_{k'}]$ pour un certain contexte C construit sur les symboles de $\Sigma_c \uplus \Sigma_0^+$, où chaque R'_j (avec $1 \leqslant j \leqslant k'$) est une Σ_0^+-recette de sous-terme telle que $\mathsf{root}(R'_j) \notin \Sigma_c$. Dans ce cas, $R = \mathsf{des}(C[R'_1, \ldots, R'_{k'}], R_2, \ldots, R_k)$, et p_0 ne correspond pas à une position du contexte C puisque R est en forme normale pour $\twoheadrightarrow$. Soit p' le plus long préfixe de p'_0 qui corresponde à une position de C. On a $C[R'_1, \ldots, R'_{k'}]|_{p'} = R'|_j$ pour un certain j. Soit ψ une Σ_0^+-trame telle que $R\psi{\downarrow}$ est un Σ_0^+-message. Alors $R\psi{\downarrow} \in St(R'_j\psi{\downarrow}) \subseteq St(\psi)$ car R'_j est une Σ_0^+-recette de sous-terme. On en déduit que R est une Σ_0^+-recette de sous-terme, et donc une recette simple.

□

Associé au lemme 2.2, le lemme 2.3 montre que l'on peut remplacer les recettes arbitraires par des recettes simples sans changer les capacités de déduction de l'attaquant, et que la réduction forcée nous permet de calculer la recette simple correspondant à la recette de départ.

2.3.2 Mesure

Cette section introduit une mesure sur les recettes, qui constitue un élément essentiel pour la démonstration du résultat de typage. Le rôle de cette mesure est de permettre de transformer une trace quelconque en trace bien typée, en modifiant les recettes. Plutôt que d'envoyer des messages utilisant un grand nombre de symboles, l'attaquant peut se contenter d'envoyer de petits messages bien typés. Cette transformation dépend de la trace tr et de la trame ϕ_S considérée, donc ϕ_S est un paramètre de la mesure, qui sera déterminée par trois éléments, en ordre lexicographique :

1. Premièrement, la taille du terme calculé par R, c'est-à-dire $R\phi_S\downarrow$.
2. Deuxièmement, le chapeau de symboles constructeurs de R doit être maximal. Par exemple, $\langle \mathsf{proj}_1^2(\mathsf{w}), \mathsf{proj}_2^2(\mathsf{w})\rangle$ sera préférée à w si $\mathsf{w}\phi_S\downarrow$ est une paire. De cette manière, il est possible d'agir localement sur la recette : on pourra plus facilement remplacer $\langle \mathsf{proj}_1^2(\mathsf{w}), \mathsf{proj}_1^2(\mathsf{w})\rangle$ par $\langle a, a\rangle$ avec $a \in \Sigma_{\mathsf{fresh}}$ si nécessaire.
3. Enfin, la taille de la recette R elle-même.

Le but de cette section est de définir formellement cette mesure et d'établir certaines de ses propriétés. Étant donné un terme $t \in \mathcal{T}(\Sigma, \Sigma_0^+ \uplus \mathcal{N})$, notons $\mathsf{Multi}(t)$ le multi-ensemble d'éléments de $\Sigma \uplus \Sigma_0^+ \uplus \mathcal{N}$ défini comme suit :

— $\mathsf{Multi}(a) = \{a\}$ quand $a \in \Sigma_0^+ \uplus \mathcal{N}$;

— $\mathsf{Multi}(\mathsf{f}(t_1, \ldots, t_n)) = \{\mathsf{f}\} \uplus \mathsf{Multi}(t_1) \uplus \ldots \mathsf{Multi}(t_n)$ quand $\mathsf{f} \in \Sigma$.

Étant donné un ensemble de données D et un terme $t \in \mathcal{T}(\Sigma, D)$, la taille de t, notée $|t|$, est le nombre de symboles de fonction de t. Le *chapeau* de t est le contexte constructeur C (c'est-à-dire un terme à trous construit sur Σ_c) tel que $t = C[t_1, \ldots, t_n]$ avec $\mathsf{root}(t_1), \ldots, \mathsf{root}(t_n) \notin \Sigma_c$. Le chapeau de t sera noté $\mathsf{hat}(t)$.

Étant donné une Σ_0^+-trame ϕ_S et un ordre $\prec$ sur Σ_{fresh} (par exemple l'ordre d'apparition de ces constantes dans la trace tr_S associée à ϕ_S), la mesure μ_{ϕ_S} associée à une Σ_0^+-recette R est définie par les éléments suivants dans l'ordre lexicographique.

1. $\mu^1_{\phi_S}(R) = \mathsf{Multi}(R\phi_S\downarrow)$ où les éléments des multi-ensembles sont ordonnés par :

$$c_{min} < \Sigma_0^- \setminus \{c_{min}\} \uplus \Sigma_c < \Sigma_{\mathsf{fresh}} < \Sigma_d$$

   avec $c_{min} \in \Sigma_0^-$, et pour les éléments de Σ_{fresh}, on a $c < c'$ quand $c \prec c'$.
2. $\mu^2_{\phi_S}(R) = |R\phi_S\downarrow| - |\mathsf{hat}(R)|$.
3. $\mu^3_{\phi_S}(R) = |R|$.

Notons que $c_{min} \in \Sigma_0^-$ est la Σ_0^--recette minimale pour cette mesure. Nous commençons par établir quelques propriétés concernant cette mesure. D'abord, remarquons que la mesure μ^2 reste positive car aucun contexte constructeur ne peut se réécrire.

Lemme 2.4. *Soient ϕ_S une Σ_0^+-trame, et R une Σ_0^+-recette. On a $\mu^2_{\phi_S}(R) \geqslant 0$.*

Démonstration. Soit $R = R_0[R_1, \ldots, R_n]$ où R_0 est le chapeau de R. Comme R_0 ne contient que des constructeurs, on a $R\phi_S\downarrow = R_0[R_1\phi_S\downarrow, \ldots, R_n\phi_S\downarrow]$. On en déduit que $|R\phi_S\downarrow| \geqslant |R_0| = \mathsf{hat}(R)$, ce qui implique que $\mu^2_{\phi_S}(R) \geqslant 0$. □

Ensuite, si une recette R_2 qui ne se normalise pas vers un message est plus grande qu'une autre recette R_1, alors $R_0[R_2]$ reste plus grande que $R_0[R_1]$. Nous utiliserons ce résultat dans les preuves, afin de raisonner sur certaines réductions qui échouent à l'intérieur de termes plus grands. Comme ce lemme est assez technique, nous introduisons d'abord le lemme suivant qui démontre que toute réduction fait diminuer le terme réduit pour la mesure.

Lemme 2.5. *Soit $\prec$ un ordre sur Σ_{fresh}, $\ell_{\mathsf{des}} \to r_{\mathsf{des}}$ une règle de réécriture d'une théorie quasi-linéaire $\mathcal{R}$ et σ une substitution telle que $img(\sigma) \subseteq \mathcal{T}(\Sigma, \Sigma_0^+ \uplus \mathcal{N})$. On a $\mathsf{Multi}(r_{\mathsf{des}}\sigma) < \mathsf{Multi}(\ell_{\mathsf{des}}\sigma)$, et ce résultat vaut aussi pour la forme normale forcée.*

Démonstration. Premièrement, considérons le cas où $r_{\mathsf{des}} \in \mathcal{T}_0(\Sigma_c, \emptyset)$. On a alors :

$$\mathsf{Multi}(r_{\mathsf{des}}\sigma) = \mathsf{Multi}(r_{\mathsf{des}}) < \{\mathsf{des}\} < \mathsf{Multi}(\ell_{\mathsf{des}}\sigma)$$

Maintenant, considérons le cas où $r_{\mathsf{des}} \in St(\ell_{\mathsf{des}})$. On a :

$$\mathsf{Multi}(r_{\mathsf{des}}\sigma) < \{\mathsf{des}\} \uplus \mathsf{Multi}(r_{\mathsf{des}}\sigma) \leqslant \mathsf{Multi}(\ell_{\mathsf{des}}\sigma)$$

Dans chacun de ces cas, on a $\mathsf{Multi}(r_{\mathsf{des}}\sigma) < \mathsf{Multi}(\ell_{\mathsf{des}}\sigma)$. La preuve peut se faire de la même manière pour le cas des règles de réécriture forcée. □

Nous en déduisons que la normalisation fait diminuer la mesure d'un terme.

Lemme 2.6. *Soient $\prec$ un ordre sur Σ_{fresh}, $\mathsf{f} \in \Sigma$ d'arité k, et $t_1, \ldots, t_k \in \mathcal{T}(\Sigma, D)$ pour un certain ensemble D de données. On a $\mathsf{Multi}(\mathsf{f}(t_1, \ldots, t_k)\downarrow) \leqslant \mathsf{Multi}(\mathsf{f}(t_1\downarrow, \ldots, t_k\downarrow))$, et $\mathsf{Multi}(\mathsf{f}(t_1, \ldots, t_k)\downarrow\!\!\downarrow) \leqslant \mathsf{Multi}(\mathsf{f}(t_1\downarrow\!\!\downarrow, \ldots, t_k\downarrow\!\!\downarrow))$.*

Démonstration. Pour commencer, nous considérons le cas où $\mathsf{f}(t_1, \ldots, t_k)\downarrow = \mathsf{f}(t_1\downarrow \ldots, t_k\downarrow)$. Le résultat est trivial. Donc on a $\mathsf{f} = \mathsf{des} \in \Sigma_d$, et $\mathsf{des}(t_1, \ldots, t_k)\downarrow \neq \mathsf{des}(t_1\downarrow, \ldots, t_k\downarrow)$. On en déduit qu'il existe une substitution σ telle que

$$\mathsf{des}(t_1\downarrow, \ldots, t_k\downarrow) = \ell_{\mathsf{des}}\sigma \text{ and } \mathsf{des}(t_1, \ldots, t_k)\downarrow = r_{\mathsf{des}}\sigma$$

D'après le lemme 2.5, $\mathsf{Multi}(\mathsf{des}(t_1, \ldots, t_k)\downarrow) < \mathsf{Multi}(\mathsf{des}(t_1\downarrow, \ldots, t_k\downarrow))$, ce qui conclut la preuve. Un raisonnement similaire permet également de conclure pour $\downarrow\!\!\downarrow$. □

Nous pouvons finalement démontrer le lemme annoncé.

Lemme 2.7. *Soient $\prec$ un ordre sur Σ_{fresh}, ϕ_S une Σ_0^+-trame, R_1, R_2 deux Σ_0^+-recettes telles que $R_2\phi_S\downarrow$ n'est pas un Σ_0^+-message, et $\mu^1_{\phi_S}(R_1) < \mu^1_{\phi_S}(R_2)$. Soit R_0 une Σ_0^+-recette et p une position de R_0. On a :*

$$\mu^1_{\phi_S}(R_0[R_1]_p) < \mu^1_{\phi_S}(R_0[R_2]_p)$$

Démonstration. Commençons par établir le fait suivant, par induction sur p ($\Box$ n'est pas un message).

$$\mathsf{Multi}(R_0[R_1]_p\phi_S\downarrow) < (\mathsf{Multi}(R_0\phi_S[\Box]_p\downarrow)) \setminus \{\Box\} \uplus \mathsf{Multi}(R_1\phi_S\downarrow)$$

Initialisation : $p = \epsilon$. On a $\mathsf{Multi}(R_0\phi_S[\Box]_p\downarrow) = \{\Box\}$ donc

$$\mathsf{Multi}(R_0[R_1]_p\phi_S\downarrow) = \mathsf{Multi}(R_1\phi_S\downarrow) = (\mathsf{Multi}(R_0\phi_S[\Box]_p\downarrow) \setminus \{\Box\}) \uplus \mathsf{Multi}(R_1\phi_S\downarrow)$$

Hérédité : $p = j.p'$, et $R_0 = \mathsf{f}(R'_1, \ldots, R'_k)$ pour certaines recettes $R'_1, \ldots, R'_k$. D'après le lemme 2.6, on obtient $\mathsf{Multi}(R_0[R_1]_p\phi_S\downarrow) \leqslant \mathsf{Multi}(\mathsf{f}(R'_1\phi_S\downarrow, \ldots, R'_j[R_1]_{p'}\phi_S\downarrow, \ldots, R'_k\phi_S\downarrow))$, et donc

$$\mathsf{Multi}(R_0[R_1]_p\phi_S\downarrow) \leqslant \{\mathsf{f}\} \uplus (\uplus_{i\neq j}\mathsf{Multi}(R'_i\phi_S\downarrow)) \uplus \mathsf{Multi}(R'_j[R_1]_{p'}\phi_S\downarrow).$$

Par hypothèse d'induction, on a $\mathsf{Multi}(R'_j[R_1\phi_S\downarrow]_{p'}) \leqslant (\mathsf{Multi}(R'_j\phi_S[\Box]_{p'}\downarrow) \setminus \{\Box\}) \uplus \mathsf{Multi}(R_1\phi_S\downarrow)$ et donc :

$$\mathsf{Multi}(R_0[R_1]_p\phi_S\downarrow) \leqslant \{\mathsf{f}\} \uplus (\uplus_{i\neq j}\mathsf{Multi}(R'_i\phi_S\downarrow)) \uplus (\mathsf{Multi}(R'_j\phi_S[\Box]_{p'}\downarrow) \setminus \{\Box\}) \uplus \mathsf{Multi}(R_1\phi_S\downarrow)$$

Mais $R'_j\phi_S[\Box]_{p'}\downarrow$ n'est pas un message, donc aucune réduction ne se produit au-dessus de $\Box$. D'où :

$$\mathsf{Multi}(R_0\phi_S[\Box]_p\downarrow) = \{\mathsf{f}\} \uplus (\uplus_{i\neq j}\mathsf{Multi}(R'_i\phi_S\downarrow)) \uplus \mathsf{Multi}(R'_j[\Box]_{p'}\phi_S\downarrow).$$

On en déduit que :

$$\mathsf{Multi}(R_0\phi_S[\Box]_p\downarrow) \setminus \{\Box\} = \{\mathsf{f}\} \uplus (\uplus_{i\neq j}\mathsf{Multi}(R'_i\phi_S\downarrow) \uplus (\mathsf{Multi}(R'_j[\Box]_{p'}\phi_S\downarrow) \setminus \{\Box\})$$

Et on obtient :

$$\mathsf{Multi}(R_0[R_1]_p\phi_S\downarrow) \leqslant (\mathsf{Multi}(R_0[\Box]_p\phi_S\downarrow) \setminus \{\Box\}) \uplus \mathsf{Multi}(R_1\phi_S\downarrow)$$

Ce qui prouve le fait annoncé.

Maintenant, comme $R_2\phi_S\downarrow$ n'est pas un message, $R_0[R_2]_p\phi_S\downarrow = (\mathsf{Multi}(R_0\phi_S[\Box]_p\downarrow) \setminus \{\Box\}) \uplus \mathsf{Multi}(R_2\phi_S\downarrow)$. Puisque $\mu^1_{\phi_S}(R_1) < \mu^1_{\phi_S}(R_2)$, il est facile de prouver que $\mu^1_{\phi_S}(R_0[R_1]_p) < \mu^1_{\phi_S}(R_0[R_2]_p)$ grâce au fait qui a été démontré. □

Nous pouvons également montrer que la mesure diminue toujours après application de la réduction forcée.

Lemme 2.8. *Soient $\prec$ un ordre sur Σ_{fresh}, ϕ_S une Σ_0^+-trame et R, R' deux Σ_0^+-recettes telles que $R \twoheadrightarrow R'$. On a $\mu_{\phi_S}(R') < \mu_{\phi_S}(R)$.*

Démonstration. Considérons d'abord le cas où $R\phi_S\downarrow = R'\phi_S\downarrow$. On a $\mu^1_{\phi_S}(R) = \mu^1_{\phi_S}(R')$, et aussi $\mu^3(R) < \mu^3(R')$. Comme $|R\phi_S\downarrow| = |R'\phi_S\downarrow|$, il suffit de montrer que $|\mathsf{hat}(R')| \geqslant |\mathsf{hat}(R)|$ pour conclure. On a $R = R_0[R_1, \ldots, R_n]$ avec $R_0 = \mathsf{hat}(R)$. Comme $R \twoheadrightarrow R'$, on sait que $R' = R_0[R_1, \ldots, R'_i, \ldots, R_n]$ avec $R_i \twoheadrightarrow R'_i$, et donc $|\mathsf{hat}(R')| \geqslant |R_0|$.

Maintenant, considérons le cas où $R\phi_S\downarrow \neq R'\phi_S\downarrow$. Soit $\ell'_{\mathsf{des}} \twoheadrightarrow r_{\mathsf{des}}$ la règle appliquée pour réécrire R dans R'. On a $R = R[\ell'_{\mathsf{des}}\delta]_p$ et $R' = R[r_{\mathsf{des}}\delta]_p$ pour une certaine position p, et une certaine position δ. Donc on a $R\phi_S\downarrow = (R\phi_S)[(\ell'_{\mathsf{des}}\delta)\phi_S\downarrow]_p\downarrow$ et $R'\phi_S\downarrow = (R\phi_S)[(r_{\mathsf{des}}\delta)\phi_S\downarrow]_p\downarrow$. D'après le lemme 2.5, et comme $([\ell'_{\mathsf{des}}\delta]\phi_S\downarrow)$ ne se réduit pas (sinon on aurait $R\phi_S\downarrow = R'\phi_S\downarrow$) on a $\mathsf{Multi}((\ell'_{\mathsf{des}}\delta)\phi_S\downarrow) > \mathsf{Multi}([r_{\mathsf{des}}\delta]\phi_S\downarrow)$. On en déduit que $\mu^1_{\phi_S}(\ell'_{\mathsf{des}}\delta) > \mu^1_{\phi_S}(r_{\mathsf{des}}\delta)$. Le lemme 2.7 permet de conclure que $\mu^1_{\phi_S}(R') < \mu^1_{\phi_S}(R)$. □

L'exemple suivant reprend la mesure et illustre le lemme 2.8.

Exemple 2.7. *Considérons la Σ_0^+-trame $\phi_S = \{\mathsf{w}_1 \triangleright \langle m, a\rangle^2; \mathsf{w}_2 \triangleright \mathsf{senc}(n,k); \mathsf{w}_3 \triangleright k\}$ avec $a \in \Sigma_{\mathsf{fresh}}$ et $n, m, k \in \mathcal{N}$. Il existe plusieurs recettes pour déduire n dans ϕ_S, par exemple $R_1 = \mathsf{sdec}(\mathsf{w}_2, \mathsf{w}_3)$ et $R_2 = \mathsf{sdec}(\mathsf{w}_2, \mathsf{proj}_1^2(\langle \mathsf{w}_3, a\rangle^2))$. $R_2\downarrow = R_1$, $R_2\phi_S\downarrow = R_1\phi_S\downarrow$, et $\mathsf{hat}(R_2) = \mathsf{hat}(R_1)$ donc $\mu^3(R_2) > \mu^3(R_1)$ donne $\mu_{\phi_S}(R_2) > \mu_{\phi_S}(R_1)$, ce qui est cohérent avec le lemme 2.8.*

2.3.3 Substitution au premier ordre

Pour démontrer qu'il existe une attaque bien typée s'il existe une attaque, nous devrons expliciter un lien entre une trace tr d'attaque arbitraire (par rapport à Σ_0^-) et une trace tr_S bien typée (par rapport à Σ_0^+). Ce lien sera représenté par deux substitutions :

— Une substitution θ des constantes de Σ_{fresh} vers des Σ_0^--recettes, pour passer de tr_S à tr.

— Une substitution λ des constantes de Σ_{fresh} vers des messages, pour passer de ϕ_S à ϕ, où ϕ et ϕ_S sont respectivement les trames issues des exécutions de tr et tr_S.

L'objet de cette section est d'établir que λ est déterminée par θ et d'expliciter le passage de la trace tr_S à la trace tr. Le lemme suivant commence par démontrer un lien entre l'ordre d'apparition des constantes dans la trame et le *rang* d'une constante $c \in \Sigma_{\mathsf{fresh}}$, défini par $\mathsf{rank}(c) = \max_{\prec}\{\mathsf{w} \mid \mathsf{w} \in vars(c\theta)\})$, quand $\{\mathsf{w} \in vars(c\theta)\} \neq \emptyset$, et $\bot$ dans tous les autres cas.

Lemme 2.9. *Soient ϕ_S une Σ_0^+-trame et $\prec$ un ordre total sur $dom(\phi_S)$. Soit θ une substitution telle que $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, et pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans ϕ_S on ait $c \in dom(\theta)$ et $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$. De plus, on suppose que $\mathsf{w}' \prec \mathsf{w}$ pour chaque $\mathsf{w}' \in vars(c\theta)$ quand $c \in \Sigma_{\mathsf{fresh}}$ apparaît dans $\mathsf{w}\phi_S$.*

Chaque constante $c' \in \Sigma_{\mathsf{fresh}}$ qui apparaît dans $(c\theta)\phi_S{\downarrow}$ vérifie que $\mathsf{rank}(c') \prec \mathsf{rank}(c)$.

Démonstration. Soient ϕ_S une Σ_0^+-trame comme définie dans l'énoncé, et $c \in dom(\theta)$. Soit $c' \in \Sigma_{\mathsf{fresh}}$ une constante qui apparaît dans $(c\theta)\phi_S{\downarrow}$. Dans le cas $\mathsf{rank}(c) = \bot$, on a $vars(c\theta) = \emptyset$, et donc aucune constante de Σ_{fresh} n'apparaît dans $c\theta$, ce qui conclut ce cas.

Maintenant, soit $\mathsf{w}_i = \mathsf{rank}(c)$, on a $vars(c\theta) \subseteq \{\mathsf{w} \mid \mathsf{w} \preceq \mathsf{w}_i\}$, et par hypothèse sur θ, on a $\mathsf{rank}(c') \prec \mathsf{w}_i$ pour chaque constante $c' \in \Sigma_{\mathsf{fresh}}$ apparaissant dans $(c\theta)\phi_S$, et donc $\mathsf{rank}(c') \prec \mathsf{rank}(c)$. □

De plus, la connaissance de θ détermine λ, de sorte qu'à θ fixé il n'existe qu'une seule substitution λ correspondante, appelée *substitution au premier ordre associée à θ par tr_S*. Ce résultat est exprimé formellement dans le lemme suivant.

Lemme 2.10. *Soit ϕ_S une Σ_0^+-trame ainsi que $\prec$ un ordre total sur $dom(\phi_S)$. Soit θ une substitution telle que $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, et pour tout $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans ϕ_S, c appartienne à $dom(\theta)$ et $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$. De plus, on suppose que $\mathsf{w}' \prec \mathsf{w}$ pour chaque $\mathsf{w}' \in vars(c\theta)$ quand $c \in \Sigma_{\mathsf{fresh}}$ apparaît dans $\mathsf{w}\phi_S$. Considérons la substitution λ dont le domaine est $dom(\theta)$, et qui vérifie :*

$$c\lambda = (c\theta)(\phi_S\lambda){\downarrow} \text{ pour chaque } c \in dom(\lambda).$$

La substitution λ est bien définie. De plus, si $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message pour chaque $c \in dom(\theta)$, et $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message atomique quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$, alors $c\lambda$ est un Σ_0^--message pour chaque $c \in dom(\lambda)$, et $c\lambda$ est un Σ_0^--message atomique quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.

On dira que λ est la substitution au premier ordre associée à θ par ϕ_S.

Démonstration. Soit $dom(\phi_S) = \{\mathsf{w}_1, \ldots, \mathsf{w}_n\}$ tel que $\mathsf{w}_1 \prec \mathsf{w}_2 \prec \ldots \prec \mathsf{w}_n$. Le résultat se montre par récurrence sur $\mathsf{rank}(c)$ rangé dans l'ordre $\prec$.

Initialisation : $c \in dom(\theta)$ tel que $\mathsf{rank}(c) = \bot$. Dans ce cas, on a $vars(c\theta) = \emptyset$, et donc $c\lambda = (c\theta){\downarrow}$ est bien défini. De plus, en supposant que $(c\theta)\phi_S{\downarrow} = (c\theta){\downarrow}$ soit un Σ_0^+-message, on sait que c'est un Σ_0^--message, et donc $c\lambda$ est un Σ_0^--message qui est atomique dans le cas où $(c\theta)\phi_S{\downarrow}$ est atomique.

Hérédité : $c \in dom(\theta)$ tel que $\mathsf{rank}(c) = \mathsf{w}_i$. Soit λ_i la susbtitution qui coïncide avec λ sur son domaine $dom(\lambda_i) = \{c' \mid c' \in dom(\lambda) \text{ and } \mathsf{rank}(c') \prec \mathsf{w}_i\}$. La substitution λ_i est bien définie, et on a en fait $w_j\phi_S\lambda_i = w_j\phi_S\lambda$ pour chaque $\mathsf{w}_j \preceq \mathsf{w}_i$. Comme $\mathsf{rank}(c) = \mathsf{w}_i$, on sait que $vars(c\theta) \subseteq \{\mathsf{w}_1, \ldots, \mathsf{w}_i\}$, et donc $c\lambda = (c\theta)(\phi_S\lambda){\downarrow} = (c\theta)(\phi_S\lambda_i){\downarrow}$ est bien défini. De plus, en supposant que $(c\theta)\phi_S{\downarrow}$ soit un Σ_0^+-message, on sait que chaque constante $c' \in \Sigma_{\mathsf{fresh}}$ apparaissant dans $(c\theta)\phi_S{\downarrow}$ vérifie $\mathsf{rank}(c') \prec \mathsf{w}_i$ (d'après le lemme 2.9), et donc $c'\lambda$ est un Σ_0^--message, et c'est un atome si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. Donc $(c\theta)\phi_S{\downarrow}\lambda$ est un Σ_0^--message et est atomique si $c \in \Sigma_{\mathsf{atom}}^{\mathsf{fresh}}$. On a en fait $(c\theta)\phi_S{\downarrow}\lambda = (c\theta)(\phi_S\lambda){\downarrow}$, ce qui permet de conclure. □

Le lemme précédent a démontré que $c\lambda = (c\theta)(\phi_S\lambda){\downarrow}$ pour toutes les constantes $c \in \Sigma_{\mathsf{fresh}}$ qui apparaissent dans la trame ϕ_S. Cependant, nous voulons souvent raisonner directement sur les recettes, sans revenir chaque fois aux constantes de Σ_{fresh}, et nous aurons donc besoin du lemme suivant.

Lemme 2.11. *Soient ϕ_S une Σ_0^+-trame et $\prec$ un ordre total sur $dom(\phi_S)$. Soit θ une substitution telle que $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, et pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans ϕ_S, c appartienne à $dom(\theta)$ et $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$. De plus, supposons que $\mathsf{w}' \prec \mathsf{w}$ pour chaque $\mathsf{w}' \in vars(c\theta)$ quand $c \in \Sigma_{\mathsf{fresh}}$ apparaît dans $\mathsf{w}\phi_S$. Soit λ la substitution au premier ordre associée à θ par ϕ_S.*

On suppose que pour chaque $c \in dom(\lambda)$, $c\lambda$ est un Σ_0^--message. De plus, $c\lambda$ est un Σ_0^--message atomique quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.

Soit $R \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\theta) \uplus dom(\phi_S))$ tel que $R\phi_S\!\downarrow$ est un Σ_0^+-message. L'égalité suivante est vérifiée :

$$(R\theta)(\phi_S\lambda)\!\downarrow = (R\phi_S\!\downarrow)\lambda$$

Démonstration. Ce résultat se démontre par récurrence structurelle sur R.
Initialisation : $R \in \Sigma_0^- \uplus dom(\theta) \uplus dom(\phi_S)$. Dans le cas où $R = c_0 \in \Sigma_0^-$, on a $(c\theta)(\phi_S\lambda)\!\downarrow = c_0 = (c\phi_S\!\downarrow)\lambda)$. Dans le cas où $R = c \in dom(\theta)$, on a $(c\theta)(\phi_S\lambda)\!\downarrow = c\lambda = (c\phi_S\!\downarrow)\lambda$ d'après le lemme 2.10. Dans le cas où $R = \mathsf{w} \in dom(\phi_S)$, on a $(c\theta)(\phi_S\lambda)\!\downarrow = \mathsf{w}\phi_S\lambda\!\downarrow = (\mathsf{w}\phi_S\!\downarrow)\lambda$.
Hérédité : $R = \mathsf{f}(R_1, \ldots, R_k)$. Dans le cas où $\mathsf{f} \in \Sigma_{\mathsf{c}}$, on a :

$$(R\theta)(\phi_S\lambda)\!\downarrow = \mathsf{f}((R_1\theta)(\phi_S\lambda)\!\downarrow, \ldots (R_k\theta)(\phi_S\lambda)\!\downarrow) = \mathsf{f}(R_1\phi_S\!\downarrow\lambda, \ldots R_k\phi_S\!\downarrow\lambda) = R\phi_S\!\downarrow\lambda.$$

Dans le cas où $\mathsf{f} \in \Sigma_{\mathsf{d}}$, pour chaque $i \in \{i, \ldots, k\}$, $R_i\phi_S\!\downarrow$ est un Σ_0^+-message par hypothèse. Donc, par hypothèse de récurrence :

$$(R\theta)(\phi_S\lambda)\!\downarrow = \mathsf{f}((R_1\theta)(\phi_S\lambda)\!\downarrow, \ldots, (R_k\theta)(\phi_S\lambda)\!\downarrow)\!\downarrow = \mathsf{f}((R_1\phi_S)\!\downarrow\lambda, \ldots, (R_k\phi_S)\!\downarrow\lambda)\!\downarrow$$

Cependant, on a $R\phi_S\!\downarrow\lambda = \mathsf{f}(R_1\phi_S\!\downarrow, \ldots, R_k\phi_S\!\downarrow)\!\downarrow\lambda = \mathsf{f}(R_1\phi_S\!\downarrow, \ldots, R_k\phi_S\!\downarrow)\lambda\!\downarrow$ comme $c\lambda$ est un Σ_0^--message quand $c \in dom(\lambda)$ et $c\lambda$ est un Σ_0^--message atomique quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ d'après le lemme 2.10. Donc $R\phi_S\!\downarrow\lambda = \mathsf{f}(R_1\phi_S\!\downarrow\lambda, \ldots, R_k\phi_S\!\downarrow\lambda)\!\downarrow = (R\theta)(\phi_S\lambda)\!\downarrow$. □

Illustrons la notion de substitution au premier ordre sur l'exemple de référence.

Exemple 2.8. *Reprenons les traces tr' de l'exemple 1.16 et tr'' de l'exemple 2.2, le protocole P'_{OR} de l'exemple 1.16 et la Σ_0^--trame ϕ_1, définie à l'exemple 1.14, résultant de l'exécution de tr' dans P'_{OR}.*

Pour le système de types $(\mathcal{T}'_0, \delta')$, tr'' est un témoin bien typé tandis que tr' est mal typé. Appelons ϕ'' la Σ_0^+-trame telle que $(\mathsf{tr}'', \phi'') \in \mathsf{trace}_{\Sigma_0^+}(P'_{OR})$. ϕ'' ne diffère de ϕ_1 que sur la variable w_2 :

$$\mathsf{w}_2\phi'' = \langle m, a, b, c_{fwd}, t_B \rangle^5$$

Posons $\theta = \{c_{fwd} \triangleright \mathsf{proj}_4^4(\mathsf{w}_1)\}$. Posons $\phi_S = \phi''$ et $\mathsf{tr}_S = \mathsf{tr}''$, et l'ordre $\mathsf{w}_1 \prec \mathsf{w}_2 \prec \mathsf{w}_3 \prec \mathsf{w}_4$ sur $dom(\phi'')$. θ vérifie clairement les hypothèses du lemme 2.10, en particulier $vars(c_{fwd}\theta) = \{\mathsf{w}_1\}$ et $\mathsf{w}_1 \prec \mathsf{w}_2$. La trace $\mathsf{tr}''\theta$ mène à la trame ϕ_1. La substitution λ au premier ordre associée à θ par tr'' vérifie $dom(\lambda) = dom(\theta) = \{c_{fwd}\}$ et $c_{fwd}\lambda = (c_{fwd}\theta)(\phi''\lambda)\!\downarrow$. Comme $c_{fwd}\theta = \mathsf{proj}_4^4(\mathsf{w}_1)$, on a $c_{fwd}\lambda = \mathsf{proj}_4^4(\mathsf{w}_1(\phi''\lambda))\!\downarrow$. Comme $\mathsf{w}_1\phi''$ ne contient pas de variable de Σ_{fresh}, $c_{fwd}\lambda = \mathsf{proj}_4^4(\mathsf{w}_1\phi'')\!\downarrow$ et donc $c_{fwd}\lambda = t_A$. La substitution λ est donc $\lambda = \{c_{fwd} \triangleright t_A\}$. On peut remarquer que $\phi''\lambda = \phi_1$. Ainsi, les constantes de Σ_{fresh} servent à capturer les instanciations possibles et jouent le rôle de variables de θ et λ.

Dans la suite, nous montrerons que l'exemple précédent se généralise : les traces arbitraires peuvent être simulées à partir des traces bien typées en choisissant une substitution θ adaptée. Remarquons que $\mathsf{tr}''\theta \neq \mathsf{tr}$, mais que ces deux traces correspondent à une même trace concrète.

Pour compléter cette section, introduisons l'ordre induit par une trace tr_S, qui sera l'ordre utilisé pour invoquer les lemmes précédents.

Définition 2.6. *Étant donné une Σ_0^--configuration initiale $\mathcal{K}_0 = (\mathcal{P}_0; \phi_0; \emptyset; i)$ et une trace de cette configuration $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_0)$, l'ordre $\prec$* induit par tr_S *sur le sous-ensemble de $\Sigma_{\mathsf{fresh}} \uplus \mathcal{W}$ qui correspond aux éléments apparaissant dans tr_S et ϕ_S est défini par :*

$$u \prec v \;\Leftrightarrow\; \begin{cases} \textit{ou bien } u \in dom(\phi_0) \textit{ and } v \notin dom(\phi_0) \\ \textit{ou bien } u \textit{ apparaît dans } \mathsf{tr}_S \textit{ avant la première apparition de } v \textit{ dans } \mathsf{tr}_S. \end{cases}$$

2.4 Accessibilité

Dans cette section, nous allons démontrer le théorème 2.1, que nous rappelons ici.

Théorème 2.1. *Soit $\mathcal{K}_P$ une Σ_0^--configuration conforme à un système de types $(\mathcal{T}_0, \delta_0)$. Si $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^- alors il existe une exécution bien typée $\mathcal{K}_P \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}; \phi'; \sigma'; i)$ par rapport à Σ_0^+ telle que $\overline{\mathsf{tr}'} = \overline{\mathsf{tr}}$.*

Réciproquement, si $\mathcal{K}_P \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}; \phi'; \sigma'; i)$ est une exécution bien typée par rapport à Σ_0^+, alors il existe une exécution $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^- telle que $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}$.

Il s'agit donc de prouver que, pour des protocoles conformes à leur système de types, s'il existe une trace, il en existe une bien typée et qui passe par les mêmes points de contrôle. De cette manière, l'espace de recherche est restreint aux témoins bien typés. La section 2.4.1 démontre la correction, en introduisant un lemme plus général qui sera également utile pour l'équivalence. La section 2.4.2 présente le raisonnement général pour démontrer le théorème 2.1, et la section 2.4.3 expose les preuves laissées de côté dans la section précédente.

2.4.1 Correction

L'étape la plus facile consiste à démontrer la correction, c'est-à-dire que s'il existe une attaque utilisant les constantes de Σ_0^+, alors il existe une attaque utilisant seulement les constantes de Σ_0^-. Ce résultat nous autorisera à utiliser les constantes de Σ_{fresh} pour représenter des termes d'une trace d'attaque donnée.

Plutôt que de démontrer directement un tel résultat, nous introduisons un lemme plus fort qui sera utilisé dans la preuve de complétude dans toute sa généralité. Rappelons que l'ordre induit par tr_S a été introduit par la définition 2.6.

Lemme 2.12. *Soient $\mathcal{K}_Q = (\mathcal{Q}_0; \psi_0; \sigma_0; i_0)$ une Σ_0^--configuration, et $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$. Soit θ une substitution telle que $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, c appartienne à $dom(\theta)$, $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\psi_S))$, et $w' \preccurlyeq c$ pour chaque $w' \in vars(c\theta)$ (où $\preccurlyeq$ est l'ordre induit par tr_S). Supposons aussi que $(c\theta)\psi_S{\downarrow}$ est un Σ_0^+-message pour chaque $c \in dom(\theta)$ et est atomique dans le cas où $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. On a $(\mathsf{tr}_S\theta, \psi_S\lambda) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_Q)$ où λ est la substitution du premier-ordre associée θ par ψ_S.*

Démonstration. Comme $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$, l'exécution de tr_S est $\mathcal{K}_Q = (\mathcal{Q}_0; \psi_0; \sigma_0; i_0) \overset{\mathsf{tr}_S}{\Longrightarrow} (\mathcal{Q}_S; \psi_S; \sigma_S; i_S)$. Étant donnés θ et λ comme définis dans l'énoncé, il faut démontrer que : $\mathcal{K}_Q = (\mathcal{Q}_0; \psi_0; \sigma_0; i_0) \overset{\mathsf{tr}_S\theta}{\Longrightarrow} (\mathcal{Q}; \psi; \sigma; i)$ où $\mathcal{Q} = \mathcal{Q}_S$, $\phi = \psi_S\lambda$, $\sigma = \sigma_S\lambda$, et $i = i_S$. Ce résultat se montre par récurrence sur la longueur de la trace.

Initialisation : tr est vide. Soit $(\mathcal{Q}; \psi; \sigma; i) = \mathcal{K}_Q$. Comme $\mathcal{K}_Q$ est une Σ_0^--configuration, $\psi\lambda = \psi$, et $\sigma\lambda = \sigma$. Donc le résultat est évident.

Hérédité : $\mathsf{tr} = \mathsf{tr}^-.\alpha_S$. Dans ce cas :

$$\mathcal{K}_Q = (\mathcal{Q}_0; \psi_0; \sigma_0; i_0) \overset{\mathsf{tr}_S}{\Longrightarrow} (\mathcal{Q}_S^-; \psi_S^-; \sigma_S^-; i_S^-) \overset{\alpha_S}{\Longrightarrow} (\mathcal{Q}_S; \psi_S; \sigma_S; i_S)$$

D'après l'hypothèse de récurrence appliquée à tr^- :

$$\mathcal{K}_Q = (\mathcal{Q}_0; \psi_0; \sigma_0; i_0) \xrightarrow{\mathsf{tr}_S\theta} (\mathcal{Q}^-; \psi^-; \sigma^-; i^-)$$

avec $\mathcal{Q}^- = \mathcal{Q}_S^-$, $\psi^- = \psi_S^-\lambda$, $\sigma^- = \sigma_S^-\lambda$, et $i^- = i_S^-$. Distinguons trois cas suivant la valeur de α_S.

— *Cas* $\alpha_S = \mathsf{phase}\ j$ *pour un certain entier* j. Dans ce cas, $\mathcal{Q}_S = \mathcal{Q}_S^-$, $\psi_S = \psi_S^-$, $\sigma_S = \sigma_S^-$, et $i_S = j > i_S^-$. Soit $\mathcal{Q} = \mathcal{Q}^-$, $\psi = \psi^-$, $\sigma = \sigma^-$, $i = j$. Comme $i^- = i_S^- < j$:

$$(\mathcal{Q}^-; \psi^-; \sigma^-; i^-) \xrightarrow{\mathsf{phase}\ j} (\mathcal{Q}^-; \psi^-; \sigma^-; j) = (\mathcal{Q}; \psi; \sigma; i)$$

Les égalités $\mathcal{Q} = \mathcal{Q}_S$, $\psi = \psi_S\lambda$, $\sigma = \sigma_S\lambda$, et $i = i_S$ permettent de conclure.

— *Cas* $\alpha_S = \mathsf{in}(c, R)$ *pour une certaine* Σ_0^+*-recette* R. Dans ce cas, $\mathcal{Q}_S^- = \{\mathsf{in}(c, u).Q_c\} \uplus \mathcal{P}$, et $u\sigma_S^-$ et $R\psi_S^-\!\downarrow$ (qui est un terme clos) sont unifiables par une certaine substitution τ. De plus, $\sigma_S = \sigma_S^- \uplus \tau$. D'après le lemme 2.10, le lemme 2.11 s'applique, et donc :

$$(R\theta)\psi^-\!\downarrow = (R\theta)(\psi_S\lambda)\!\downarrow = (R\psi_S^-\!\downarrow)\lambda = [(u\sigma_S^-)\tau]\lambda = u(\sigma_S^- \uplus \tau)\lambda = (u(\sigma_S^-\lambda))(\tau\lambda).$$

Soit $\mathcal{Q} = \{Q_c\} \uplus \mathcal{P}$, $\psi = \psi^-$, $\sigma = \sigma^- \uplus \tau\lambda$, et $i = i^-$. Le pas suivant s'exécute :

$$(\{\mathsf{in}(c, u).Q_c\} \uplus \mathcal{P}; \psi^-; \sigma^-; i^-) \xrightarrow{\mathsf{in}(c, R\theta)} (\{Q_c\} \uplus \mathcal{P}; \psi^-; \sigma^- \uplus \tau\lambda; i^-) = (\mathcal{Q}; \psi; \sigma; i)$$

Les égalités $\mathcal{Q} = \mathcal{Q}_S$, $\psi = \psi_S\lambda$, $i = i_S$, et $\sigma = \sigma^- \uplus \tau\lambda = \sigma_S^-\lambda \uplus \tau\lambda = (\sigma_S^- \uplus \tau)\lambda = \sigma_S\lambda$ permettent de conclure.

— *Cas* $\alpha_S = \mathsf{out}(c, \mathsf{w})$. Dans ce cas, $\mathcal{Q}_S^- = \mathcal{Q}^- = \{\mathsf{out}(c, u).Q_c\} \uplus \mathcal{P}$, et $\psi_S = \psi_S^- \uplus \{\mathsf{w} \triangleright u\sigma_S^-\}$. Soient $\mathcal{Q} = \{Q_c\} \uplus \mathcal{P}$, $\psi = \psi^- \uplus \{\mathsf{w} \triangleright u\sigma^-\}$, $\sigma = \sigma^-$, et $i = i^-$. Montrons maintenant que $u\sigma^- = u\sigma_S^-\lambda$ est un Σ_0^--message. Remarquons que $u\sigma_S^-$ est un Σ_0^+-message. De plus, d'après le lemme 2.10, $c\lambda$ est un Σ_0^--message pour chaque $c \in dom(\lambda)$, et $c\lambda$ est atomique quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. Il en découle que $u\sigma^- = u\sigma_S^-\lambda$ est un Σ_0^--message. Donc :

$$(\{\mathsf{out}(c, u).Q_c\} \uplus \mathcal{P}; \psi^-; \sigma^-; i^-) \xrightarrow{\mathsf{out}(c, \mathsf{w})} (\{Q_c\} \uplus \mathcal{P}; \psi^- \uplus \{\mathsf{w} \triangleright u\sigma^-\}; \sigma^-; i^-) = (\mathcal{Q}; \psi; \sigma; i).$$

Les égalités $\mathcal{Q} = \mathcal{Q}_S$, $\sigma = \sigma_S\lambda$, $i = i_S$, et $\psi = \psi_S\lambda$ (puisque $\sigma_S^-\lambda = \sigma^-$, par hypothèse de récurrence), nous donnent le résultat.

Remarquons que la trace $\mathsf{tr}_S\theta$ ne contient que des Σ_0^--recettes : les constantes de Σ_{fresh} apparaissant dans tr_S ont été remplacées grâce à θ, ce qui permet de conclure. □

La correction découle directement du lemme précédent.

Lemme 2.13. *Soit* $\mathcal{K}_\mathcal{P}$ *une* Σ_0^-*-configuration conforme au système de type* $(\mathcal{T}_0, \delta_0)$. *Si* $\mathcal{K}_\mathcal{P} \xRightarrow{\mathsf{tr}'} (\mathcal{P}; \phi'; \sigma'; i)$ *est une exécution par rapport à* Σ_0^+*, alors il existe une exécution* $\mathcal{K}_\mathcal{P} \xRightarrow{\mathsf{tr}'} (\mathcal{P}; \phi; \sigma; i)$ *par rapport à* Σ_0^- *telle que* $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}$.

Démonstration. Le résultat est une conséquence directe du lemme 2.12 en considérant la substitution θ définie par $\theta(c) = a \in \Sigma_0^-$ pour chaque $c \in \Sigma_{\mathsf{fresh}}$. □

2.4.2 Complétude

Cette section expose le raisonnement qui prouve la complétude, et se termine par une preuve du théorème 2.1. Les autres démonstrations, plus techniques, sont laissées de côté, et sont reprises dans la section 2.4.3.

La première étape consiste à démontrer que l'exécution n'introduit pas de nouveaux termes chiffrés. Dans la suite, le lemme suivant nous permet de connaître les sous-termes chiffrés à tout moment de l'exécution directement à partir des sous-termes chiffrés du processus de départ. Comme l'hypothèse de conformité du protocole au système de type porte sur les sous-termes chiffrés du processus de départ, il sera possible de déduire qu'au cours d'une exécution, le type des sous-termes chiffrés reste contrôlé.

Lemme 2.14. *Soient $\mathcal{K}_0 = (\mathcal{P}_0; \phi_0; \emptyset; 0)$ et $\mathcal{K} = (\mathcal{P}; \phi; \sigma; i)$ deux Σ_0-configurations telles que $\mathcal{K}_0 \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}$ pour une certaine trace tr par rapport à Σ_0.*

1. *On a $ESt(\mathcal{K}\sigma) \subseteq ESt(\mathcal{K}_0\sigma)$.*
2. *De plus, si σ est un unificateur le plus général entre des paires de termes de $ESt(\mathcal{K}_0)$, alors $ESt(\mathcal{K}\sigma) \subseteq ESt(\mathcal{K}_0)\sigma$.*

Le premier item du lemme découle directement des définitions : les processus de $\mathcal{K}$ sont inclus dans ceux de $\mathcal{K}_0$ et les termes émis et conservés dans la trame ϕ de $\mathcal{K}$ apparaissent d'abord dans $\mathcal{K}_0$. Par ailleurs, aucun sous-terme chiffré n'est créé par unification, ce qui permet d'obtenir le second item. Cette intuition de preuve est formellement explicitée dans la section 2.4.3.

Nous avons maintenant tous les éléments pour exprimer la transformation qui permet de passer d'une trace quelconque à une trace bien typée. Si nous reprenons les notations de l'exemple 2.8, il s'agit de calculer la trace tr'' à partir de tr', tout en déterminant une substitution θ adaptée. Cette transformation est la même que celle qui sera utilisée pour l'équivalence, et la propriété suivante sera donc appelée au cours de la démonstration du théorème 2.2. Cette propriété possède un intérêt par elle-même, car elle ne dépend pas du système de type et ne nécessite donc pas d'hypothèse de conformité. L'espace de recherche est réduit aux exécutions qui unifient exclusivement des sous-termes chiffrés. Ensuite, l'hypothèse de conformité permet de formuler une restriction en termes de système de types.

L'énoncé sera suivi d'une intuition de la preuve. La démonstration complète se trouve dans la section suivante.

Proposition 2.1. *Soient $\mathcal{K}_P = (\mathcal{P}_0; \phi_0; \emptyset; i_0)$ une Σ_0^--configuration, et $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$ avec la substitution sous-jacente σ. Alors il existe $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ avec la substitution sous-jacente σ_S de domaine $dom(\sigma_S) = dom(\sigma)$ telle que $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ et θ qui vérifient :*

- *$\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P)$ tel que $u\sigma = v\sigma\}$.*
- *ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constantes de Σ_{fresh} telles que $x\rho \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.*
- *$dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, c appartient à $dom(\theta)$, $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \preccurlyeq c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\preccurlyeq$ est l'ordre induit par tr_S).*
- *pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c est un atome si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.*
- *λ est la substitution au premier ordre associée à θ par ϕ_S.*
- *$\phi = \phi_S\lambda$, $\sigma = \sigma_S\lambda$, et $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$.*

L'égalité $\sigma_S = mgu(\Gamma)\rho$ garantit que (tr_S, ϕ_S) est bien typée. En effet, comme Γ est un ensemble de sous-termes chiffrés unifiables du protocole, $mgu(\Gamma)$ est bien typé (par hypothèse sur

le protocole). Donc la substitution σ_S est bien typée. Le théorème 2.1 découlera facilement de la proposition 2.1.

La preuve de cette proposition est l'étape fondamentale pour prouver l'existence d'un témoin bien typé. Donnons-en une intuition, dans laquelle nous omettrons provisoirement la distinction entre Σ_0^+ et Σ_0^-. Nous procédons par récurrence sur une trace d'exécution

$$\mathcal{K}_P \overset{\mathsf{tr}^-}{\Longrightarrow}{}^* (\mathcal{P}; \phi^-; \sigma^-; i^-) \xrightarrow{\alpha} (\mathcal{P}; \phi; \sigma; i)$$

Pa récurrence, il existe ϕ_S^-, et σ_S^- tels que

$$\mathcal{K}_P \overset{\mathsf{tr}_S^-}{\longrightarrow}{}^* (\mathcal{P}; \phi_S^-; \sigma_S^-; i^-)$$

avec les substitutions λ^-, et θ^- comme spécifiées dans l'énoncé. Considérons la transition α.

— Le cas de la phase est direct : $(\mathsf{tr}_S^-, \phi_S^-)$ peut être étendue facilement.

— Le cas d'une émission $\alpha = \mathsf{out}(c, \mathsf{w})$ avec un processus correspondant $\mathsf{out}(c, u).P$, est relativement simple. Il faut simplement garantir que $u\sigma_S^-$ est un message, ce qui découle du fait que $u\sigma^-$ est un message et $\sigma^- = \sigma_S^-\lambda^-$.

— Le cas difficile est celui de la réception : $\alpha = \mathsf{in}(c, R)$, avec le processus correspondant $\mathsf{in}(c, u).P$. L'égalité $R\phi^-\!\downarrow = u\sigma$ est vérifiée. Il s'agit de chercher une recette R_S telle que

$$(R_S\theta^-)\phi^-\!\downarrow = u\sigma$$

Une telle recette R_S existe puisque R conviendrait. Considérons donc la recette R_S minimale (par rapport à notre mesure), qui satisfait cette propriété.

Étape 1 La minimalité de R_S permet de montrer que $R_S\phi_S\!\downarrow$ est un message.

Étape 2 Montrons que $R_S = C[R_1, \ldots, R_n]$ où C est un contexte de constructeurs et chaque $R_i\phi_S\!\downarrow$ est un sous-terme chiffré. En effet, si la racine de $R_i\phi_S\!\downarrow$ était un symbole transparent, il serait possible de le pousser dans le contexte C et d'obtenir une recette R_S plus petite (pour la mesure $\mu^2_{\phi_S}$). En d'autres termes, R_S est une recette simple.

Étape 3 R_S ne satisfait toujours pas toutes les propriétés nécessaires pour passer $\mathsf{in}(c, u)$. En particulier, la relation suivante n'est pas garantie.

$$R_S\phi_S\!\downarrow = u\sigma_S$$

Il suffit alors de construire $\overline{R_S}$ à partir de R_S en coupant les parties qui sortent de $u\sigma_S$. D'après le lemme 2.11, $(R_S\phi_S\!\downarrow)\lambda = u\sigma_S\lambda$. Considérons une feuille c de R_S qui sort de $u\sigma_S$. Si $R_S\phi_S\!\downarrow \neq u\sigma_S$, alors deux cas se présentent.

— soit c apparaît dans le contexte C, et alors il est possible de couper toute la partie du contexte C qui sort de $u\sigma_S$ au-dessus de c.

— soit c appartient à l'un des R_i. Alors $R_i\phi_S\!\downarrow$ est un sous-terme chiffré de ϕ_S qui est égal à un sous-terme chiffré de $u\sigma_S$. D'après le lemme 2.14, ce sont des sous-termes de Γ et donc, par application de l'unificateur le plus général, ils sont égaux (et donc il est inutile de couper).

La proposition précédente permet de démontrer le théorème 2.1.

Théorème 2.1. *Soit $\mathcal{K}_P$ une Σ_0^--configuration conforme à un système de types $(\mathcal{T}_0, \delta_0)$. Si $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^- alors il existe une exécution bien typée $\mathcal{K}_P \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}; \phi'; \sigma'; i)$ par rapport à Σ_0^+ telle que $\overline{\mathsf{tr}'} = \overline{\mathsf{tr}}$.*

Réciproquement, si $\mathcal{K}_P \overset{\mathsf{tr}'}{\Longrightarrow} (\mathcal{P}; \phi'; \sigma'; i)$ est une exécution bien typée par rapport à Σ_0^+, alors il existe une exécution $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^- telle que $\overline{\mathsf{tr}} = \overline{\mathsf{tr}'}$.

Démonstration. La réciproque découle du lemme 2.13. Il suffit donc de prouver le sens direct. Soit $\mathcal{K}_P$ une Σ_0^--configuration conforme à $(\mathcal{T}_0, \delta_0)$. Supposons que $\mathcal{K}_P \overset{\mathsf{tr}}{\Longrightarrow} (\mathcal{P}; \phi; \sigma; i)$ par rapport à Σ_0^-. D'après la proposition 2.1, il existe $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ avec la substitution sous-jacente σ_S vérifiant $dom(\sigma_S) = dom(\sigma)$ tel que $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ et θ qui vérifient :

— $\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P) \text{ tels que } u\sigma = v\sigma\}$.

— ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constantes de Σ_{fresh} tel que $x\rho \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.

— $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, c appartient à $dom(\theta)$, $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \preccurlyeq c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\preccurlyeq$ est l'ordre induit par tr_S).

— pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.

— λ est la substitution au premier ordre associée à θ par ϕ_S.

— $\phi = \phi_S\lambda$, $\sigma = \sigma_S\lambda$, et $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$.

Comme $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$, $\overline{\mathsf{tr}_S} = \overline{\mathsf{tr}}$. Comme il existe suffisamment de constantes de chaque type, supposons sans perte de généralité que ρ est bien typé. Comme $\mathcal{K}_P$ est conforme au système de type, les sous-termes chiffrés $ESt(\mathcal{K}_P)$ qui sont unifiables ont le même type. Donc, par définition d'un système de type, il en découle que $mgu(\Gamma)$ est bien typée, et donc $\sigma_S = mgu(\Gamma)\rho$ est bien typée. Donc la trace tr_S est bien typée, ce qui conclut la preuve. □

2.4.3 Démonstrations

Démontrons d'abord le lemme 2.14.

Lemme 2.14. *Soient $\mathcal{K}_0 = (\mathcal{P}_0; \phi_0; \emptyset; 0)$ et $\mathcal{K} = (\mathcal{P}; \phi; \sigma; i)$ deux Σ_0-configurations telles que $\mathcal{K}_0 \overset{\mathsf{tr}}{\Longrightarrow} \mathcal{K}$ pour une certaine trace tr par rapport à Σ_0.*

1. *On a $ESt(\mathcal{K}\sigma) \subseteq ESt(\mathcal{K}_0\sigma)$.*
2. *De plus, si σ est un unificateur le plus général entre des paires de termes de $ESt(\mathcal{K}_0)$, alors $ESt(\mathcal{K}\sigma) \subseteq ESt(\mathcal{K}_0)\sigma$.*

Démonstration. Montrons que $ESt(\mathcal{K}\sigma') \subseteq ESt(\mathcal{K}_0\sigma')$ pour chaque σ' qui coïncide avec σ sur $dom(\sigma)$ par récurrence sur la longueur de l'exécution $\mathcal{K}_0 \xrightarrow{\mathsf{tr}} \mathcal{K}$.

Initialisation : tr *est vide.* Dans ce cas, le resultat est évident.

Hérédité : $\mathsf{tr} = \mathsf{tr}_0 \cdot \alpha$. L'exécution de tr se décompose en $\mathcal{K}_0 \xrightarrow{\mathsf{tr}_0} \mathcal{K}_1 \xrightarrow{\alpha} \mathcal{K}$ où $\mathcal{K}_1 = (\mathcal{P}_1; \phi_1; \sigma_1; i_1)$, et $ESt(\mathcal{K}_1\sigma_1') \subseteq ESt(\mathcal{K}_0\sigma_1')$ pour chaque σ_1' qui coïncide avec σ_1 sur $dom(\sigma_1)$. Distinguons trois cas selon α :

— Dans le cas où α est une τ-action ou une action $\mathsf{phase}\ \ i$, les égalités $\sigma = \sigma_1$ et $St(\mathcal{K}) = St(\mathcal{K}_1)$ sont vérifiées. Soit σ' une substitution qui coïncide avec $\sigma = \sigma_1$ sur $dom(\sigma)$. Les égalités précédentes donnent $ESt(\mathcal{K}\sigma') = ESt(\mathcal{K}_1\sigma') \subseteq ESt(\mathcal{K}_0\sigma')$.

— Dans le cas où α est une réception (in), alors $\mathcal{P}_1 = \mathsf{in}(c, u).P \uplus \mathcal{P}_1'$ et $\mathcal{P} = P \uplus \mathcal{P}_1'$ donc $St(\mathcal{P}) \subseteq St(\mathcal{P}_1)$. De plus, $\sigma = \sigma_1 \uplus \tau$ et $dom(\tau) = vars(u\sigma_1)$. Il en découle que $ESt(\tau) \subseteq ESt(u\sigma_1\tau) = ESt(u\sigma)$. De plus $\phi = \phi_1$, et donc $ESt(\phi) = ESt(\phi_1)$. Donc soit σ' une substitution qui coïncide avec σ sur $dom(\sigma)$. L'inclusion $ESt(\mathcal{K}\sigma') \subseteq ESt(\mathcal{K}_1\sigma')$ est vérifiée. Remarquons que σ' coïncide avec σ_1 sur $dom(\sigma_1)$, donc d'après l'hypothèse de récurrence, $ESt(\mathcal{K}_1\sigma') \subseteq ESt(\mathcal{K}_0\sigma')$.

— Dans le cas où α est une émission (out), la trame ϕ vaut $\phi_1 \uplus \{\mathsf{w} \triangleright u\sigma_1\}$ pour un certain $u \in St(\mathcal{K}_1)$ et $\sigma = \sigma_1$. Donc $ESt(\phi) \subseteq ESt(\mathcal{K}_1\sigma_1)$. Soit σ' une substitution qui coïncide avec $\sigma = \sigma_1$ sur $dom(\sigma)$. Les égalités précédentes donnent $ESt(\mathcal{K}\sigma') \subseteq ESt(\mathcal{K}_1\sigma') \subseteq ESt(\mathcal{K}_0\sigma')$.

Ce qui conclut la preuve du premier item.

Pour démontrer l'item 2, considérons σ l'unificateur le plus général entre des paires de termes de $ESt(\mathcal{K}_0)$, et montrons que $ESt(\mathcal{K}_0\sigma) \subseteq ESt(\mathcal{K}_0)\sigma$. Remarquons que cette inclusion associée à l'item 1 permet de conclure. Soit $t \in ESt(\mathcal{K}_0\sigma)$ tel que $t \notin ESt(\mathcal{K}_0)\sigma$. Donc $t \in ESt(img(\sigma))$. Soit $\overline{\sigma} = \sigma\delta$ où δ remplace toute occurrence de t dans $img(\sigma)$ par une variable fraîche x. Les deux items suivants sont vérifiés :

— $u\overline{\sigma} = v\overline{\sigma}$ pour chaque $u = v \in \Gamma$. En effet, $u\sigma = v\sigma$, et donc $(u\sigma)\delta = (v\sigma)\delta$. Comme $t \notin ESt(\mathcal{K}_0)\sigma$, et $u, v \in ESt(\mathcal{K}_0)$, il en découle que $(u\sigma)\delta = u(\sigma\delta) = u\overline{\sigma}$ et de même $(v\sigma)\delta = v(\sigma\delta) = v\overline{\sigma}$.

— $\overline{\sigma}$ est strictement plus générale que σ. En effet, $\sigma = \overline{\sigma}\tau$ avec $\tau = \{x \mapsto t\}$ et t est un terme chiffré, donc t n'est pas une variable.

Ces résultats contredisent $\sigma = mgu(\Gamma)$, ce qui conclut la preuve. □

Passons à la démonstration formelle de la propriété 2.1.

Proposition 2.1. *Soient $\mathcal{K}_P = (\mathcal{P}_0; \phi_0; \emptyset; i_0)$ une Σ_0^--configuration, et $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$ avec la substitution sous-jacente σ. Alors il existe $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ avec la substitution sous-jacente σ_S de domaine $dom(\sigma_S) = dom(\sigma)$ telle que $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ et θ qui vérifient :*

— $\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P)$ *tel que* $u\sigma = v\sigma\}$.

— *ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constantes de Σ_{fresh} telles que $x\rho \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.*

— *$dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, c appartient à $dom(\theta)$, $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \preccurlyeq c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\preccurlyeq$ est l'ordre induit par tr_S).*

— *pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$.*

— *λ est la substitution au premier ordre associée à θ par ϕ_S.*

— *$\phi = \phi_S\lambda$, $\sigma = \sigma_S\lambda$, et $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$.*

Démonstration. L'exécution de la trace tr est $\mathcal{K}_P \xRightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma; i)$. Soit Γ l'ensemble :

$$\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P) \text{ tel que } u\sigma = v\sigma\}$$

Soit ρ un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constantes de Σ_{fresh} tel que $x\rho \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique. Soit σ_S tel que $dom(\sigma_S) = dom(\sigma)$ et $\sigma_S = mgu(\Gamma)\rho$. Comme $mgu(\Gamma)$ est une substitution plus générale que σ, $\sigma = mgu(\Gamma)\lambda_0$ pour un certain λ_0. Soit $\lambda = \rho^{-1}\lambda_0$. L'égalité $\sigma = \sigma_S\lambda$ est vérifiée.

Montrons par induction sur la longueur du préfixe $\mathcal{K}_P \xRightarrow{\mathsf{tr}^+} (\mathcal{P}^+; \phi^+; \sigma^+; i^+)$ de cette trace d'exécution qu'il existe $(\mathsf{tr}_S^+, \phi_S^+) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ avec $\sigma_S^+ = \sigma_S|_{dom(\sigma^+)}$, ainsi que deux substitutions θ^+ et λ^+ telles que :

— $dom(\theta^+) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S^+, c appartient à $dom(\theta^+)$, $c\theta^+ \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S^+))$, et $\mathsf{w}' \preccurlyeq^+ c$ pour chaque $\mathsf{w}' \in vars(c\theta^+)$ (où $\preccurlyeq^+$ est l'ordre induit par tr_S^+).

— pour chaque $c \in dom(\theta^+)$, $(c\theta^+)\phi_S^+{\downarrow}$ est un Σ_0^+-message et c'est un atome quand $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$.

— λ^+ est la substitution au premier ordre associée à θ^+ par ϕ_S^+.

— $\phi^+ = \phi_S^+\lambda^+$, $\sigma^+ = \sigma_S^+\lambda^+$, et $(\mathsf{tr}_S^+\theta^+)\phi^+{\downarrow} = \mathsf{tr}^+\phi^+{\downarrow}$.

Initialisation : tr^+ est vide. Dans ce cas, $dom(\sigma^+) = \emptyset$, et $\phi^+ = \phi_0$. Soit $\mathsf{tr}_S^+ = \epsilon$, $\phi_S^+ = \phi_0$, et $\sigma_S^+ = \emptyset$. En choisissant les substitutions θ^+ et λ^+ telles que $dom(\theta^+) = dom(\lambda^+) = \emptyset$, le résultat est clairement vérifié.

Hérédité : $\mathsf{tr}^+ = \mathsf{tr}^-.\alpha$. Dans ce cas :

$$\mathcal{K}_P = (\mathcal{P}_0; \phi_0; \emptyset; i_0) \xRightarrow{\mathsf{tr}^-} (\mathcal{P}^-; \phi^-; \sigma^-; i^-) \xRightarrow{\alpha} (\mathcal{P}^+; \phi^+; \sigma^+; i^+).$$

Par hypothèse de récurrence appliquée à tr^-, il existe $(\mathsf{tr}_S^-, \phi_S^-) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ avec la substitution sous-jacente $\sigma_S^- = \sigma_S|_{dom(\sigma^-)}$, ainsi que deux substitutions θ^- et λ^- telles que :

- $dom(\theta^-) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S^-, c appartient à $dom(\theta^-)$, $c\theta^- \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S^-))$, et $\mathsf{w}' \preccurlyeq^- c$ pour chaque $\mathsf{w}' \in vars(c\theta^-)$ (où $\preccurlyeq^-$ est l'ordre induit par tr_S^-).
- pour chaque $c \in dom(\theta^-)$, $(c\theta^-)\phi_S^-\!\downarrow$ est un Σ_0^+-message et c'est un atome quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.
- λ^- est la substitution au premier ordre associée à θ^- par ϕ_S^-.
- $\phi^- = \phi_S^-\lambda^-$, $\sigma^- = \sigma_S^-\lambda^-$, et $(\mathsf{tr}_S^-\theta^-)\phi^-\!\downarrow = \mathsf{tr}^-\phi^-\!\downarrow$.

Soit $c \in \Sigma_{\mathsf{fresh}}$ et $\mathsf{w} \in dom(\phi_S^-)$ tel que c apparaît dans $\mathsf{w}\phi_S$. Donc, comme c n'apparaît pas dans la Σ_0^--configuration $\mathcal{K}_P$, c a été introduit par une réception (in) avant l'émission de w. Donc $c \preccurlyeq^- \mathsf{w}$. Soit $\mathsf{w}' \in vars(c\theta^-)$. La définition de $\preccurlyeq^-$ implique $\mathsf{w}' \preccurlyeq^- c$. Donc $\mathsf{w}' \preccurlyeq^- \mathsf{w}$ et l'ordre induit par $\preccurlyeq^-$ sur $dom(\phi_S^-)$ satisfait la condition des lemmes 2.10 et 2.11. Trois cas se présentent suivant α.

Cas où $\alpha = \mathsf{phase}\ j$ pour un certain entier j. Dans ce cas, $\mathcal{P}^+ = \mathcal{P}^-$, $\phi^+ = \phi^-$, $\sigma^+ = \sigma^-$, et $i^+ = j > i^-$. Soit $\mathcal{P}_S^+ = \mathcal{P}_S^-$, $\phi_S^+ = \phi_S^-$, $\sigma_S^+ = \sigma_S^-$, et $i_S^+ = j$. Comme $i_S^- = i^- < j$:

$$(\mathcal{P}_S^-; \phi_S^-; \sigma_S^-; i_S^-) \xRightarrow{\mathsf{phase}\ j} (\mathcal{P}_S^+; \phi_S^+; \sigma_S^+; j) = (\mathcal{P}_S^+; \phi_S^+; \sigma_S^+; i_S^+).$$

En considérant $\theta^+ = \theta^-$, $\lambda^+ = \lambda^-$, et $\preccurlyeq^+ = \preccurlyeq^-$, it est facile de montrer que toutes les conditions sont satisfaites.

Cas où $\alpha = \mathsf{out}(c, \mathsf{w}_0)$. Dans ce cas, $\mathcal{P}^- = \{\mathsf{out}(c,u).P_c\} \uplus \mathcal{Q}$ pour certains u, P_c, et $\mathcal{Q}$, $\mathcal{P}^+ = \{P_c\} \uplus \mathcal{Q}$, $\phi^+ = \phi^- \uplus \{\mathsf{w}_0 \triangleright u\sigma^-\}$, $\sigma^+ = \sigma^-$, et $i^+ = i^-$. Soit $\mathcal{P}_S^+ = \{P_c\} \uplus \mathcal{Q}$, $\phi_S^+ = \phi_S^- \uplus \{\mathsf{w}_0 \triangleright u\sigma_S^-\}$, $\sigma_S^+ = \sigma_S^-$, et $i_S^+ = i_S^-$. Définissons $\theta^+ = \theta^-$ et $\lambda^+ = \lambda^-$.

Premièrement, montrons que $u\sigma_S^-$ est un Σ_0^+-message. D'après l'hypothèse de récurrence, $u\sigma^- = u\sigma_S^-\lambda^-$, et par hypothèse $u\sigma^-$ est un Σ_0^--message. Donc, afin de conclure, il suffit de montrer que $c\lambda^- \in \Sigma_0^-$ implique $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans $u\sigma_S^-$. Soit $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ apparaissant dans $u\sigma_S^-$. Comme $u\sigma^- = u\sigma_S^-\lambda^-$, c appartient à $dom(\lambda^-)$. Supposons que $c\lambda^- \in \Sigma_0^-$ et soit $x \in dom(\sigma) \setminus dom(mgu(\Gamma))$ l'unique variable telle que $\rho(x) = c$. Comme $xmgu(\Gamma) = x$, il en découle que $c\lambda^- = ((xmgu(\Gamma))\rho)\lambda^- = x\sigma_S^-\lambda^- = x\sigma^-$. D'où $x\sigma \in \Sigma_0^-$, et donc $\rho(x) = c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ par définition du renommage ρ.

Il a donc été montré que $u\sigma_S^-$ est un Σ_0^+-message, et il en découle :

$$(\mathcal{P}_S^-; \phi_S^-; \sigma_S^-; i_S^-) \xRightarrow{\mathsf{out}(c,\mathsf{w}_0)} (\mathcal{P}_S^+; \phi_S^- \uplus \{\mathsf{w} \triangleright u\sigma_S^-\}; \sigma_S^-; i_S^-) = (\mathcal{P}_S^+; \phi_S^+; \sigma_S^+; i_S^+).$$

En considérant $\theta^+ = \theta^-$, $\lambda^+ = \lambda^-$, et $\preccurlyeq^+$ l'extension de $\preccurlyeq^-$ qui vérifie $u \preccurlyeq^+ \mathsf{w}_0$ pour chaque $u \in \mathcal{W} \uplus \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S^- ou dans $dom(\phi_0)$, il est facile de montrer que toutes les conditions sont satisfaites.

Cas où $\alpha = \mathsf{in}(c, R)$. Le protocole $\mathcal{P}^-$ vérifie $\mathcal{P}^- = \{\mathsf{in}(c,u).P_c\} \cup \mathcal{Q}$ pour un certain $u \in \mathcal{T}_0(\Sigma_c, \Sigma_0^- \uplus \mathcal{N})$ et $R\phi^-\!\downarrow = (u\sigma^-)\tau$ pour un certain τ avec $dom(\tau) = vars(u\sigma^-)$. De plus, $\sigma^+ = \sigma^- \uplus \tau$, $(u\sigma^-)\tau = u\sigma^+$, $\phi^+ = \phi^-$, et $\mathcal{P}^+ = \{P_c\} \cup \mathcal{Q}$. Considérons la recette R_S minimale par rapport à

$\mu_{\phi_S^-}$ telle que $(R_S\theta^-)\phi^-\!\downarrow = (u\sigma^-)\tau = u\sigma^+$. Une telle recette R_S existe puisque $R_S = R$ est un candidat (qui n'est pas nécessairement minimal). Supposons sans perte de généralité que R_S utilise seulement les constantes de Σ_{fresh} introduites par tr_S^-, et qui sont donc dans $dom(\theta^-)$.

Étape 1 : Prouvons que $R_S\phi_S^-\!\downarrow$ est un Σ_0^+-message. Supposons que $R_S\phi_S^-\!\downarrow$ n'est pas un Σ_0^+-message. Prenons le plus petit sous-terme R' de R_S tel que $R'\phi_S^-\!\downarrow$ n'est pas un Σ_0^+-message. Soit p tel que $R_S|_p = R'$. Comme $R'\phi_S^-\!\downarrow$ n'est pas un Σ_0^+-message, nécessairement $R' \notin \Sigma_0^+ \uplus dom(\phi_S^-)$. Donc, $R' = \mathsf{f}(R_1, \ldots, R_k)$ pour un certain $\mathsf{f} \in \Sigma$. De plus, par minimalité de R', $R_i\phi_S^-\!\downarrow$ est un Σ_0^+-message pour chaque $1 \leqslant i \leqslant k$. Prouvons le fait suivant.

Fait. Si $R_i\phi_S^-\!\downarrow \in \Sigma_{\mathsf{fresh}}$ pour un certain $i \in \{1, \ldots, k\}$, alors R_S n'est pas minimale.

Démonstration. Supposons $R_i\phi_S^-\!\downarrow = c \in \Sigma_{\mathsf{fresh}}$ pour un certain $i \in \{1, \ldots, k\}$. Remarquons que cette hypothèse implique que c apparaît dans ϕ_S^-, et donc c apparaît dans tr_S, et ainsi $c \in dom(\theta^-)$. Considérons $R_i' = c\theta^-$. D'après le lemme 2.11, $(R_i\theta^-)\phi^-\!\downarrow = R_i\phi_S^-\!\downarrow\lambda^-$. Les égalités $R_i\phi_S^-\!\downarrow\lambda^- = c\lambda^- = (c\theta^-)(\phi_S^-\lambda^-)\!\downarrow$ proviennent de ce que λ^- est la substitution au premier ordre associée à θ^- par ϕ_S^-. Donc $(R_i\theta^-)\phi^-\!\downarrow = (c\theta^-)\phi^-\!\downarrow = (R_i'\theta^-)\phi'\!\downarrow$. Soit $\overline{R_S} = R_S[\mathsf{f}(R_1, \ldots, R_i', \ldots, R_k)]_p$. L'égalité $(\overline{R_S}\theta^-)\phi^-\!\downarrow = (R_S\theta^-)\phi^-\!\downarrow$ est vérifiée. De plus $\mu^1_{\phi_S^-}(R_i') < \mu^1_{\phi_S^-}(R_i)$ puisque $R_i'\phi_S^-\!\downarrow = (c\theta^-)\phi_S^-\!\downarrow$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$, et $R_i\phi_S^-\!\downarrow = c$. Donc, d'après le lemme 2.7, il en découle que $\mu^1_{\phi_S^-}(\overline{R_S}) < \mu^1_{\phi_S^-}(R_S)$. Donc la recette R_S n'est pas minimale, ce qui prouve le fait.

Distinguons maintenant deux cas suivant que $\mathsf{f} \in \Sigma_{\mathsf{c}}$ ou $\mathsf{f} \in \Sigma_{\mathsf{d}}$.

Cas où $\mathsf{f} \in \Sigma_{\mathsf{c}}$. $R'\phi_S^-\!\downarrow$ ne respecte pas les contours ou les sortes, puisque ce n'est pas un Σ_0^+-message. S'il ne respecte pas les sortes, alors pour l'une de ses sous-recettes R_i, $R_i\phi_S^-\!\downarrow$ n'est pas un atome (il respecte les sortes par minimalité de R') alors qu'il se trouve en position de clé. En particulier, $(R_i\theta^-)\phi^-\!\downarrow = R_i\phi_S^-\!\downarrow\lambda^-$ est un atome. Donc $R_i\phi_S^-\!\downarrow \in \Sigma_{\mathsf{fresh}}$, ce qui contredit la minimalité de R_S d'après le fait qui a été démontré. Il en découle que $R'\phi_S^-\!\downarrow$ respecte les sortes.

Supposons que $R'\phi_S^-\!\downarrow$ ne respecte pas les contours, considérons le contour de f, $\mathsf{sh}_{\mathsf{f}} = \mathsf{f}(s_1, \ldots, s_k)$ pour certains $s_1, \ldots, s_k$. Comme $R'\phi_S^-\!\downarrow$ ne respecte pas les contours et $\mathsf{root}(R'\phi_S^-\!\downarrow) = \mathsf{f}$, il existe un j tel que $R_j\phi_S^-\!\downarrow$ n'est pas une instance de s_j. En particulier, s_j n'est pas une variable et $s_j = \mathsf{sh}_{\mathsf{g}}$ pour un certain symbole g (d'après la compatibilité des contours). $R_j\phi_S^-\!\downarrow$ est un Σ_0^+-message et donc $(R_j\theta^-)\phi^-\!\downarrow = (R_j\phi_S^-\!\downarrow)\lambda^-$ (d'après le lemme 2.11) est une instance de s_j puisque $(R\theta^-)\phi^-\!\downarrow$ est un Σ_0^--message. En appliquant le fait qui a été démontré, $R_j\phi_S^-\!\downarrow \notin \Sigma_{\mathsf{fresh}}$, donc $\mathsf{root}(R_j\phi_S^-\!\downarrow) = \mathsf{g}$, et comme $R_j\phi_S^-\!\downarrow$ est un Σ_0^+-message, c'est une instance de s_j, ce qui conduit à une contradiction.

Cas où $\mathsf{f} = \mathsf{des} \in \Sigma_{\mathsf{d}}$. Dans ce cas, $R' = \mathsf{des}(R_1, \ldots, R_k)$. Soit $\ell_{\mathsf{des}} = \mathsf{des}(t_1, t_2, \ldots, t_k)$. Distinguons deux sous-cas.

D'abord, supposons qu'il existe un $i \in \{1, \ldots, k\}$ tel que $R_i\phi_S^-\!\downarrow$ ne s'unifie pas avec t_i. Comme $R_i\phi_S^-\!\downarrow$ est un Σ_0^+-message, il respecte les contours. Donc, le seul moyen de ne pas s'unifier avec le terme linéaire t_i tandis que $R_i\phi_S^-\!\downarrow\lambda^- = (R_i\theta^-)\phi^-\!\downarrow$ (d'après le lemme 2.11) s'unifie, est que $R_i\phi_S^-\!\downarrow = c$ pour un certain $c \in \Sigma_{\mathsf{fresh}}$. D'après le fait qui a été démontré, c'est une contradiction.

Ensuite, supposons que $R_i\phi_S^-\!\downarrow$ s'unifie avec t_i pour chaque $i \in \{1, \ldots k\}$. Dans ce cas, $\ell_{\mathsf{des}} = \mathsf{des}(t_1, t_2, \ldots, t_k)$ est un terme non linéaire. Notons x la variable non-linéaire apparaissant dans ℓ_{des}. Soit $I_0 = \{1 \leqslant i \leqslant k \mid x \text{ apparaît dans } t_i\}$. Remarquons que $1 \in I_0$. Pour chaque $i \in I_0$, notons p_i la position dans t_i telle que $t_i|_{p_i} = x$.

Considérons le terme suivant :

$$t = \mathsf{des}(R_1\phi_S^-\!\downarrow\lambda^-, \ldots, R_k\phi_S^-\!\downarrow\lambda^-)$$

Puisque $R'\phi_S^-\downarrow\lambda^-$ est un Σ_0^--message, t s'unifie avec $\mathsf{des}(t_1,\ldots,t_k)$. Donc, il existe un Σ_0^--message atomique a tel que $t|_{i.p_i} = a$ pour chaque $i \in I_0$. $R'\phi_S^-\downarrow$ n'est pas un Σ_0^+-message tandis que les $R_i\phi_S^-\downarrow$ sont des Σ_0^+-messages. Donc, $t_S = \mathsf{des}(R_1\phi_S^-\downarrow,\ldots,R_k\phi_S^-\downarrow)$ ne s'unifie pas avec $\mathsf{des}(t_1,\ldots,t_k)$. Il en découle que pour chaque $i \in I_0$, ou bien $t_S|_{i.p_i} = a$, ou bien $t_S|_{i.p_i} = c$ pour un certain $c \in \Sigma_{\mathsf{fresh}}$ tel que $c\lambda^- = a$. Distinguons deux cas suivant que $R_1\phi_S^-\downarrow|_{p_1} = c'$ pour un certain $c' \in \Sigma_{\mathsf{fresh}}$ ou non.

Dans un premier temps, supposons que $R_1\phi_S^-\downarrow|_{p_1} = c'$ pour un certain $c' \in \Sigma_{\mathsf{fresh}}$. Soit ν_{des} la substitution telle que $x\nu_{\mathsf{des}} = c'$ et $y\nu_{\mathsf{des}} = c_{min}$ pour chacune des autres variables $y \in vars(\ell_{\mathsf{des}})$. Soit $\overline{R'} = \mathsf{des}(R_1, t_2\nu_{\mathsf{des}},\ldots,t_k\nu_{\mathsf{des}})$. En fait, $\overline{R'}\phi_S^-\downarrow$ est un Σ_0^+-message, et d'après le lemme 2.11, $(\overline{R'}\theta^-)\phi^-\downarrow = \overline{R'}\phi_S^-\downarrow\lambda^-$. Comme $\overline{R'}\phi_S^-\downarrow$ est un Σ_0^+-message, $\mu^1_{\phi_S^-}(\overline{R'}) < \{\mathsf{des}\}$, et la comparaison $\mu^1_{\phi_S^-}(\overline{R'}) < \mu^1_{\phi_S^-}(R')$ en découle. Il en découle également que $(\overline{R'}\theta^-)\phi^-\downarrow$ est un Σ_0^--message, et $(\overline{R'}\theta^-)\phi^-\downarrow = (R'\theta^-)\phi^-\downarrow$ car les Σ_0^+-recettes $\overline{R'}$ et R' coïncident sur leur premier argument.

Dans un second temps, supposons que $R_1\phi_S^-\downarrow|_{p_1} \notin \Sigma_{\mathsf{fresh}}$, c'est-à-dire $R_1\phi_S^-\downarrow|_{p_1} = a$. Il existe $i_0 \in I_0$ tel que $R_{i_0}\phi_S^-\downarrow|_{p_{i_0}} = c'$ pour un certain $c' \in \Sigma_{\mathsf{fresh}}$ (sinon t_S se réduirait), et $c'\lambda^- = a$. Soit ν_{des} la substitution telle que $x\nu_{\mathsf{des}} = c'\theta^-$ et $y\nu_{\mathsf{des}} = c_{min}$ pour chacune des autres variables $y \in vars(\ell_{\mathsf{des}})$. Soit $\overline{R'} = \mathsf{des}(R_1, t_2\nu_{\mathsf{des}},\ldots,t_k\nu_{\mathsf{des}})$. De $\mu^1_{\phi_S^-}(c'\theta^-) < \mu^1_{\phi_S^-}(c')$ découle la comparaison $\mu^1_{\phi_S^-}(R_1) \uplus \mu^1_{\phi_S^-}(t_2\nu_{\mathsf{des}}) \uplus \ldots \uplus \mu^1_{\phi_S^-}(t_k\nu_{\mathsf{des}}) < \mu^1_{\phi_S^-}(R_1) \uplus \ldots \uplus \mu^1_{\phi_S^-}(R_k)$. Donc, d'après le lemme 2.6, il en découle :

$$\begin{array}{rcl}
\mu^1_{\phi_S^-}(\overline{R'}) & = & \mathsf{Multi}(\overline{R'}\phi_S^-\downarrow) \\
& \leqslant & \mathsf{Multi}(\mathsf{des}(R_1\phi_S^-\downarrow, t_2\nu_{\mathsf{des}}\phi_S^-\downarrow,\ldots,t_k\nu_{\mathsf{des}}\phi_S^-\downarrow)) \\
& = & \{\mathsf{des}\} \uplus \mu^1_{\phi_S^-}(R_1) \uplus \mu^1_{\phi_S^-}(t_2\nu_{\mathsf{des}}) \uplus \cdots \uplus \mu^1_{\phi_S^-}(t_k\nu_{\mathsf{des}}) \\
& < & \{\mathsf{des}\} \uplus \mu^1_{\phi_S^-}(R_1) \uplus \cdots \uplus \mu^1_{\phi_S^-}(R_k) \\
& = & \mu^1_{\phi_S^-}(R')
\end{array}$$

Comme $R_1\phi_S^-\downarrow$ est un Σ_0^+-message :

$$\begin{array}{l}
\mathsf{des}((R_1\theta^-)\phi^-\downarrow, ((t_2\nu_{\mathsf{des}})\theta^-)\phi^-\downarrow,\ldots,((t_k\nu_{\mathsf{des}})\theta^-)\phi^-\downarrow) \\
\quad = \mathsf{des}((R_1\phi_S^-)\downarrow\lambda^-, ((t_2\nu_{\mathsf{des}})\theta^-)\phi^-\downarrow,\ldots,((t_k\nu_{\mathsf{des}})\theta^-)\phi^-\downarrow)
\end{array}$$

Un tel terme s'unifie avec ℓ_{des} puisque $R_1\phi_S^-\downarrow\lambda^-|_{p_1} = R_1\phi_S^-\downarrow|_{p_1}\lambda^- = a\lambda^- = a$, et pour chaque $i \in I_0$:

$$((t_i\nu_{\mathsf{des}})\theta^-)\phi^-\downarrow|_{p_i} = (t_i\nu_{\mathsf{des}})\phi^-\downarrow|_{p_i} = ((t_i\nu_{\mathsf{des}})|_{p_i})\phi^-\downarrow = (c'\theta^-)\phi^-\downarrow = c'\lambda^- = a.$$

Donc, $(\overline{R'}\theta^-)\phi^-\downarrow$ est un Σ_0^--message, et $(\overline{R}'\theta^-)\phi^-\downarrow = (R'\theta^-)\phi^-\downarrow$ car les Σ_0^+-recettes $\overline{R'}$ et R' coïncident sur leur premier argument.

Dans chaque cas, $(\overline{R'}\theta^-)\phi^-\downarrow = (R'\theta^-)\phi^-\downarrow$, et $\mu^1_{\phi_S^-}(\overline{R'}) < \mu^1_{\phi_S^-}(R')$. Comme $R'\phi_S^-\downarrow$ n'est pas un Σ_0^+-message, le lemme 2.7 s'applique, et donc $\mu^1_{\phi_S^-}(R_S[\overline{R'}]_p) < \mu^1_{\phi_S^-}(R_S[R']_p) = \mu^1_{\phi_S^-}(R_S)$, ce qui contredit la minimalité de R_S.

Étape 2 : Prouvons que R_S est une Σ_0^+-recette simple, en forme normale pour $\twoheadrightarrow$, et de la forme $C[R_1,\ldots,R_n]$ où pour chaque $i \in \{1,\ldots,n\}$, $R_i\phi_S^-\downarrow$ est soit un terme chiffré, un nom de $\mathcal{N}$, ou une constante de Σ_0^+.

Supposons que R_S *ne* soit *pas* en forme normale pour $\twoheadrightarrow$. D'après le lemme 2.8, $\mu^1_{\phi_S^-}(R_S\downarrow) < \mu^1_{\phi_S^-}(R_S)$. De plus, comme $R_S\theta^- \twoheadrightarrow^* R_S\downarrow\theta^-$, le lemme 2.2 s'applique : $(R_S\downarrow\theta^-)\phi^-\downarrow = (R_S\theta^-)\phi^-\downarrow$,

ce qui contredit la minimalité de R_S. Donc, R_S est en forme normale pour $\twoheadrightarrow$, et ainsi R_S est une Σ_0^+-recette simple d'après le lemme 2.3. Donc, $R_S = C[R_1, \ldots, R_n]$ où chaque R_i est une recette de sous-terme avec $1 \leqslant i \leqslant n$. S'il existe $i_0 \in \{1, \ldots, n\}$ tel que $\mathsf{root}(R_{i_0}\phi_S^-\downarrow) = \mathsf{f} \in \Sigma_\mathsf{c}$ où f est un symbole transparent, alors $R_{i_0}\phi_S^-\downarrow = \mathsf{f}(C_1^\mathsf{f}[R_{i_0}], \ldots, C_k^\mathsf{f}[R_{i_0}])\phi_S^-\downarrow$. Considérons le contexte $\overline{C}$ tel que

$$\overline{C}[R_1, \ldots, R_n] = C[R_1, \ldots, \mathsf{f}(C_1^\mathsf{f}[R_{i_0}], \ldots, C_k^\mathsf{f}[R_{i_0}]), \ldots, R_n]$$

La recette $R'_S = \overline{C}[R_1, \ldots, R_n]$ est une Σ_0^+-recette telle que $R'_S\phi_S^-\downarrow = R_S\phi_S^-\downarrow$, et donc $\mu^1_{\phi_S}(R_S) = \mu^1_{\phi_S}(R'_S)$, et $(R'_S\theta^-)\phi^-\downarrow = R'_S\phi_S^-\downarrow\lambda^- = R_S\phi_S^-\downarrow\lambda^- = (R_S\theta^-)\phi^-\downarrow$. La comparaison $\mu^2_{\phi_S}(R'_S) < \mu^2_{\phi_S}(R_S)$ contredit la minimalité de R_S. Donc, il en découle que chaque $R_i\phi_S\downarrow$ (avec $1 \leqslant i \leqslant n$) est un terme chiffré, une constante de Σ_0^+, ou un nom de $\mathcal{N}$.

Étape 3. Soit $\mathcal{P}_S^+ = \mathcal{P}^+$, $\phi_S^+ = \phi_S^-$, $\sigma_S^+ = \sigma_S|_{dom(\sigma^+)}$, et $i_S^+ = i_S^-$. Montrons qu'il existe une Σ_0^+-recette $\overline{R_S}$ telle que :

$$(\mathcal{P}_S^-; \phi_S^-; \sigma_S^-; i_S^-) \xrightarrow{\mathsf{in}(c, \overline{R_S})} (\mathcal{P}_S^+; \phi_S^+; \sigma_S^+; i_S^+)$$

ainsi que deux substitutions θ^+ et λ^+ qui satisfont toutes les conditions demandées.

Si $R_S\phi_S^-\downarrow = u\sigma_S^+$, alors soit $\overline{R_S} = R_S$, $\theta^+ = \theta^-$, et $\lambda^+ = \lambda^-$. Pour conclure, il reste à montrer que toutes les conditions sont satisfaites. En particulier, il faut montrer que *(i)* $(\overline{R_S}\theta^+)\phi^+\downarrow = R\phi^+\downarrow$, et *(ii)* $\sigma^+ = \sigma_S^+\lambda^+$.

(i) Les égalités $(\overline{R_S}\theta^+)\phi^+\downarrow = (R_S\theta^-)\phi^-\downarrow = u\sigma^+ = R\phi^-\downarrow = R\phi^+\downarrow$ sont vérifiées.

(ii) Soit $Z = vars(u\sigma^-) = dom(\tau)$. Le terme $\overline{R_S}\phi_S^-\downarrow = u\sigma_S^+ = u(\sigma_S^- \uplus \sigma_S|_Z)$ est un Σ_0^+-message, et $(R_S\theta^-)\phi^-\downarrow = R\phi^-\downarrow = u\sigma^+ = u(\sigma^- \uplus \sigma|_Z)$. D'après le lemme 2.11, $\overline{R_S}\phi_S^-\downarrow\lambda^- = (R_S\theta^-)\phi^-\downarrow$, et donc $u(\sigma_S^-\lambda^- \uplus \sigma_S|_Z\lambda^-) = u(\sigma^- \uplus \sigma|_Z)$. Il en découle que $\sigma_S|_Z\lambda^- = \sigma|_Z$, et :

$$\sigma_S^+\lambda^+ = (\sigma_S^- \uplus \sigma_S|_Z)\lambda^- = \sigma_S^-\lambda^- \uplus \sigma_S|_Z\lambda^- = \sigma^- \uplus \sigma|_Z = \sigma^- \uplus \tau = \sigma^+.$$

Donc, à partir de maintenant, supposons que $R_S\phi_S^-\downarrow \neq u\sigma_S^+$. Soit A l'ensemble des constantes fraîches qui apparaissent avant cette étape de l'exécution, c'est-à-dire $A = St(img(\sigma_S^+)) \cap \Sigma_{\mathsf{fresh}}$. Définissons $\lambda^+ = \lambda|_A$. De $\sigma = \sigma_S\lambda$, découle l'égalité $\sigma^+ = \sigma_S^+\lambda^+$. Comme $R_S\phi_S^-\downarrow$ est un Σ_0^+-message, le lemme 2.11 donne $R_S\phi_S^-\downarrow\lambda^+ = R_S\phi_S^-\downarrow\lambda^- = (R_S\theta^-)\phi^-\downarrow = R\phi^-\downarrow = u\sigma^+ = (u\sigma_S^+)\lambda^+$. Soit $t = R_S\phi_S\downarrow$ et $v = u\sigma_S^+$. De plus, $t \neq v$ et $t\lambda^+ = v\lambda^+$.

Comme $t \neq v$, il existe une position p de t et v telle que $\mathsf{root}(t|_p) \neq \mathsf{root}(v|_p)$. Soit p une position de t et v tel que $\mathsf{root}(t|_p) \neq \mathsf{root}(v|_p)$. Comme $t\lambda^+ = v\lambda^+$ et $dom(\lambda^+) \subseteq \Sigma_{\mathsf{fresh}}$, ou bien $t|_p \in \Sigma_{\mathsf{fresh}}$ ou bien $v|_p \in \Sigma_{\mathsf{fresh}}$.

Supposons d'abord qu'il existe une telle position p qui tombe en dehors du contexte C. Plus précisément, $p = p'.p''$ (avec p' un préfixe strict de p) et $C[R_1, \ldots, R_n]|_{p'} = R_{i_0}$ pour un certain entier $i_0 \in \{1, \ldots, n\}$. Donc, comme R_{i_0} est une recette de sous-terme, we know that $t|_{p'} = R_{i_0}\phi_S^-\downarrow$ est un sous-terme chiffré de ϕ_S^- D'après le lemme 2.14, et en notant $\mathcal{K}_S^- = (\mathcal{P}_S^-; \phi_S^-; \sigma_S^-; i_S^-)$:

— $t|_{p'} \in ESt(\phi_S^-) \subseteq ESt(\mathcal{K}_S^-\sigma_S^-) \subseteq ESt(\mathcal{K}_P\sigma_S^-) = ESt(\mathcal{K}_P(mgu(\Gamma)|_{dom(\sigma_S^-)}\rho)) \subseteq ESt(\mathcal{K}_P)\sigma_S^-$.

— $v|_{p'} = u\sigma_S^+|_{p'} \in ESt(\mathcal{K}_P\sigma_S^+) \subseteq ESt(\mathcal{K}_P(mgu(\Gamma)|_{dom(\sigma_S^+)})\rho) \subseteq ESt(\mathcal{K}_P)\sigma_S^+$.

Il en découle qu'il existe $t', v' \in ESt(\mathcal{K}_0)$ tels que $t'\sigma_S^- = t|_{p'}$, et $v'\sigma_S^+ = v|_{p'}$. Puisque $t|_{p'}$ est un Σ_0^+-message, donc l'égalité $t'\sigma_S^+ = t|_{p'}$ est vérifiée. L'égalité $t|_{p'}\lambda^+ = v|_{p'}\lambda^+$ provient de $t\lambda^+ = v\lambda^+$, et il en découle $(t'\sigma_S^+)\lambda^+ = (v'\sigma_S^+)\lambda^+$. Les égalités $(t'\sigma_S^+)\lambda^+ = t'(\sigma_S^+\lambda^+)$ et $(v'\sigma_S^+)\lambda^+ = v'(\sigma_S^+\lambda^+)$ sont vérifiées. Comme $\sigma^+ = \sigma_S^+\lambda^+$, il en découle que $t'\sigma^+ = v'\sigma^+$, et donc $t'\sigma = v'\sigma$. Par définition

de σ_S, $t'\sigma_S = v'\sigma_S$. Puisque $t'\sigma_S^- = t|_{p'}$, et comme $t|_{p'}$ est clos, l'égalité $t'\sigma_S = t|_{p'}$ est vérifiée. De même, $v'\sigma_S^+ = v|_{p'}$, et comme $v|_{p'}$ est clos, il en découle que $v'\sigma_S = v|_{p'}$. Donc $t|_{p'} = v|_{p'}$, ce qui contredit la supposition que t et v sont différents sous la position p'.

Il a été montré que pour chaque position p définie dans t et v telle que $\mathsf{root}(t|_p) \neq \mathsf{root}(v|_p)$, $t|_p$ ou $v|_p$ est dans Σ_{fresh}, et p est une position de C. Si $t|_p = c \in \Sigma_{\mathsf{fresh}}$, soit $R'_S = R_S[c\theta^-]_p$, alors la comparaison $\mu^1_{\phi_S^-}(R'_S) < \mu^1_{\phi_S^-}(R_S)$ découle de $\mu^1_{\phi_S^-}(c\theta^-) < \mu^1_{\phi_S^-}(R_S|_p)$ (remarquons que $R_S|_p\phi_S^-\downarrow = c$ tandis que $(c\theta^-)\phi_S^-\downarrow$ est un Σ_0^+-message et c'est un atome quand $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$). De plus, $(R'_S\theta^-)\phi^-\downarrow = (R_S\theta^-)\phi^-\downarrow$, ce qui contradit la minimalité de R_S. Donc $t|_p \notin \Sigma_{\mathsf{fresh}}$, et donc $v|_p = c$ pour un certain $c \in \Sigma_{\mathsf{fresh}}$.

Soient $p_1, \ldots, p_m$ les positions telles que pour chaque $i \in \{1, \ldots, m\}$, p_i est une position de t et v, et $\mathsf{root}(t|_{p_i}) \neq \mathsf{root}(v|_{p_i})$. Pour chaque $i \in \{1, \ldots, m\}$, p_i est une position de C telle que $t|_{p_i} \notin \Sigma_{\mathsf{fresh}}$, et $v|_{p_i} \in \Sigma_{\mathsf{fresh}}$. Notons c^i_{fresh} la constante de Σ_{fresh} telle que $v|_{p_i} = c^i_{\mathsf{fresh}}$. Remarquons qu'il peut se produire que $c^i_{\mathsf{fresh}} = c^j_{\mathsf{fresh}}$ pour certains $i \neq j$. Soit $\overline{C}$ le contexte obtenu à partir de C en plaçant c^1_{fresh} à la position p_1, c^2_{fresh} à la position p_2, ... Soit $\overline{R_S} = \overline{C}[R_1, \ldots, R_n]$. Par construction, $\overline{R_S}\phi_S^-\downarrow = u\sigma_S^+$. Soit $R_{p_i} = (C[R_1, \ldots, R_n]|_{p_i})\theta^-$ pour chaque $i \in \{1, \ldots, m\}$. D'après le lemme 2.11, $R_{p_i}\phi^-\downarrow = R_{p_i}\phi_S^-\downarrow\lambda^-$ pour chaque $i \in \{1, \ldots, m\}$. Comme pour chaque $i \in \{1, \ldots, m\}$, l'égalité $R_{p_i}\phi_S^-\downarrow\lambda^- = u\sigma_S^+\lambda^+|_{p_i}$ est vérifiée, il en découle que $R_{p_i}\phi_S^-\downarrow\lambda^- = c^i_{\mathsf{fresh}}\lambda^+$ pour chaque $i \in \{1, \ldots, m\}$. Remarquons que si $c^i_{\mathsf{fresh}} = c^j_{\mathsf{fresh}}$, alors $R_{p_i}\phi^-\downarrow = R_{p_j}\phi^-\downarrow$. Soit $\theta^+ = \theta^- \uplus \{c^i_{\mathsf{fresh}} \mapsto R_{p_i} \mid c^i_{\mathsf{fresh}} \notin dom(\theta^-)\}$. Dans le cas où $c^i_{\mathsf{fresh}} = c^j_{\mathsf{fresh}}$ pour certains $i \neq j$, il suffit de choisir arbitrairement entre R_{p_i} et R_{p_j}.

Il reste à montrer que toutes les conditions sont satisfaites. En particulier :

(i) $\sigma_S^+ = \sigma_S^- \uplus \tau_S$ pour une certaine substitution τ_S telle que $dom(\tau_S) = vars(u\sigma_S^-)$ et $\overline{R_S}\phi_S^-\downarrow = (u\sigma_S^-)\tau_S$.

(ii) pour chaque $c \in dom(\theta^+) \setminus dom(\theta^-)$, $(c\theta^+)\phi_S^-\downarrow$ est un Σ_0^+-message, et c'est un atome quand $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$.

(iii) pour chaque $c \in dom(\theta^+) \setminus dom(\theta^-)$, $c\lambda^+ = (c\theta^+)(\phi_S^-\lambda^-)\downarrow$.

(iv) $(\overline{R_S}\theta^+)\phi^+\downarrow = R\phi^+\downarrow$.

Démontrons chaque item.

(i) Soit $\tau_S = \sigma_S|_{dom(\tau)}$. Le domaine de τ_S vérifie $dom(\tau_S) = dom(\tau) = vars(u\sigma^-) = vars(u\sigma_S^-)$. Il a été montré que $\overline{R_S}\phi_S^-\downarrow = u\sigma_S^+$, et donc $\overline{R_S}\phi_S^-\downarrow = u(\sigma_S^- \uplus \tau_S) = (u\sigma_S^-)\tau_S$.

(ii) Soit $c \in dom(\theta^+) \setminus dom(\theta^-)$. La constante c vaut c^i_{fresh} pour un certain $i \in \{1, \ldots, m\}$, et $(c^i_{\mathsf{fresh}}\theta^+)\phi_S^-\downarrow = R_{p_i}\phi_S^-\downarrow$ est un Σ_0^+-message. Supposons que $c^i_{\mathsf{fresh}} \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$. Dans ce cas, il existe une variable $x_i \in dom(\rho)$ telle que $x_i\rho = c^i_{\mathsf{fresh}}$. Comme $x_i\rho$ est atomique, $x_i\sigma$ est atomique (par définition de ρ), et $x_i\sigma = x_i\sigma_S\lambda$. De $\sigma_S = mgu(\Gamma)\rho$, se déduit $x_i\sigma = x_i mgu(\Gamma)\rho\lambda$. Comme $x_i \in dom(\rho)$, $x_i \notin dom(mgu(\Gamma))$ par définition de ρ. Donc $x_i\sigma = x_i\rho\lambda = c^i_{\mathsf{fresh}}\lambda = c^i_{\mathsf{fresh}}\lambda^+$. Puisque $x_i\sigma$ est atomique, il en découle que $c^i_{\mathsf{fresh}}\lambda^+$ est atomique. Donc, comme il a été montré que $R_{p_i}\phi_S^-\downarrow\lambda^- = c^i_{\mathsf{fresh}}\lambda^+$, $R_{p_i}\phi_S^-\downarrow\lambda^-$ est atomique ce qui implique ou bien que $R_{p_i}\phi_S^-\downarrow$ est atomique, ou bien que $c \in \Sigma_{\mathsf{fresh}}$ avec $c \in dom(\lambda^-)$. Supposons donc $c \in \Sigma_{\mathsf{fresh}}$ avec $c \in dom(\lambda^-)$. Il existe une variable $x \in dom(\rho)$ telle que $x\rho = c$, et la variable x vérifie $x\sigma = x\sigma_S\lambda = x mgu(\Gamma)\rho\lambda$. Comme $x \in dom(\rho)$, $x \notin dom(mgu(\Gamma))$ et $x\sigma = x\rho\lambda = c\lambda = c\lambda^-$. Comme $x\sigma$ est atomique, $x\rho$ est atomique (par définition de ρ), donc c est atomique, et ainsi $R_{p_i}\phi_S^-\downarrow$ est atomique.

(iii) Soit $c \in dom(\theta^+) \setminus dom(\theta^-)$. La variable c vaut c^i_{fresh} pour un certain $i \in \{1, \ldots, m\}$, et $c^i_{\mathsf{fresh}}\lambda^+ = R_{p_i}\phi^-\downarrow = R_{p_i}(\phi_S^-\lambda^-)\downarrow = (c^i_{\mathsf{fresh}}\theta^+)(\phi_S^-\lambda^-)\downarrow$.

(iv) $(\overline{R_S}\theta^+)\phi^+\downarrow = (R_S\theta^-)\phi^-\downarrow = R\phi^-\downarrow = R\phi^+\downarrow$.

Ce qui conclut le cas où α est une réception, et prouve la propriété. □

2.5 Équivalence

Dans cette section, nous démontrons le théorème 2.2, rappelé ci-dessous.

Théorème 2.2. *Soient $\mathcal{K}_P$ une Σ_0^--configuration conforme à $(\mathcal{T}_0, \delta_0)$ et $\mathcal{K}_Q$ une Σ_0^--configuration déterministe. On a $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^- si, et seulement si, il existe un témoin bien typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ de cette non-inclusion.*

Nous allons donc montrer que, si $\mathcal{K}_P$ est un protocole conforme à son système de types, et si $\mathcal{K}_Q$ est un protocole déterministe, il suffit de considérer les traces bien typées de $\mathcal{K}_P$ pour savoir si ces deux configurations sont en inclusion approximative de trace. Cette section est structurée comme la précédente, dont elle reprend certains résultats. Cependant, la preuve de correction de la section 2.5.1 est plus technique que celle de l'accessibilité. La section 2.5.2 définit une autre notion d'équivalence statique, qui simplifiera le raisonnement menant à la preuve de complétude, exposé dans la section 2.5.3. La section 2.5.4 contient la preuve formelle de la proposition principale de la section 2.5.3.

2.5.1 Correction

Le rôle de cette section est de démontrer la correction, c'est-à-dire la réciproque du théorème 2.2. Comme dans le cas de l'accessibilité, développé dans la section 2.4.1, le principe est de remplacer les constantes de Σ_{fresh} par des constantes de Σ_0^-. Cependant, dans ce cas précis, il n'est pas toujours possible de créer des égalités supplémentaires en projetant toutes les constantes sur la même, par exemple si la non-inclusion provient d'un test d'égalité valable dans $\mathcal{K}_P$ mais pas dans $\mathcal{K}_Q$. Il faudra donc utiliser un renommage bijectif. De même, le remplacement de constantes non atomiques de $\Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$ par des constantes atomiques permet parfois plus d'exécutions, ce qui peut faire disparaître des témoins. C'est pourquoi les constantes de sorte bitstring seront parfois remplacées par un message t_0 de sorte bitstring, qui existe par hypothèse sur la théorie.

Lemme 2.15. *Soient $\mathcal{K}_P$ une Σ_0^--configuration conforme à $(\mathcal{T}_0, \delta_0)$ et $\mathcal{K}_Q$ une Σ_0^--configuration déterministe. Si $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^+ alors $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^-.*

Démonstration. Par hypothèse sur la théorie, il existe un message t_0 de sorte bitstring, puisque la théorie contient au moins une règle non linéaire $\mathsf{des}(t_1, \ldots, t_n) \to r_{\mathsf{des}}$ avec une position de clé dans t_1, ce qui signifie que $t_1\tau$ est un message de sorte bitstring, si τ est la substitution de domaine $dom(\tau) = vars(t_1)$ et d'image $img(\tau) = \{a\}$ avec $a \in \Sigma_0^-$.

Considérons un témoin minimal (en longueur) de non-inclusion, c'est-à-dire une trace $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ telle que :

1. ou bien $(\mathsf{tr}_S, \psi_S) \notin \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$ pour chaque ψ_S ;
2. ou bien $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$ mais $\phi_S \not\sqsubseteq_s \psi_S$.

Considérons deux substitutions :

— θ, un renommage injectif de Σ_{fresh} vers des constantes de Σ_0^- qui n'ont pas déjà été utilisées au cours de l'exécution étudiée.

— θ', un renommage bijectif depuis $\Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ vers Σ_0^-, et depuis $\Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$ vers t_0. Remarquons que θ' préserve l'atomicité, c'est-à-dire $\theta'(c)$ est un atome si, et seulement si, $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.

θ et θ' permettent de renommer les constantes de Σ_{fresh} pour les remplacer par soit par des constantes, soit par un terme composé t_0.

Cas 1 : $\mathsf{tr}_S = \mathsf{tr}_S^-.\alpha_S$ ne s'exécute pas dans $\mathcal{K}_Q$. Dans ce cas :

— $\mathcal{K}_P \xrightarrow{\mathsf{tr}_S^-} (\mathcal{P}_S^-; \phi_S^-; \sigma_P^-; i_P^-) \xrightarrow{\alpha_S} (\mathcal{P}_S; \phi_S; \sigma_P; i_P)$;

— $\mathcal{K}_Q \xrightarrow{\mathsf{tr}_S^-} (\mathcal{Q}_S^-; \psi_S^-; \sigma_Q^-; i_Q^-)$; et

— $\phi_S^- \sqsubseteq_s \psi_S^-$.

D'après le lemme 2.12, $(\mathsf{tr}_S^-\theta, \psi_S^-\theta) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_Q)$. Plus précisément, $\mathcal{K}_Q \xrightarrow{\mathsf{tr}_S^-\theta} (\mathcal{Q}_S^-; \psi_S^-\theta; \sigma_Q^-\theta; i_Q^-)$. Il en va de même pour la substitution θ'. Remarquons que la substitution au premier ordre associée à θ (resp. θ') par ψ_S^- est θ (resp. θ') elle-même. Donc, pour conclure, il suffit de montrer que $\alpha_S\theta$ ne peut pas être exécutée à partir de $(\mathcal{Q}_S^-; \psi_S^-\theta; \sigma_Q^-\theta; i_Q^-)$, ou alors que $\alpha_S\theta'$ ne peut pas être exécutée à partir de $(\mathcal{Q}_S^-; \psi_S^-\theta'; \sigma_Q^-\theta'; i_Q^-)$ Considérons trois cas suivant l'action α_S.

1. $\alpha_S = \mathsf{phase}\ \ i$. Comme $i > i_P^-$ et $i \leqslant i_Q^-$, $\mathsf{phase}\ \ i$ ne peut toujours pas être exécutée à partir de $(\mathcal{Q}_S^-; \psi_S^-\theta; \sigma_Q^-\theta; i_Q^-)$. Donc $\mathsf{tr}_S\theta$ est un témoin de non-inclusion par rapport à Σ_0^-.
2. $\alpha_S = \mathsf{out}(c, \mathsf{w})$ avec $\mathcal{P}_S^- = \mathsf{out}(c, u_S).P \uplus \mathcal{P}$,alors ou bien $\mathcal{Q}_S^-$ ne peut pas rien émettre sur le canal c, et donc $(\mathcal{Q}_S^-; \psi_S^-\theta; \sigma_Q^-\theta; i_Q^-)$, ce qui permet de conclure. Sinon, $\mathcal{Q}_S^- = \mathsf{out}(c, v_S).Q \uplus \mathcal{Q}$ mais $v_S\sigma_Q^-$ n'est pas Σ_0^+-message. Si $v_S\sigma_Q^-$ ne respecte pas les contours, alors considérons θ. Puisque v_S ne contient pas de constantes de Σ_{fresh}, l'égalité $(v_S\sigma_Q^-)\theta = v_S(\sigma_Q^-\theta)$ est vérifiée, et $v_S(\sigma_Q^-\theta)$ ne respecte pas les contours, donc ce n'est pas un Σ_0^--message. Si $v_S\sigma^-Q$ ne respecte pas les sortes, alors considérons θ'. Puisque v_S ne contient pas de constantes de Σ_{fresh} l'égalité $(v_S\sigma_Q^-)\theta' = v_S(\sigma_Q^-\theta')$ est vérifiée et $v_S(\sigma_Q^-\theta)$ ne respecte pas les sortes puisque θ' préserve l'atomicité.
3. $\alpha_S = \mathsf{in}(c, R)$ avec $\mathcal{P}_S^- = \mathsf{in}(c, u_S).P \uplus \mathcal{P}$, alors ou bien $\mathcal{Q}_S^-$ ne peut rien recevoir sur le canal c, et donc $(\mathcal{Q}_S^-; \psi_S^-\theta; \sigma_Q^-\theta; i_Q^-)$ ne peut rien recevoir non plus sur le canal c, ce qui conclut dans ce cas. Sinon, comme $\phi_S^- \sqsubseteq_s \psi_S^-$, et $R\phi_S^-\downarrow$ est un message, le terme $R\psi_S^-\downarrow$ est un message, et $R\psi_S^-\downarrow\theta = (R\psi_S^-)\theta\downarrow = R(\psi_S^-\theta)\downarrow$ est un Σ_0^--message. Donc, comme la réception ne peut pas être exécutée, il y a un problème de filtrage, et il n'existe pas de substitution τ_S telle que $(v_S\sigma_Q^-)\tau = R\psi_S\downarrow$. Supposons *ad absurdum* qu'il existe τ' tel que $(v_S(\sigma_Q^-\theta))\tau' = R(\psi_S^-\theta)\downarrow$. Alors $[(v_S(\sigma_Q^-\theta))\tau']\theta^{-1} = (R(\psi_S^-\theta)\downarrow)\theta^{-1}$, c'est-à-dire $(v_S\sigma_Q^-)(\tau'\theta^{-1}) = R\psi_S^-\downarrow$. Contradiction.

Cas 2 : tr_S s'exécute dans $\mathcal{K}_P$ et $\mathcal{K}_Q$: $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$, $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$, et $\phi_S \not\sqsubseteq_s \psi_S$. D'après le lemme 2.12, $(\mathsf{tr}_S\theta, \phi_S\theta) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ et $(\mathsf{tr}_S\theta, \psi_S\theta) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$, et de même pour θ'. Pour conclure, il suffit de montrer que $\phi_S\theta \not\sqsubseteq \psi_S\theta$, ou alors que $\phi_S\theta' \not\sqsubseteq \psi_S\theta'$. Distinguons deux cas suivant la forme du test, en considérant un test contenant un nombre minimal de symboles.

1. Les termes $R\phi_S\downarrow$ et $R'\phi_S\downarrow$ sont des messages tels que $R\phi_S\downarrow = R'\phi_S\downarrow$. Par minimalité du témoin, $R\psi_S\downarrow$ et $R'\psi_S\downarrow$ sont des messages. Cependant, $R\psi_S\downarrow \neq R'\psi_S\downarrow$. Dans ce cas, considérons θ. Les égalités $R\phi_S\downarrow\theta = R(\phi_S\theta)\downarrow$ (et de même pour R'), et $R\psi_S\downarrow\theta = R(\psi_S\theta)\downarrow$ (et de même pour R') sont vérifiées. Maintenant, supposons que $R(\psi_S\theta)\downarrow = R'(\psi_S\theta)\downarrow$, c'est-à-dire $(R\psi_S\downarrow)\theta = (R'\psi_S\downarrow)\theta$, et en appliquant θ^{-1}, il en découle que $R\psi_S\downarrow = R'\psi_S\downarrow$, ce qui mène à une contradiction.
2. Le terme $R\phi_S\downarrow$ est un message tandis que $R\psi_S\downarrow$ n'en est pas un. Commençons par remarquer que la recette R ne peut pas être réduite à une variable w ou une constante. Donc $R = \mathsf{g}(R_1, \ldots, R_k)$, et par minimalité du test, $R_i\phi_S\downarrow$ est un message. Considérons le cas où $\mathsf{g} \in \Sigma_c$. Comme $R\psi_S\downarrow$ n'est pas un message tandis que $R_1\psi_S\downarrow, \ldots, R_k\psi_S\downarrow$ sont des messages, ou bien $R_i\psi_S\downarrow$ n'est pas un atome alors qu'un atome était attendu à la $i^{\text{ème}}$ position ou $R\phi_S\downarrow$ ne respecte pas les contours. Dans le premier cas, $R_i\phi_S\downarrow$ est un atome. Considérons alors θ', et le test nous permet de voir que $\phi_S\theta' \not\sqsubseteq_s \psi_S\theta'$. Dans le cas où $R\psi_S\downarrow$ ne respecte pas les contours, considérons θ, et le test nous permet de voir que $\phi_S\theta \not\sqsubseteq_s \psi_S\theta$.

□

2.5.2 Équivalence statique

Dans cette section, nous introduisons une définition de l'inclusion statique qui permet à l'attaquant de savoir si un message est de sorte atom. Nous montrerons ensuite que les deux définitions de l'inclusion statique coïncident.

Définition 2.7. *Soit Σ_0 un ensemble de constantes. Soient ϕ et ψ deux Σ_0-trames avec $dom(\phi) = dom(\psi)$. L'inclusion statique étendue par rapport à Σ_0, notée $\phi \sqsubseteq_s^{\mathsf{atom}} \psi$, est définie comme suit.*

- *Pour chaque Σ_0-recette R, si $R\phi\downarrow$ est un Σ_0^+-message, alors $R\psi\downarrow$ est un Σ_0^+-message.*
- *Pour chaque Σ_0-recette R, si $R\phi\downarrow$ est un Σ_0^+-message atomique, alors $R\psi\downarrow$ est un Σ_0^+-message atomique.*
- *Pour toutes Σ_0-recettes R, R', si $R\phi\downarrow = R'\phi\downarrow$ est un Σ_0^+-message, alors $R\psi\downarrow = R'\psi\downarrow$.*

L'exemple suivant illustre la différence entre cette inclusion statique et celle qui a été définie dans le chapitre précédent.

Exemple 2.9. *Considérons la signature $\Sigma_c^{\mathsf{sign}} \uplus \Sigma_d^{\mathsf{sign}}$ de l'exemple 1.10, avec les règles et les sortes associées. Considérons les trames $\phi = \{\mathsf{w} \triangleright n\}$ et $\psi = \{\mathsf{w} \triangleright \mathsf{hash}(n)\}$ où $n \in \mathcal{N}$. La recette w permet de démontrer la non-inclusion $\phi \not\sqsubseteq_s^{\mathsf{atom}} \psi$, car $\mathsf{w}\phi\downarrow$ est un message atomique, mais $\mathsf{w}\psi\downarrow$ n'est pas atomique. Pour démontrer que $\phi \not\sqsubseteq_s \psi$, il faut utiliser une autre recette, par exemple $R' = \mathsf{check}(\mathsf{sign}(\mathsf{w},\mathsf{w}),\mathsf{vk}(\mathsf{w}))$. $R'\phi\downarrow$ est un message, car $\mathsf{w}\phi$ est atomique et donc $R'\phi\downarrow \to \mathsf{ok}$, tandis que $R'\psi\downarrow$ n'est pas un message, car $\mathsf{w}\psi$ n'est pas un atome, donc $\mathsf{sign}(\mathsf{w},\mathsf{w})\psi\downarrow$ n'est pas un message, donc la réduction ne se produit pas.*

Remarquons que dans l'exemple 2.9, l'utilisation de la règle non linéaire $\mathsf{check}(\mathsf{sign}(\mathsf{w},\mathsf{w}),\mathsf{w}) \to \mathsf{ok}$ a été nécessaire pour trouver une recette R'. En l'absence de règle non-linéaire, les deux notions d'équivalence statique $\sqsubseteq_s$ et $\sqsubseteq_s^{\mathsf{atom}}$ ne coïncident pas. Le lemme suivant démontre que les deux définitions sont équivalentes dans notre cas, car nous avons déjà supposé l'existence d'une règle non-linéaire.

Lemme 2.16. *Soient ϕ et ψ deux Σ_0-trames. On a $\phi \sqsubseteq_s \psi$ si, et seulement si, $\phi \sqsubseteq_s^{\mathsf{atom}} \psi$.*

Démonstration. La preuve repose sur la capacité de l'attaquant à simuler les tests d'atomicité en utilisant une règle non-linéaire. Si $\phi \sqsubseteq_s^{\mathsf{atom}} \psi$, alors clairement $\phi \sqsubseteq_s \psi$. Considérons donc l'autre implication. Soient ϕ et ψ deux Σ_0-trames telles que $\phi \sqsubseteq_s \psi$, et il faut montrer que $\phi \sqsubseteq_s^{\mathsf{atom}} \psi$. Soit R une Σ_0-recette telle que $R\phi\downarrow$ est un Σ_0-message atomique. Montrons que $R\psi\downarrow$ est un Σ_0-message atomique. Par hypothèse, il existe $\ell_{\mathsf{des}} \to r_{\mathsf{des}} \in \mathcal{R}$ avec ℓ_{des} non-linéaire, c'est-à-dire que ℓ_{des} contient plusieurs occurences d'une même variable x dont au moins une fois en position de clé. Soit σ la substitution vérifiant $dom(\sigma) = vars(\ell_{\mathsf{des}})$ et $img(\sigma) = \{R\}$, et posons $R' = \ell_{\mathsf{des}}\sigma$. R' est une Σ_0-recette telle que $R'\phi\downarrow$ est un Σ_0-message. En effet, la règle de réécriture s'applique puisque le même message atomique apparaît à chaque position de clé.

D'après l'hypothèse que $\phi \sqsubseteq_s \psi$, les termes $R\psi\downarrow$ et $R'\psi\downarrow$ sont des Σ_0-messages. En particulier, $(\ell_{\mathsf{des}}\sigma)\psi\downarrow$ se réduit en utilisant $\ell_{\mathsf{des}} \to r_{\mathsf{des}}$, ce qui signifie que des messages atomiques se trouvent en position de clé, et donc $R\psi\downarrow$ est un Σ_0-message atomique. □

Un *test* T de non-inclusion est soit une recette R, soit une paire de recettes R, R'. Lorsque c'est une paire de recettes, la mesure μ peut être étendue comme suit.

$$\mu_{\phi_S}(T) = (\mu^1_{\phi_S}(R) \uplus \mu^1_{\phi_S}(R')\,;\, \mu^2_{\phi_S}(R) + \mu^2_{\phi_S}(R')\,;\, \mu^3(R) + \mu^3(R'))$$

2.5.3 Complétude

L'objectif de cette section est de démontrer la complétude. Il s'agit donc d'expliquer comment obtenir un témoin bien typé à partir d'un témoin quelconque. La preuve commence comme pour l'accessibilité, mais nécessite un travail supplémentaire pour montrer que la trace bien typée est effectivement un témoin de non-inclusion, c'est-à-dire que les trames correspondantes ne sont pas en équivalence statique. Il faut donc montrer que tout test de non-inclusion statique peut se tranformer en un test de non-inclusion sur les trames qui résultent des exécutions bien typées.

Comme pour l'accessibilité, nous commençons par établir une proposition, qui démontre que l'on peut se contenter d'explorer les traces qui unifient exclusivement des sous-termes chiffrés. À nouveau, l'hypothèse de conformité permet de reformuler les restrictions sur l'unification en termes de système de types. Nous donnerons une intuition de la preuve, et la démonstration formelle sera laissée pour la section suivante.

Proposition 2.2. *Soient $\mathcal{K}_P$ et $\mathcal{K}_Q$ deux Σ_0^--configurations telles que $\mathcal{K}_Q$ est deterministe, et $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^-. Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$ (avec la substitution sous-jacente σ) un témoin de non-inclusion de longueur minimale. Alors il existe un témoin $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ de cette non-inclusion, avec la substitution sous-jacente σ_S telle que $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ_P and θ vérifiant :*

- *$\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P)$ tels que $u\sigma = v\sigma\}$.*
- *ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constante de Σ_{fresh} tel que $x\rho \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.*
- *$dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, la constante c appartient à $dom(\theta)$ et vérifie $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \prec c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\prec$ est l'ordre induit par tr_S).*
- *pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$.*
- *λ_P est la substitution au premier ordre associée à θ par ϕ_S.*
- *$\phi = \phi_S\lambda_P$, $\sigma = \sigma_S\lambda_P$, et $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$.*

Donnons un schéma détaillé de la preuve. Considérons un témoin de non-inclusion de longueur minimale $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$, avec la substitution sous-jacente σ. Il s'agit d'expliquer comment construire un témoin bien typé. D'après la proposition 2.1, il existe une trace bien typée $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$, de substitution sous-jacente σ_S avec $\sigma_S = mgu(\Gamma)\rho$, et λ_P et θ satisfaisant les conditions de la proposition 2.1. Montrons que (tr_S, ϕ_S) est un témoin de non-inclusion. Il existe une trace $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$, ou alors (tr_S, ϕ_S) est un témoin de non-inclusion. Pour la même raison, $\phi_S \sqsubseteq_s \psi_S$. La première étape de la preuve consiste à montrer que (tr_S, ψ_S) peut être instancié par θ et λ_Q tels que $\psi = \psi_S\lambda_Q$, $\mathsf{tr}\psi{\downarrow} = (\mathsf{tr}_S\theta)\psi{\downarrow}$, et $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_Q)$. La trame ψ satisfait les conditions des lemmes 2.10 et 2.11.

La seconde étape de la preuve permet de monter que $\phi_S \sqsubseteq_s \psi_S$ implique $\phi \sqsubseteq_s \psi$, et il en découle une contradiction. Plus spécifiquement, nous montrons par récurrence sur $\mu_{\phi_S}(T)$, que :

pour chaque test T, si $T\theta$ est vrai dans ϕ alors $T\theta$ est vrai dans ψ. (*)

Il en découle que chaque test T vrai dans ϕ est également vrai dans ψ, quitte à considérer des tests sans constantes de $dom(\theta)$. Donc $\phi \sqsubseteq_s \psi$.

Considérons tous les tests définis par l'inclusion statique $\sqsubseteq_s^{\mathsf{atom}}$. En particulier, l'attaquant peut savoir directement si un message est (ou non) un atome, ce qui évite de considérer une recette de la forme $\mathsf{check}(\mathsf{sign}(R, R), \mathsf{vk}(R)) = \mathsf{ok}$ à la place de la recette plus simple R. (*) se montre par récurrence, à l'aide de la mesure μ. Supposons donc que la propriété (*) soit vraie pour tout test T' tel que $\mu_{\phi_S}(T') < \mu_{\phi_S}(T)$. Les trois cas possibles pour T sont les suivants.

— T vérifie si le terme induit par R est un message. Il faut montrer que si $(R\theta)\phi\downarrow$ alors $(R\theta)\phi\downarrow$ est un message. Si $R\phi_S\downarrow$ est un message alors $R\psi_S\downarrow$ est un message par inclusion statique. Il est alors facile d'en conclure que $(R\theta)\psi\downarrow$ est un message. Si $R\phi_S\downarrow$ n'est pas message, comme dans le cas de l'accessibilité, il suffit de construire de plus petits tests, d'appliquer l'hypothèse de récurrence, et de reconstruire R. Une difficulté supplémentaire vient de ce qu'il faut aussi transférer les propriétés à ψ.

— T vérifie que le terme induit par R est atomique. Il faut montrer que si $(R\theta)\phi\downarrow$ est un message atomique alors $(R\theta)\psi\downarrow$ est également un message atomique. $(R\theta)\psi\downarrow$ est un message d'après le cas précédent. $R\phi_S\downarrow$ est nécessairement atomique, d'une part car $(R\theta)\phi\downarrow = R\phi_S\downarrow\lambda_P$ est atomique, et d'autre part à cause des contraintes sur λ_P. Il en découle que $R\psi_S\downarrow$ est atomique par inclusion statique, donc $(R\theta)\psi\downarrow = R\psi_S\downarrow\lambda_Q$ est atomique car λ_Q préserve l'atomicité.

— T est un test d'égalité $R = R'$. Si R et R' ont un symbole constructeur pour racine, il est possible d'ouvrir chaque recette et d'appliquer l'hypothèse de récurrence. Supposons donc que $R = C[R_1, \ldots, R_n]$ et que la racine de R' est un destructeur. D'après une démonstration analogue à ce qui a été fait pour l'accessibilité, les R_i ainsi que les R' sont des recettes de sous-termes qui donnent des sous-termes chiffrés. Les termes $R\phi_S\downarrow$ et $R'\phi_S\downarrow$ sont des messages. Si $R\phi_S\downarrow = R'\phi_S\downarrow$, alors se résultat se propage à ψ_S et donc à ψ. Sinon $R\phi_S\downarrow \neq R'\phi_S\downarrow$ mais $(R\phi_S)\downarrow\lambda_P = (R'\phi_S)\downarrow\lambda_P$. Considérons la position p sur laquelle les deux termes sont différents, c'est-à-dire que p est une position de $R\phi_S\downarrow$ et de $R'\phi_S$,mais $R\phi_S\downarrow|_p \neq R'\phi_S\downarrow|_p$. Supposons que p est une feuille de $R'\phi_S\downarrow$, et comme λ_P rend ces deux termes égaux, l'égalité $R'\phi_S\downarrow = c \in \Sigma_{\mathsf{fresh}}$ est vérifiée (le cas où p est une feuille de $R\phi_S$ est plus simple).

  — ou bien p appartient à l'un des $R_i\phi_S\downarrow$. Alors, comme pour l'accessibilité, $R_i\phi_S\downarrow$ est un sous-terme chiffré de ϕ_S qui est égal à un sous-terme chiffré de $R'\phi_S\downarrow$, et donc un sous-terme chiffré de ϕ_S puisque R' est une recette de sous-terme. Par le lemme 2.14, ce sont des sous-termes de Γ et donc, par application du mgu, ils sont égaux.

  — Ou bien p tombe dans le contexte C, et alors il suffit de considérer le contexte $\overline{C}$ obtenu à partir de C en remplaçant $C|_p$ par c pour chaque telle position p. En considérant la recette correspondante $\overline{R} = \overline{C}[R_1, \ldots, R_n]$, l'égalité $\overline{R} = R'$ est vraie dans ϕ_S puisque toutes les différences ont été effacées. Donc $\overline{R} = R'$ vaut dans ψ_S. De plus l'égalité $\overline{R} = R$ est vraie par hypothèse de récurrence.

Ce qui conclut le shéma de preuve.

Comme dans le cas de l'accessibilité, $\sigma_S = mgu(\Gamma)\rho$ assure que la trace tr_S est bien typée. En effet, puisque Γ est un ensemble de sous-termes chiffrés unifiables de $\mathcal{K}_P$, les éléments de Γ sont de même type (par conformité) et $mgu(\Gamma)$ est bien typé (par définition d'un système de types). Donc la substitution σ_S est bien typée. Ainsi le théorème 2.2 est une conséquence de la proposition 2.2. Plus formellement, démontrons ce théorème.

Théorème 2.2. *Soient $\mathcal{K}_P$ une Σ_0^--configuration conforme à $(\mathcal{T}_0, \delta_0)$ et $\mathcal{K}_Q$ une Σ_0^--configuration déterministe. On a $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^- si, et seulement si, il existe un témoin bien typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ de cette non-inclusion.*

Démonstration. La réciproque a été démontrée dans le lemme 2.15. Il s'agit donc de démontrer le sens direct. Soient $\mathcal{K}_P$ une Σ_0^--configuration conforme au système $(\mathcal{T}_0, \delta_0)$ et $\mathcal{K}_Q$ une Σ_0^--configuration déterministe. Supposons que $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^-. Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$, avec la substitution sous-jacente σ, un témoin de non-inclusion de longueur minimale. D'après la proposition 2.2, il existe un témoin $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ de cette non-inclusion avec une substitution sous-jacente σ_S telle que $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ_P et θ vérifiant :

— $\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P)$ tels que $u\sigma = v\sigma\}$.

— ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constantes de Σ_{fresh} telles que $x\rho \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.

— $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, la constante c appartient à $dom(\theta)$ et $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \prec c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\prec$ est l'ordre induit par tr_S).

— pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$.

— λ_P est la substitution au premier ordre associée à θ par ϕ_S.

— $\phi = \phi_S\lambda_P$, $\sigma = \sigma_S\lambda_P$, et $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$.

Comme $\mathcal{K}_P$ est conforme au système de types $(\mathcal{T}_0, \delta_0)$, les sous-termes chiffrés unifiables de $ESt(\mathcal{K}_P)$ ont le même type. Ensuite, par définition d'un système de types, $mgu(\Gamma)$ est bien typé. Puisqu'il existe suffisamment de constantes de chaque type, supposons sans perte de généralité que ρ est bien typé. Donc $\sigma_S = mgu(\Gamma)\rho$ est bien typé, ce qui implique que (tr_S, ϕ_S) est un témoin bien typé de $\mathcal{K}_P \not\sqsubseteq_t \mathcal{K}_Q$ par rapport à Σ_0^+, et conclut donc la preuve. □

2.5.4 Démonstration

Montrons maintenant la proposition 2.2.

Proposition 2.2. *Soient $\mathcal{K}_P$ et $\mathcal{K}_Q$ deux Σ_0^--configurations telles que $\mathcal{K}_Q$ est deterministe, et $\mathcal{K}_P \not\sqsubseteq_{at} \mathcal{K}_Q$ par rapport à Σ_0^-. Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$ (avec la substitution sous-jacente σ) un témoin de non-inclusion de longueur minimale. Alors il existe un témoin $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ de cette non-inclusion, avec la substitution sous-jacente σ_S telle que $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ_P and θ vérifiant :*

— $\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P)$ *tels que* $u\sigma = v\sigma\}$.

— *ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers les constante de Σ_{fresh} tel que $x\rho \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.*

— *$dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, la constante c appartient à $dom(\theta)$ et vérifie $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \prec c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\prec$ est l'ordre induit par tr_S).*

— *pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$.*

— *λ_P est la substitution au premier ordre associée à θ par ϕ_S.*

— *$\phi = \phi_S\lambda_P$, $\sigma = \sigma_S\lambda_P$, et $(\mathsf{tr}_S\theta)\phi{\downarrow} = \mathsf{tr}\phi{\downarrow}$.*

Démonstration. Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$ un témoin de non-inclusion de longueur minimale, avec la substitution sous-jacente σ. Commençons par appliquer la proposition 2.1. Il existe $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ avec la substitution sous-jacente σ_S vérifiant $dom(\sigma_S) = dom(\sigma)$ et $\sigma_S = mgu(\Gamma)\rho$, ainsi que deux substitutions λ_P et θ telles que :

— $\Gamma = \{(u, v) \mid u, v \in ESt(\mathcal{K}_P)$ tels que $u\sigma = v\sigma\}$.

— ρ est un renommage bijectif des variables de $dom(\sigma) \setminus dom(mgu(\Gamma))$ vers des constantes de Σ_{fresh} tel que $x\rho \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$ si, et seulement si, $x\sigma$ est un Σ_0^--message atomique.

— $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, pour chaque $c \in \Sigma_{\mathsf{fresh}}$ apparaissant dans tr_S, la constante c appartient à $dom(\theta)$, $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$, et $\mathsf{w}' \prec c$ pour chaque $\mathsf{w}' \in vars(c\theta)$ (où $\prec$ est l'ordre induit par tr_S).

— pour chaque $c \in dom(\theta)$, $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome si $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$.

— λ_P est la substitution au premier ordre associée à θ par ϕ_S.

— $\phi = \phi_S\lambda_P$, $\sigma = \sigma_S\lambda_P$, et $(\mathsf{tr}_S\theta)\phi\downarrow = \mathsf{tr}\phi\downarrow$.

Nous allons montrer que $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ est un témoin de $\mathcal{K}_P \not\sqsubseteq_t \mathcal{K}_Q$ par rapport à Σ_0^+.

Soit ψ_S tel que $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_Q)$. Si jamais un tel ψ_S n'existe pas, le résultat est évidemment vérifié. De plus, $\phi_S \sqsubseteq_s \psi_S$ (et donc $dom(\phi_S) = dom(\psi_S)$), sinon le résultat est également vérifié.

La relation $\prec$ est un ordre sur $dom(\phi_S)$, et ϕ_S est une Σ_0^+-frame. Le domaine de θ vérifie $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$ et $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S))$ pour chaque $c \in dom(\theta)$. De plus, si $c \in \Sigma_{\mathsf{fresh}}$ apparaît dans $\mathsf{w}\phi_S$, alors, comme c n'apparaît pas dans $\mathcal{K}_P$, c un été introduit par une réception avant l'émission correspondant à w. Donc $c \prec \mathsf{w}$. Soit $\mathsf{w}' \in vars(c\theta)$. Par définition de $\prec$, $\mathsf{w}' \prec c$. Donc $\mathsf{w}' \prec \mathsf{w}$ et l'ordre $\prec$ induit sur $dom(\phi_S)$ satisfisfont la condition des lemmes 2.10 et 2.11. $(c\theta)\phi_S\downarrow$ est un Σ_0^+-message pour chaque $c \in dom(\theta)$ et $(c\theta)\phi_S\downarrow$ est un Σ_0^+-message atomique si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. Donc le lemme 2.10 s'applique et pour chaque $c \in dom(\lambda_P)$, $c\lambda_P$ est un Σ_0^--message et $c\lambda_P$ est atomique si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. Donc le lemme 2.11 s'applique, et pour chaque recette R telle que $R\phi_S\downarrow$ est un Σ_0^+-message, l'égalité $(R\theta)(\phi_S\lambda_P)\downarrow = (R\phi_S\downarrow)\lambda_P$ est vérifiée.

$\mathcal{K}_\mathcal{Q}$ est une Σ_0^--configuration, et $(\mathsf{tr}_S, \psi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_\mathcal{Q})$. De plus, $dom(\theta) \subseteq \Sigma_{\mathsf{fresh}}$, et $c \in dom(\theta)$ pour chaque c apparaissant dans tr_S. Comme $\phi_S \sqsubseteq_s \psi_S$, les domaines coïncident ($dom(\phi_S) = dom(\psi_S)$) et donc pour chaque $c \in dom(\theta)$, $c\theta \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\phi_S)) \subseteq \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\psi_S))$. De plus, si $c \in \Sigma_{\mathsf{fresh}}$ apparaît dans $\mathsf{w}\psi_S$, alors, comme c n'apparaît pas dans le protocole, c un été introduite au cours d'une réception qui un eu lieu avant l'émission de w. Donc $c \prec \mathsf{w}$. Soit $\mathsf{w}' \in vars(c\theta)$. Par définition de $\prec$, $\mathsf{w}' \prec c$. Donc $\mathsf{w}' \prec \mathsf{w}$ et l'ordre induit par $\prec$ sur $dom(\psi_S)$ satisfait la condition des lemmes 2.10 et 2.11. Par $\phi_S \sqsubseteq_s \psi_S$, $(c\theta)\psi_S\downarrow$ est un Σ_0^+-message pour chaque $c \in dom(\theta)$ et un atome si $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. Donc les lemmes 2.12 et 2.10 s'appliquent. Il en découle que $(\mathsf{tr}_S\theta, \psi_S\lambda_Q) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_Q)$ où λ_Q est la substitution au premier ordre associée à θ par ψ_S. De plus, $c\lambda_Q$ est un Σ_0^--message pour chaque $c \in dom(\lambda_Q)$ et $c\lambda_Q$ est atomique quand $c \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$. Donc le lemme 2.11 s'applique, et pour chaque $R \in \mathcal{T}(\Sigma, \Sigma_0^- \uplus dom(\theta) \uplus dom(\psi_S))$ tel que $R\psi_S\downarrow$ est un Σ_0^+-message, $(R\theta)(\psi_S\lambda_Q)\downarrow = (R\psi_S\downarrow)\lambda_Q$. De plus, puisque λ_Q préserve l'atomicité, $(R\psi_S\downarrow)\lambda_Q$ est un Σ_0^--message dès que $(R\psi_S\downarrow)$ est un Σ_0^+-message.

Dans la suite, nous supposerons que $\phi_S \sqsubseteq_s \psi_S$ (ce qui signifie que $(\mathsf{tr}_S, \phi_S) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_P)$ n'est pas un témoin), et il faudra montrer que $\phi \sqsubseteq_s \psi$, ce qui mène à une contradiction puisque par hypothèse, $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_P)$ est un témoin, et $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^-}(\mathcal{K}_Q)$.

Le lemme 2.16 nous autorise à raisonner avec l'inclusion $\sqsubseteq_s^{\mathsf{atom}}$. Considérons un test T (construit sur $\mathcal{T}(\Sigma, \Sigma_0^+ \cup dom(\phi_S))$ tel que $T\theta$ est vrai ϕ. Supposons que pour tout T' vérifiant $\mu_{\phi_S}(T') < \mu_{\phi_S}(T)$, l'implication suivante soit vérifiée :

$$\text{si } T'\theta \text{ est vrai dans } \phi, \text{ alors } T'\theta \text{ est vrai dans } \psi.$$

Il faut montrer que $T\theta$ est vrai dans ψ. Distinguons trois cas suivant la forme du test :

1. Le test T est une Σ_0^+-recette R telle que $(R\theta)\phi\downarrow$ est un Σ_0^--message. Dans ce cas, il faut montrer que $(R\theta)\psi\downarrow$ est un Σ_0^--message.
2. Le test T est une Σ_0^+-recette telle que $(R\theta)\phi\downarrow$ est un Σ_0^--message atomique, c'est-à-dire $(R\theta)\phi\downarrow \in \Sigma_0^- \uplus \mathcal{N}$. Dans ce cas, il faut établir que $(R\theta)\psi\downarrow$ est un Σ_0^--message atomique.
3. Le test T est une paire de Σ_0^+-recettes R, R' telles que $(R\theta)\phi\downarrow$ et $(R'\theta)\phi\downarrow$ sont des Σ_0^--messages, et $(R\theta)\phi\downarrow = (R'\theta)\phi\downarrow$. Dans ce cas, il faut montrer que $(R\theta)\psi\downarrow = (R'\theta)\psi\downarrow$.

(1) *R est une Σ_0^+-recette telle que $(R\theta)\phi\!\downarrow$ est un Σ_0^--message.*

Supposons que $R\phi_S\!\downarrow$ n'est pas un Σ_0^+-message. Prenons un plus petit sous-terme R' de R tel que $R'\phi_S\!\downarrow$ n'est pas un Σ_0^+-message. Soit p tel que $R|_p = R'$. Comme $R'\phi_S\!\downarrow$ n'est pas un Σ_0^+-message, $R' \notin \Sigma_0^+ \uplus dom(\phi_S)$. Donc $R' = \mathsf{f}(R_1, \ldots, R_k)$ pour un certain $\mathsf{f} \in \Sigma$. De plus, par minimalité of R', $R_i\phi_S\!\downarrow$ est un Σ_0^+-message pour chaque $1 \leqslant i \leqslant k$. Établissons maintenant le fait suivant.

Fait 1. Si $R_i\phi_S\!\downarrow \in \Sigma_{\mathsf{fresh}}$ pour un certain $i \in \{1, \ldots, k\}$, alors $(R\theta)\psi\!\downarrow$ est un Σ_0^--message.

Démonstration. Supposons $R_i\phi_S\!\downarrow = c \in \Sigma_{\mathsf{fresh}}$ pour un certain $i \in \{1, \ldots, k\}$, et soit $R'_i = c\theta$. Alors $R'_i\theta = c\theta$, et donc $(R'_i\theta)\phi\!\downarrow = (c\theta)\phi\!\downarrow$, et $(R'_i\theta)\psi\!\downarrow = (c\theta)\psi\!\downarrow$. Le test $c = R_i$ est vrai dans ϕ_S, et donc puisque $\phi_S \sqsubseteq_s \psi_S$, il est vrai aussi dans ψ_S. Le lemme 2.11 s'applique : $(R_i\theta)\psi\!\downarrow = (R_i\psi_S\!\downarrow)\lambda_Q = c\lambda_Q = (c\theta)\psi\!\downarrow = (R'_i\theta)\psi\!\downarrow$. Soit $\overline{R} = R[R'_i]_{(p.i)}$.

— $(\overline{R}\theta = R\theta)$ est vrai à la fois dans ϕ et dans ψ ;

— $\mu^1_{\phi_S}(R'_i) < \mu^1_{\phi_S}(R_i)$ d'après le lemme 2.9, et donc $\mu^1_{\phi_S}(\overline{R}) < \mu^1_{\phi_S}(R)$ d'après les lemmes 2.6 et 2.7.

Comme $(\overline{R}\theta)\phi\!\downarrow = (R\theta)\phi\!\downarrow$ est un Σ_0^--message, un tel test se transfère à ψ (par hypothèse de récurrence), et $(\overline{R}\theta)\psi\!\downarrow$ est un Σ_0^--message, ce qui prouve le fait.

Distinguons maintenant de cas selon que $\mathsf{f} \in \Sigma_{\mathsf{c}}$ ou $\mathsf{f} \in \Sigma_{\mathsf{d}}$.

Cas où $\mathsf{f} \in \Sigma_{\mathsf{c}}$. Dans ce cas, $R'\phi_S\!\downarrow$ ne respecte pas les contours ou les sortes. Si $R'\phi_S\!\downarrow$ ne respecte pas les sortes, alors pour l'une des ses sous-recettes R_i, $R_i\phi_S\!\downarrow$ n'est pas un atome ($R_i\phi_S\!\downarrow$ respecte les sortes par minimalité of R') alors qu'un atome était attendu. En particulier, $(R_i\theta)\phi\!\downarrow = R_i\phi_S\!\downarrow\lambda_P$ est un atome. Donc $R_i\phi_S\!\downarrow$ est une constante de Σ_{fresh}, ce qui implique, d'après le fait 1, que $(R\theta)\psi\!\downarrow$ est un Σ_0^--message, d'où le résultat. Il en découle que $R'\phi_S\!\downarrow$ respecte les sortes.

Maintenant, supposons que $R'\phi_S\!\downarrow$ ne respecte pas les coutours, et considérons le coutour de f, $\mathsf{sh}_{\mathsf{f}} = \mathsf{f}(s_1, \ldots, s_k)$ pour certains $s_1, \ldots, s_k$. Comme $R'\phi_S\!\downarrow$ ne respecte pas les coutours et un f pour racine, il existe un j tel que $R_j\phi_S\!\downarrow$ n'est pas une instance de s_j. En particulier, s_j n'est pas une variable et $s_j = \mathsf{sh}_{\mathsf{g}}$ pour un certain symbole de fonction g (d'après la compatibilité des coutours). Mais $R_j\phi_S\!\downarrow$ est un Σ_0^+-message et donc $(R_j\theta)\phi\!\downarrow = (R_j\phi_S\!\downarrow)\lambda_P$ (d'après le lemme 2.11) est une instance de s_j comme $(R_j\theta)\phi\!\downarrow$ est un Σ_0^+-message. D'après le fait 1, $R_j\phi_S\!\downarrow \notin \Sigma_{\mathsf{fresh}}$, donc $R_j\phi_S\!\downarrow$ un g pour racine, et puisque $R_j\phi_S\!\downarrow$ est un Σ_0^+-message, c'est une instance de s_j, ce qui mène à une contradiction.

Cas où $\mathsf{f} = \mathsf{des} \in \Sigma_{\mathsf{d}}$. Dans ce cas, $R' = \mathsf{des}(R_1, \ldots, R_k)$. Soit $\ell_{\mathsf{des}} = \mathsf{des}(t_1, t_2, \ldots, t_k)$. Distinguons deux sous-cas.

Premièrement, supposons qu'il existe $i \in \{1, \ldots, k\}$ tel que $R_i\phi_S\!\downarrow$ ne s'unifie pas avec t_i. Comme $R_i\phi_S\!\downarrow$ est un Σ_0^+-message, $R_i\phi_S\!\downarrow$ respecte les contours. La seule raison pour laquelle $R_i\phi_S\!\downarrow$ pourrait ne pas s'unifier avec t_i tandis que $(R_i\phi_S)\lambda_P = (R_i\theta)\phi\!\downarrow$ (d'après le lemme 2.11) s'unifie avec t_i, serait que $R_i\phi_S\!\downarrow = c$ pour un certain $c \in \Sigma_{\mathsf{fresh}}$. Ce qui contredit le fait 1.

Deuxièmement, supposons que $R_i\phi_S\!\downarrow$ s'unifie avec t_i pour chaque i. Dans ce cas, le terme $\ell_{\mathsf{des}} = \mathsf{des}(t_1, \ldots, t_k)$ n'est pas linéaire. Posons x la variable non-linéaire apparaissant dans ℓ_{des}. Soit $I_0 = \{1 \leqslant i \leqslant k \mid x \text{ apparaît dans } t_i\}$. Remarquons que $1 \in I_0$. Pour chaque $i \in I_0$, posons p_i la position de t_i telle que $t_i|_{p_i} = x$. Puisque $R'\phi_S\!\downarrow\lambda_P$ est un Σ_0^--message, $t = \mathsf{des}(R_1\phi_S\!\downarrow\lambda_P, \ldots, R_k\phi_S\!\downarrow\lambda_P)$ s'unifie avec $\mathsf{des}(t_1, \ldots, t_k)$. En conséquence, il existe un Σ_0^--message atomique a tel que $t|_{i.p_i} = a$ pour chaque $i \in I_0$. Le terme $R'\phi_S\!\downarrow$ n'est pas un Σ_0^+-message tandis que chaque $R_i\phi_S\!\downarrow$ est un Σ_0^+-message. Donc $t_S = \mathsf{des}(R_1\phi_S\!\downarrow, \ldots, R_k\phi_S\!\downarrow)$ ne s'unifie pas avec $\mathsf{des}(t_1, \ldots, t_k)$. D'où le fait suivant :

Fait 2 : Pour chaque $i \in I_0$, ou bien $t_S|_{i.p_i} = a$, ou bien $t_S|_{i.p_i} = c$ pour un certain $c \in \Sigma_{\mathsf{fresh}}$ tel que $c\lambda_P = (c\theta)\phi\!\downarrow = a$.

Soit c une constante de Σ_{fresh} telle que $t_S|_{i.p_i} = c$ pour un certain $i \in I_0$. Soit $I_1 = \{i \in I_0 \mid R_i\phi_S{\downarrow}|_{p_i} = c\}$. Construisons deux recettes $\overline{R}'$ et $\overline{R}''$ derivées de R' et munies de propriétés agréables : en particulier $(\overline{R}'\theta)\psi{\downarrow}$ et $(\overline{R}''\theta)\psi{\downarrow}$ seront des Σ_0^--messages, ce qui permettra de montrer que $(R\theta)\psi{\downarrow}$ est également un Σ_0^--message.

Construction de $\overline{R}'$. Soit ν'_{des} la substitution telle que $x\nu'_{\mathsf{des}} = c\theta$, et $y\nu'_{\mathsf{des}} = c_{\mathsf{min}}$ pour toute autre variable $y \in vars(\ell_{\mathsf{des}})$. Pour $i \in \{1, \ldots, k\}$, soit $R'_i = t_i\nu'_{\mathsf{des}}$ quand $i \in I_1$, et $R'_i = R_i$ sinon. Soit $\overline{R}' = \mathsf{des}(R'_1, \ldots, R'_k)$. Chaque $R'_i\phi_S{\downarrow}$ est un Σ_0^+-message, et donc par le lemme 2.11, le terme $(R'_i\theta)\phi{\downarrow} = R'_i\phi_S{\downarrow}\lambda_P$ est un Σ_0^--message. Par construction de $\overline{R}'$ et d'après le fait 2, $(\overline{R}'\theta)\phi{\downarrow}$ est un Σ_0^--message. Par conséquent, d'après le lemme 2.6, et puisque $\mu^1_{\phi_S}(R'_i) < \mu^1_{\phi_S}(R_i)$ pour chaque $i \in I_1 \neq \emptyset$, et $\mu^1_{\phi_S}(R'_i) = \mu^1_{\phi_S}(R_i)$ sinon, il en découle :

$$\begin{array}{rcl}
\mu^1_{\phi_S^-}(\overline{R}') & = & \mathsf{Multi}(\overline{R'}\phi_S^-{\downarrow}) \\
& \leqslant & \mathsf{Multi}(\mathsf{des}(R'_1\phi_S{\downarrow}, \ldots, R'_k\phi_S{\downarrow})) \\
& = & \{\mathsf{des}\} \uplus \mu^1_{\phi_S}(R'_1) \uplus \cdots \uplus \mu^1_{\phi_S}(R'_k) \\
& < & \{\mathsf{des}\} \uplus \mu^1_{\phi_S}(R_1) \uplus \cdots \uplus \mu^1_{\phi_S}(R_k) \\
& = & \mu^1_{\phi_S}(R)
\end{array}$$

Comme $(\overline{R}'\theta)\phi{\downarrow}$ est un Σ_0^--message, $(\overline{R}'\theta)\psi{\downarrow}$ est un Σ_0^--message. Puisque $R'\phi_S{\downarrow}$ n'est pas un Σ_0^+-message, le lemme 2.7 s'applique, et ainsi $\mu^1_{\phi_S}(R[\overline{R}']_p) < \mu^1_{\phi_S}(R[R']_p) = \mu^1_{\phi_S}(R)$.

Construction of $\overline{R''}$. Soit ν''_{des} la substitution telle que $x\nu''_{\mathsf{des}} = c$, et $y\nu''_{\mathsf{des}} = c_{\mathsf{min}}$ pour toute autre variable $y \in vars(\ell_{\mathsf{des}})$. Pour $i \in \{1, \ldots, k\}$, soit $R''_i = R_i$ si $i \in I_1$, et $R''_i = t_i\nu''_{\mathsf{des}}$ sinon. Soit $\overline{R}'' = \mathsf{des}(R''_1, \ldots, R''_k)$. Par construction de $\overline{R}''$, $\overline{R}''\phi_S{\downarrow}$ est un Σ_0^+-message. Par hypothèse $\phi_S \sqsubseteq_s \psi_S$, et donc $\overline{R}''\psi_S{\downarrow}$ est un Σ_0^+-message. Alors, d'après le lemme 2.11, $(\overline{R}''\theta)\psi{\downarrow} = \overline{R}''\psi_S{\downarrow}\lambda_Q$. En ce qui concerne la mesure, $\{\mathsf{des}\} < \mu^1_{\phi_S}(R')$ puisque des apparaît dans $R'\phi_S{\downarrow}$. $\mu^1_{\phi_S}(\overline{R}'') < \{\mathsf{des}\}$ puisque $\overline{R}''\phi_S{\downarrow}$ est un Σ_0^+-message. Par conséquent, $\mu^1_{\phi_S}(\overline{R}'') < \mu^1_{\phi_S}(R')$. Puisque $R'\phi_S{\downarrow}$ n'est pas un Σ_0^+-message, le lemme 2.7 s'applique, donc $\mu^1_{\phi_S}(R[\overline{R}'']_p) < \mu^1_{\phi_S}(R[R']_p) = \mu^1_{\phi_S}(R)$.

Arrivés à ce point, $\overline{R}' = \mathsf{des}(R'_1, \ldots, R'_k)$, $\overline{R}'' = \mathsf{des}(R''_1, \ldots, R''_k)$, et $(\overline{R}'\theta)\psi{\downarrow}$ et $(\overline{R}''\theta)\psi{\downarrow}$ sont tous deux des Σ_0^--messages. Par construction, pour chaque $1 \leqslant i \leqslant k$, ou bien $R_i = R'_i$ ou bien $R_i = R''_i$. Par conséquent, pour chaque $1 \leqslant i \leqslant k$, $(R_i\theta)\psi{\downarrow}$ est un Σ_0^--message et s'unifie avec t_i. Plaçons un $c\theta$ en position de clé de ℓ_{des} dans un certain R'_i. Comme $(\overline{R}'\theta)\psi{\downarrow}$ est un Σ_0^- message, chacun des $(R'_i\theta)\psi{\downarrow}$ a $(c\theta)\psi{\downarrow}$ en position de clé de ℓ_{des}. De même, chacun des $(R''_i\theta)\psi{\downarrow}$ a $(c\theta)\psi{\downarrow}$ en position de clé de ℓ_{des}. So there est $(c\theta)\psi{\downarrow}$ en position de clé de ℓ_{des} pour chaque $(R_i\theta)\psi{\downarrow}$. Il en découle que $(R'\theta)\psi{\downarrow}$ est un Σ_0^--message.

De plus, $R_1 = R'_1$ ou $R_1 = R''_1$. Soit $\overline{R} = \overline{R}'$ si $R_1 = R'_1$, et $\overline{R} = \overline{R}''$ si $R_1 = R''_1$. Alors :

— $\mu^1_{\phi_S}(R[\overline{R}]_p) < \mu^1_{\phi_S}(R)$,

— $(R[\overline{R}]_p\theta)\phi{\downarrow} = (R[R']_p\theta)\phi{\downarrow} = (R\theta)\phi{\downarrow}$, et

— $(R[\overline{R}]_p\theta)\psi{\downarrow} = (R\theta)\psi{\downarrow}$.

Comme $(R[\overline{R}]_p\theta)\phi{\downarrow}$ est un Σ_0^--message, par minimalité $(R[\overline{R}]_p\theta)\psi{\downarrow}$ est un Σ_0^--message, et ainsi $(R\theta)\psi{\downarrow}$ est un Σ_0^--message, ce qui est le résultat à démontrer.

Donc $R\phi_S{\downarrow}$ est un Σ_0^+-message. Par $\phi_S \sqsubseteq_s \psi_S$, le terme $R\psi_S{\downarrow}$ est un Σ_0^+-message, et le lemme 2.11 permet de conclure que $(R\theta)\psi{\downarrow} = R\psi_S{\downarrow}\lambda_Q$ est un Σ_0^--message.

(2) *R est une Σ_0^+-recette telle que $(R\theta)\phi{\downarrow}$ est un Σ_0^--message atomique.*

Pour commencer, $R\phi_S\downarrow$ est un Σ_0^+-message (d'après le cas (1)), et $(R\theta)\phi\downarrow = R\phi_S\downarrow\lambda_P$ par le lemme 2.11. Comme premère étape, établissons que $R\psi_S\downarrow$ est atomique. Comme $R\phi_S\downarrow\lambda_P$ est un atome, $R\phi_S\downarrow$ est un atome de $\Sigma_0^- \cup \mathcal{N}$, ou une constante de Σ_{fresh}.

— Si $R\phi_S\downarrow \notin \Sigma_{\mathsf{fresh}}$, alors $R\phi_S\downarrow = R\phi_S\downarrow\lambda_P$ est atomique, et par hypothèse $\phi_S \sqsubseteq_s \psi_S$, donc $R\psi_S\downarrow$ est atomique.

— Si $R\phi_S\downarrow = c \in \Sigma_{\mathsf{fresh}}$, alors il existe x tel que $x\rho = c$. Puisque $x \in dom(\rho)$, $x \notin dom(mgu(\Gamma))$, et donc $xmgu(\Gamma) = x$. Par conséquent, $x\sigma = x\sigma_S\lambda_P = x(mgu(\Gamma)\rho)\lambda_P = (x\rho)\lambda_P = c\lambda_P = R\phi_S\downarrow\lambda_P$. Comme $R\phi_S\downarrow\lambda_P$ est un atome, $x\sigma$ est atomique, et par définition de ρ, c vaut $c = x\rho \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$, et donc $R\phi_S\downarrow$ est atomique. D'après l'hypothèse $\phi_S \sqsubseteq_s \psi_S$, le message $R\psi_S\downarrow = c$ est atomique.

Puisque λ_Q remplace les atomes par des atomes, dans les deux cas $R\psi_S\downarrow\lambda_Q$ est atomique, et donc $(R\theta)\psi\downarrow = (R\psi_S\downarrow)\lambda_Q$ (voir lemme 2.11) est un Σ_0^--message atomique.

(3) *R et R' sont des Σ_0^+-recettes, $(R\theta)\phi\downarrow$, $(R'\theta)\phi\downarrow$ sont des Σ_0^--messages, et $(R\theta)\phi\downarrow = (R'\theta)\phi\downarrow$.*

Étape 1 : Montrons que R et R' sont des Σ_0^+-recettes simples, en forme normale pour $\twoheadrightarrow$. De plus, R' est une recette de sous-terme telle que $R'\phi_S\downarrow$ est soit un terme chiffré, soit un nom de $\mathcal{N}$, soit une constante de Σ_0^+. Montrons que R est de la forme $C[R_1, \ldots, R_n]$ où pour chaque $i \in \{1, \ldots, n\}$, $R_i\phi_S\downarrow$ est ou bien un terme chiffré, ou bien un nom de $\mathcal{N}$, ou encore une constante de Σ_0^+.

$(R\theta)\phi\downarrow$ et $(R'\theta)\phi\downarrow$ sont des Σ_0^--messages, $\mu_{\phi_S}(R) < \mu_{\phi_S}(R = R')$ et $\mu_{\phi_S}(R') < \mu_{\phi_S}(R = R')$. Donc $(R\theta)\psi\downarrow$ et $(R'\theta)\psi\downarrow$ sont des Σ_0^--messages.

Supposons que R ou R' n'est pas en forme normale pour $\twoheadrightarrow$, et sans perte de généralité supposons que c'est R. Alors $\mu^1_{\phi_S}(R\downarrow) < \mu^1_{\phi_S}(R)$ d'après le lemme 2.8. De plus, comme $R\theta \twoheadrightarrow^* R\downarrow\theta$, le lemme 2.2 s'applique : $(R\downarrow\theta)\phi\downarrow = (R\theta)\phi\downarrow$ et $(R\downarrow\theta)\psi\downarrow = (R\theta)\psi\downarrow$ comme $(R\theta)$ donne un Σ_0^--message à la fois dans ϕ et ψ. Donc $(R\downarrow\theta = R'\theta)$ est vrai dans ϕ. Par $\mu_{\phi_S}(R\downarrow = R') < \mu_{\phi_S}(R = R')$, ce test se trasnfère dans ψ et donc $(R\theta)\psi\downarrow = (R\downarrow\theta)\psi\downarrow = (R'\theta)\psi\downarrow$ ce qui est le résultat à démontrer.

Maintenant, supposons que R et R' sont en forme normale pour $\twoheadrightarrow$. Alors ce sont des recettes simples d'après le lemme 2.3. Si R et R' ont un constructeur pour racine, alors c'est nécessairemet le même. Par conséquent, $R = \mathsf{f}(R_1, \ldots, R_k)$ et $R' = \mathsf{f}(R'_1, \ldots, R'_k)$. $(R_i\theta)\phi\downarrow$ et $(R'_i\theta)\phi\downarrow$ sont des Σ_0^+-messages (pour $1 \leqslant i \leqslant k$), et $(R_i\theta)\phi\downarrow = (R'_i\theta)\phi\downarrow$ $(1 \leqslant i \leqslant k)$. Puisque $\mu_{\phi_S}(R_i = R'_i) < \mu_{\phi_S}(R = R')$, l'égalité $(R_i\theta)\psi\downarrow = (R'_i\theta)\psi\downarrow$ est vérifiée, et donc $(R\theta)\psi\downarrow = (R'\theta)\psi\downarrow$ ce qui est le résultat à démontrer.

Par conséquent, supposons sans perte de généralité que R' est une recette de sous-terme, et que R est simple, donc $R = C[R_1, \ldots, R_n]$ et pour chaque i, R_i est une recette de sous-terme. Maintenant, supposons qu'il existe i_0 tel que $\mathsf{root}(R_{i_0}\phi_S\downarrow) = \mathsf{f} \in \Sigma_{\mathsf{c}}$ avec f un symbole transparent, alors $R_{i_0}\phi_S\downarrow = \mathsf{f}(C^{\mathsf{f}}_1[R_{i_0}], \ldots, C^{\mathsf{f}}_k[R_{i_0}])\phi_S\downarrow$. Considérons le contexte $\overline{C}$ tel que $\overline{C}[R_1, \ldots, R_n] = C[R_1, \ldots, \mathsf{f}(C^{\mathsf{f}}_1[R_{i_0}], \ldots, C^{\mathsf{f}}_k[R_{i_0}]), \ldots, R_n]$. La Σ_0^+-recette $\overline{R} = \overline{C}[R_1, \ldots, R_n]$ vérifie $\overline{R}\phi_S\downarrow = R\phi_S\downarrow$, et donc $\mu^1_{\phi_S}(R) = \mu^1_{\phi_S}(\overline{R})$, et $(\overline{R}\theta)\phi\downarrow = \overline{R}\phi_S\downarrow\lambda_P$ d'après le lemme 2.11, ce qui donne $(\overline{R}\theta)\phi\downarrow = \overline{R}\phi_S\downarrow\lambda_P = R\phi_S\downarrow\lambda_P = (R\theta)\phi\downarrow$. Par ailleurs, $\mu^2_{\phi_S}(\overline{R}) < \mu^2_{\phi_S}(R)$. Comme $\phi_S \sqsubseteq_s \psi_S$, l'égalité $\overline{R}\phi_S\downarrow = R\phi_S\downarrow$ se transfère à ψ_S : $\overline{R}\psi_S\downarrow = R\psi_S\downarrow$. Il en découle que $(\overline{R}\theta)\psi\downarrow = (R\theta)\psi\downarrow$ d'après le lemme 2.11. Comme $\mu_{\phi_S}(\overline{R} = R') < \mu_{\phi_S}(R = R')$, l'égalité $(\overline{R}\theta)\phi\downarrow = (R'\theta)\phi\downarrow$ se transfère en $(\overline{R}\theta)\psi\downarrow = (R'\theta)\psi\downarrow$ et il en découle que $(R\theta)\psi\downarrow = (R'\theta)\psi\downarrow$, ce qui est le résultat à démontrer.

Par conséquent, chaque $R_i\phi_S\downarrow$ (pour $1 \leqslant i \leqslant n$) est un terme chiffré, une constante de Σ_0^+, ou un nom de $\mathcal{N}$. Un raisonnement similaire permet d'établir que $R'\phi_S\downarrow$ est un terme chiffré, une constante de Σ_0^+, ou un nom de $\mathcal{N}$.

Étape 2 : Établissons maintenant que $(R\theta)\psi\downarrow = (R'\theta)\psi\downarrow$.

Soit $t = R\phi_S{\downarrow}$ et $v = R'\phi_S{\downarrow}$. Le lemme 2.11 et $(R\theta)\phi{\downarrow} = (R'\theta)\phi{\downarrow}$, permettent de déduire que $t\lambda_P = v\lambda_P$. Si $t = v$, alors $R\phi_S{\downarrow} = R'\phi_S{\downarrow}$ puisque $\phi_S \sqsubseteq_s \psi_S$. Donc d'après le lemme 2.11, $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$.

À partir de maintenant, supposons que $t \neq v$. Puisque $t \neq v$, il existe une position p de t et v telle que $\mathsf{root}(t|_p) \neq \mathsf{root}(v|_p)$. Soit p une position quelconque définie dans t et v telle que $\mathsf{root}(t|_p) \neq \mathsf{root}(v|_p)$. Puisque $t\lambda_P = v\lambda_P$, et $dom(\lambda_P) \subseteq \Sigma_{\mathsf{fresh}}$, $t|_p \in \Sigma_{\mathsf{fresh}}$ ou $v|_p \in \Sigma_{\mathsf{fresh}}$.

Supposons qu'il existe une telle position p qui tombe en dehors du contexte C. Plus précisément, $p = p'.p''$ (avec p' un préfixe strict de p) et $C[R_1, \ldots, R_n]|_{p'} = R_{i_0}$ pour un certain $i_0 \in \{1, \ldots, n\}$. Par conséquent, puisque $R_{i_0}\phi_S{\downarrow}$ n'est pas une feuille ($p'' \neq \epsilon$), $t|_{p'} = R_{i_0}\phi_S{\downarrow}$ est un sous-terme chiffré de ϕ_S. Le lemme 2.14 permet d'obtenir :

— $t|_{p'} \in \mathit{ESt}(\phi_S) \subseteq \mathit{ESt}(\mathcal{K}_S\sigma_S) \subseteq \mathit{ESt}(\mathcal{K}_P(mgu(\Gamma)\rho)) \subseteq \mathit{ESt}(\mathcal{K}_P)\sigma_S$.

— $v|_{p'} \in \mathit{ESt}(\phi_S) \subseteq \mathit{ESt}(\mathcal{K}_S\sigma_S) \subseteq \mathit{ESt}(\mathcal{K}_P(mgu(\Gamma)\rho)) \subseteq \mathit{ESt}(\mathcal{K}_P)\sigma_S$.

Il existe donc $t', v' \in \mathit{ESt}(\mathcal{K}_P)$ tels que $t'\sigma_S = t|_{p'}$, et $v'\sigma_S = v|_{p'}$. Puisque $t\lambda_P = v\lambda_P$, $(t\lambda_P)|_{p'} = (v\lambda_P)|_{p'}$, donc $t|_{p'}\lambda_P = v|_{p'}\lambda_P$, et $(t'\sigma_S)\lambda_P = (v'\sigma_S)\lambda_P$. Les égalités suivantes sont vérifiées :

$$(t'\sigma_S)\lambda_P = t'(\sigma_S\lambda_P)$$
$$(v'\sigma_S)\lambda_P = v'(\sigma_S\lambda_P)$$

L'égalité $t'\sigma = v'\sigma$ découle de $\sigma = \sigma_S\lambda_P$. L'égalité $t'\sigma_S = v'\sigma_S$ provient directement de la définition de σ_S. Donc, $t|_{p'} = v|_{p'}$ ce qui mène à une contradiction avec la supposition que t et v sont différents en-dessous de la position p'.

Donc, pour chaque position p de t et v telle que $\mathsf{root}(t|_p) \neq \mathsf{root}(v|_p)$, $t|_p$ ou $v|_p$ est une constante de Σ_{fresh}, et p est une position de C.

Si $t|_p = c \in \Sigma_{\mathsf{fresh}}$, soit $\overline{R} = R[c\theta]_p$. La comparaison $\mu^1_{\phi_S}(\overline{R}) < \mu^1_{\phi_S}(R)$ se déduit de $\mu^1_{\phi_S}(c\theta) < \mu^1_{\phi_S}(R|_p)$ (remarquons que $R|_p\phi_S{\downarrow} = c$ tandis que $(c\theta)\phi_S{\downarrow}$ est un Σ_0^+-message et c'est un atome quand $c \in \Sigma^{\mathsf{atom}}_{\mathsf{fresh}}$). De plus :

— $(\overline{R}\theta)\phi{\downarrow} = (R\theta)\phi{\downarrow} = (R'\theta)\phi{\downarrow}$, et

— $\mu_{\phi_S}(\overline{R} = R') < \mu_{\phi_S}(R = R')$.

Donc, par hypothèse de récurrence, $(\overline{R}\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$. De plus $R|_p\phi_S{\downarrow} = c$ donc par $\phi_S \sqsubseteq_s \psi_S$, la relation $R|_p\psi_S{\downarrow} = c$ est vérifiée. D'après le lemme 2.11, $(R|_p\theta)\psi{\downarrow} = (c\theta)\psi{\downarrow}$. Donc $(\overline{R}\theta)\psi{\downarrow} = (R\theta)\psi{\downarrow}$. Il en découle que $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$ ce qui est le résultat à démontrer.

À partir de maintenant, supposons que $t|_p \notin \Sigma_{\mathsf{fresh}}$, et donc $v|_p = c$ pour un certain $c \in \Sigma_{\mathsf{fresh}}$. Soit $p_1, \ldots, p_m$ les positions telles que pour chaque $i \in \{1, \ldots, m\}$, la position p_i est définie à la fois dans t et dans v, et $\mathsf{root}(t|_{p_i}) \neq \mathsf{root}(v|_{p_i})$. Pour chaque $i \in \{1, \ldots, m\}$, la position p_i est définie dans C et vérifie $t|_{p_i} \notin \Sigma_{\mathsf{fresh}}$, et $v|_{p_i} \in \Sigma_{\mathsf{fresh}}$. Notons c^i_{fresh} la constante de Σ_{fresh} telle que $v|_{p_i} = c^i_{\mathsf{fresh}}$. L'égalité $(c^i_{\mathsf{fresh}}\theta)\phi{\downarrow} = (R|_{p_i}\theta)\phi{\downarrow}$ est vérifiée. Les comparaisons $\mu^1_{\phi_S}(c^i_{\mathsf{fresh}}) < \mu^1_{\phi_S}(R')$ et $\mu^1_{\phi_S}(R|_{p_i}) \leqslant \mu^1_{\phi_S}(R)$ impliquent $\mu_{\phi_S}(c^i_{\mathsf{fresh}} = R|_{p_i}) < \mu_{\phi_S}(R = R')$. Il en découle que $(c^i_{\mathsf{fresh}}\theta)\psi{\downarrow} = (R|_{p_i}\theta)\psi{\downarrow}$.

Maintenant, soit $\overline{C}$ le contexte obtenu à partir de C en plaçant c^i_{fresh} à la position p_i pour chaque i. Soit $\overline{R} = \overline{C}[R_1, \ldots, R_n]$. $(\overline{R}\theta)\psi{\downarrow} = (R\theta)\psi{\downarrow}$ par $(c^i_{\mathsf{fresh}}\theta)\psi{\downarrow} = (R|_{p_i}\theta)\psi{\downarrow}$. De plus, $\overline{R}\phi_S{\downarrow} = R'\phi_S{\downarrow}$ par construction. Par $\phi_S \sqsubseteq_s \psi_S$, l'égalité se transfère en $\overline{R}\psi_S{\downarrow} = R'\psi_S{\downarrow}$. D'après le lemme 2.11, $(\overline{R}\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$, et donc $(R\theta)\psi{\downarrow} = (R'\theta)\psi{\downarrow}$ ce qui est le résultat à démontrer. □

2.6 Retour sur les hypothèses

Cette section a pour objectif de donner des exemples de théories qui ne satisfont pas les hypothèses du modèle et pour lesquelles le théorème 2.2 ne s'applique plus. Dans chaque cas, il existe une attaque, mais aucun témoin bien typé, ce qui montre que les hypothèses ne peuvent pas être étendues facilement.

Dans cette section, on considère $n, m \in \mathcal{N}$ et $a, b \in \Sigma_0^-$.

2.6.1 Contours

Montrons d'abord le rôle des contours, grâce aux processus suivants.

$$\begin{array}{rcl} P & = & \mathsf{in}(c,x).\mathsf{out}(c,\mathsf{aenc}(\langle a,n\rangle,x)) \\ Q & = & \mathsf{in}(c,x).\mathsf{out}(c,\mathsf{hash}(n)) \end{array}$$

P n'est pas un processus de notre algèbre à cause du terme $\mathsf{aenc}(\langle a,n\rangle,x))$ qui ne respecte pas les contours, puisque x apparaît en position de clé tandis qu'un terme de la forme $\mathsf{pub}(t)$ aurait été attendu. La trace $\mathsf{tr} = \mathsf{in}(c,\mathsf{pub}(a)).\mathsf{out}(c,\mathsf{w})$ est une trace de non-inclusion car la recette $R = \mathsf{adec}(\mathsf{w},a)$ se réduit comme un message du côté de P mais pas du côté de Q.

Si l'on définit le système de types structuré où $\delta(x) = \tau$ (avec τ un type initial), alors P lui est conforme, car il n'existe qu'un seul sous-terme chiffré dans P. Cependant, il n'existe pas de témoin bien typé. En effet, on ne peut pas avoir $x\sigma = \mathsf{pub}(t)$ pour une substitution σ bien typée, puisque

$$\delta(x) = \delta(x\sigma) = \delta(\mathsf{pub}(t)) = \mathsf{pub}(\delta(t))$$

et $\mathsf{pub}(\delta(t)) \neq \tau$ car τ est un type initial. Le terme $\mathsf{aenc}(\langle a,n\rangle, x\sigma)$ ne peut être construit par aucune autre recette car n est inconnu de l'attaquant, et il ne peut pas être ouvert car $x\sigma$ n'est pas une clé. Donc, ce terme est indistinguable de $\mathsf{hash}(n)$.

2.6.2 Propriété des sous-termes

On considère la théorie $\mathsf{des}(\mathsf{f}(x)) \to \mathsf{g}(x)$ où g est un symbole libre. Cette théorie ne satisfait pas la propriété des sous-termes, et contredit les hypothèses du théorème. Considérons les processus suivants.

$$\begin{array}{rcl} P & = & \mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{out}(c,\mathsf{f}(\langle n,x\rangle)).\mathsf{out}(c,\mathsf{g}(\langle n,y\rangle)) \\ Q & = & \mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{out}(c,\mathsf{f}(\langle n,x\rangle)).\mathsf{out}(c,\mathsf{g}(\langle m,y\rangle)) \end{array}$$

La trace $\mathsf{tr} = \mathsf{in}(c,a).\mathsf{in}(c,a).\mathsf{out}(c,\mathsf{w}_1).\mathsf{out}(c,\mathsf{w}_2)$ est une trace de non-inclusion de P dans Q, comme le montrent les recettes $R_1 = \mathsf{des}(\mathsf{w}_1)$, $R_2 = \mathsf{w}_2$ et le test $R_1 = R_2$.

Le processus P est conforme à tout système de types, y compris si l'on pose $\delta(x) \neq \delta(y)$, mais alors pour toute substitution bien typée σ, on a $x\sigma \neq y\sigma$. Comme l'égalité $R_1 = R_2$ est fausse du coté de P, il n'y a aucun moyen de comparer $\mathsf{g}(\langle n, y\sigma\rangle)$ à quoique ce soit que l'attaquant puisse construire, puisqu'il ne connaît pas n. Pour pouvoir tenir compte de ce type de théories dans le cadre de notre résultat, il faudrait étendre la notion de sous-termes. Ici, on aurait par exemple $St'(\mathsf{f}(t)) = St(\mathsf{g}(t)) \cup St(\mathsf{f}(t))$ où St' désigne l'ensemble des sous-termes étendus. De cette manière, la notion de protocole conforme serait plus restrictive et le processus P ne serait plus conforme.

2.6.3 Linéarité des t_i dans $\mathsf{des}(t_1, \ldots, t_n) \to t_0$

Considérons la règle $\mathsf{des}(\mathsf{f}(x,x,y)) \to \mathsf{ok}$ (le terme $\mathsf{f}(x,x,y)$ n'est pas linéaire), et les processus :

$$\begin{array}{rcl} P & = & \mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{out}(c,\mathsf{f}(x,y,m)) \\ Q & = & \mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{out}(c,\mathsf{f}(x,n,m)) \end{array}$$

Alors $\mathsf{tr} = \mathsf{in}(c,a).\mathsf{in}(c,a).\mathsf{out}(c,\mathsf{w})$ est une trace de non-inclusion. En effet, après l'exécution de tr, le test $\mathsf{des}(\mathsf{w})$ donne un message du côté P, mais pas du côté Q.

Comme il n'y a qu'un seul sous-terme chiffré $\mathsf{f}(x,y,m)$ dans P, P est conforme à tout système de types, et en particulier à celui où $\delta(x) = \tau$ et $\delta(y) = \tau'$ avec $\tau \neq \tau'$. Alors, pour toute substitution bien typée, $x\sigma \neq y\sigma$ donc $\mathsf{des}(\mathsf{f}(x,y,m))\sigma$ n'est pas un message, et il est clair que l'on ne peut pas construire $\mathsf{f}(x\sigma, y\sigma, m)$ pour le comparer avec $\mathsf{f}(x\sigma, n, m)$ puisque m est privé.

2.6.4 Une variable non-linéaire par règle, au plus.

On considère la règle $\mathsf{des}(\mathsf{f}(x,y),\mathsf{g}(x,y)) \to \mathsf{ok}$, avec des variables non-linéaires x et y, et les processus :

$$\begin{array}{rcl} P & = & \mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{out}(c,\mathsf{f}(x,n)).\mathsf{out}(c,\mathsf{g}(y,n)) \\ Q & = & \mathsf{in}(c,x).\mathsf{in}(c,y).\mathsf{out}(c,\mathsf{f}(x,n)).\mathsf{out}(c,\mathsf{g}(y,m)) \end{array}$$

Alors $\mathsf{tr} = \mathsf{in}(c,a).\mathsf{in}(c,a).\mathsf{out}(c,\mathsf{w}_1).\mathsf{out}(c,\mathsf{w}_2)$ est une trace de non-inclusion. En effet, la recette $R = \mathsf{des}(\mathsf{w}_1,\mathsf{w}_2)$ donne un message du côté P, mais pas du côté Q.

Il n'y a pas de sous-termes chiffrés unifiables dans P, donc P est conforme à tout système de types, y compris quand $\delta(x) \neq \delta(y)$. Mais alors, pour toute substitution bien typée, $x\sigma \neq y\sigma$ et donc R n'est plus un message du côté P. De plus, dans cet exemple, n et m sont secrets, donc ils ne peuvent pas être réutilisés pour reconstruire les messages de l'attaquant. Il n'y a donc pas de témoin bien typé de cette non-inclusion.

2.7 Autres résultats de typage

Les premiers résultats de typages démontrés d'une part par Heather, Lowe et Schneider [76] et d'autre part par Ramanujam et Suresh [91] portent exclusivement sur l'accessibilité, pour une signature fixée contenant les chiffrement symétrique et asymétrique. Surtout, ils supposent un schéma de marquage très contraignant (chaque champ comporte une marque indiquant son type). Le résultat d'Arapinis et Duflot [15] traite également des propriétés d'accessibilité et d'une signature fixée, mais assouplit le schéma de marques en proposant un critère statique semblable au nôtre. En reprenant la technique de preuve, Chrétien, Cortier et Delaune [51] restreignent la signature, mais proposent le premier résultat de typage pour l'équivalence. Tous ces résultats sont essentiellement subsumés, soit par les travaux d'Almousa, Mödersheim, Modesti et Viganò [11] (que Hess et Mödersheim [78] formalisent en Isabelle/HOL en corrigeant quelques erreurs), soit par ceux qui ont été présentés ici. En particulier, ces deux résultats sont les seuls à valoir pour une classe paramétrique de primitives. Le reste de cette section est consacré à leur comparaison.

Champ d'application et généralités. Tandis que le travail d'Almousa *et al.* porte exclusivement sur l'accessibilité, notre résultat est valable aussi pour l'équivalence. De notre côté, nous ne disposons pas de preuves certifiées par un outil comme Isabelle ou Coq.

Conformité. Tandis que notre hypothèse de conformité demande seulement que deux sous-termes chiffrés unifiables soit de même type, Almousa *et al.* exigent que tous les sous-termes unifiables, à l'exception des variables, aient le même type. Évidemment, une telle restriction est

plus forte, puisqu'elle s'applique également aux paires. Rien ne dit, cependant, qu'elle ne puisse pas être levée.

Contours. La notion de contours est remplacée, chez Almousa *et al.* , par l'*invariance par analyse*, qui a l'avantage d'être statique, tandis que nous forçons dynamiquement le respect des contours à travers la notion de message. Plus précisément, l'invariance par analyse énonce que, si un sous-terme t (qui n'est pas une variable) apparaît dans une émission du protocole, et s'il est possible de déduire de t l'ensemble T à condition de connaître K, alors, pour toute substitution δ, il est possible de déduire $T\delta$ de $t\delta$ à condition de connaître $K\delta$. En particulier, le $\mathsf{aenc}(a, b)$ sera accepté par un processus chez Almousa *et al.* alors que ce n'est pas un message dans notre modèle.

Théories. Les deux théories sont incomparables. D'abord, des règles comme celles de check et de samekey (voir section 1.1.7) ne passent pas chez Almousa *et al.* car il n'y a pas de règle de la forme $\cdots \to \mathsf{ok}$ avec ok une constante. Cependant, cette absence n'est pas pénalisante puisque ce genre de règle est inutile dans le cas de l'accessibilité (elles ne servent que pour l'équivalence statique).

De plus, Almousa *et al.* ne peuvent pas tenir compte de règles réduisant plusieurs messages (par exemple $\mathsf{des}(\mathsf{f}(x), \mathsf{g}(x)) \to x$ où $\mathsf{f}(x)$ et $\mathsf{g}(x)$ sont tous deux réduits), ni de règles où des sous-termes profonds sont déductibles, comme $\mathsf{des}(\mathsf{f}(\mathsf{g}(x))) \to x$. À noter que dans notre cas, cette règle impose que le contour de f soit $\mathsf{f}(\mathsf{g}(x))$. D'un autre côté, Almousa *et al.* peuvent modéliser certaines règles contenant plusieurs variables non linéaires, telles que $\mathsf{des}(\mathsf{f}(x, y, z), y, z) \to x$.

Cette comparaison laisse penser qu'il existe un modèle plus général, qui permettrait de subsumer les deux résultats.

2.8 Conclusion

Dans ce chapitre, deux résultats de typage, l'un pour l'accessibilité, et l'autre pour l'équivalence, ont été présentés. Ces résultats démontrent qu'il existe une attaque si, et seulement si, il existe une attaque bien typée. Nous utiliserons exclusivement le théorème 2.2, qui porte sur l'équivalence, mais le théorème 2.1 sur l'accessibilité est d'un intérêt indépendant. Il ne serait probablement pas difficile d'améliorer ce résultat en enrichissant l'algèbre de processus, par exemple avec des événements ou des états, pour modéliser des propriétés de correspondance, comme l'authentification. Ces extensions ont été laissées de côté pour se focaliser sur l'équivalence. Par ailleurs, Chrétien, Cortier et Delaune [53] ont déterminé une classe de protocoles pour laquelle l'équivalence est décidable (pour un nombre arbitraire de sessions) en s'appuyant sur le résultat [51] que nous étendons. Il semble possible de démontrer que ces protocoles restent décidables en présence de primitives asymétriques.

Il serait également intéressant d'unifier les preuves des propositions 2.1 et 2.2, qui sont assez proches, afin de simplifier toute extension future. En effet, le cas de la réception d'un message dans la démonstration de la proposition 2.1 ressemble à la preuve de l'équivalence statique dans la proposition 2.2. Une simplification aiderait pour toute autre extension. Par exemple, il serait possible d'ajouter un opérateur de déchiffrement interne (let), et ainsi d'écrire $\mathsf{in}(c, x).=\mathsf{sdec}(x, k)$ au lieu de $\mathsf{in}(c, \mathsf{senc}(y, k))$, sachant que ces deux processus ne sont pas strictement identiques (dans le premier cas, la réception ne peut jamais échouer). La première difficulté consisterait à exprimer une hypothèse de conformité relative à cette algèbre de processus. Il faudrait également tenir compte d'un plus grand nombre d'actions silencieuses. De même, les contre-exemples de la section 2.6 sont plutôt des obstacles à surmonter que des limites théoriques : nous avons par exemple donné une piste pour se passer de la propriété des sous-termes avec une notion de sous-termes étendus.

Une autre extension intéressante consisterait à tirer parti des propositions 2.1 et 2.2 y compris pour des protocoles ne respectant aucune hypothèse de conformité. Il est possible de reformuler la proposition 2.2 : pour décider de l'équivalence de traces du protocoles, il suffit d'énumérer toutes les traces obtenues par unification de sous-ensembles de sous-termes chiffrés. De cette manière, nous obtiendrions un nombre fini de traces à examiner (dans le cas de processus sans réplication qui nous occupe) et donc une procédure de décision.

À l'issue de ce chapitre, nous avons réduit l'espace de recherche aux traces bien typées, mais l'ensemble des constantes disponibles pour l'attaquant reste infini. Le chapitre suivant permettra de limiter ces constantes à un nombre fini et petit, ce qui sera important pour l'efficacité de la procédure de décision par codage. Enfin, le chapitre 6 présentera, entre autres, une liste de protocoles (voir sections 6.2 et 6.3) et étudiera la possibilité de les rendre conformes à un système de types structuré, au besoin en marquant les messages, c'est-à-dire en ajoutant quelques bits permettant de lever la confusion entre les messages.

3 Une borne sur le nombre de constantes

Dans le chapitre précédent, nous avons restreint l'espace de recherche aux attaques bien typées. Cependant, nous avons dû supposer que l'attaquant disposait d'une infinité de constantes de chaque type. Comme nous nous intéressons plus particulièrement aux protocoles conformes à des systèmes de types structurés, il est facile de voir que l'on peut se contenter de fournir un nombre fini de constantes à l'attaquant, simplement parce que la structure des termes est connue, et que nous supposons que le nombre de sessions est borné ; mais la borne calculée de cette manière serait bien trop importante, et croissante avec le nombre de sessions considérées. Ce chapitre se consacre à établir une borne beaucoup plus petite de trois constantes (en plus de celles qui apparaissent explicitement dans le protocole), indépendante à la fois du protocole et du nombre de sessions.

Des résultats similaires ont été démontrés. D'une part, dans le cas des propriétés d'accessibilité, le nombre d'agents à considérer peut être réduit à un agent honnête et un agent malhonnête [58]. D'autre part, pour les propriétés d'équivalence, il est correct d'abstraire les nonces, à condition de garder au moins deux copies de chacun d'entre eux [52]. Nous nous sommes inspirés de ces travaux pour démontrer que deux protocoles sont équivalents en présence d'un nombre arbitraire d'agents si, et seulement si, ils sont en équivalence en présence de deux agents honnêtes et de deux agents malhonnêtes [60]. De plus, nous donnons également des contre-exemples qui garantissent que chacune de nos hypothèses est nécessaire.

Ce chapitre porte sur les constantes de l'attaquant, plutôt que sur les agents ou les nonces. Dans un modèle non typé et sans sorte, le résultat est évident, puisque l'attaquant peut remplacer les constantes $c_0, c_1, \ldots, c_n$ par exemple par $c_0, \mathsf{hash}(c_0), \ldots, \mathsf{hash}(\ldots, \mathsf{hash}(c_0)\ldots)$. Nous adaptons donc le raisonnement présenté dans [60] pour démontrer que nous pouvons borner les constantes dans un modèle typé. Cependant, la méthode précédente, appliquée directement, implique que l'attaquant doit disposer de deux constantes de chaque type. Pour contourner ce problème, nous exposons la notion de trace quasiment typée dans la section 3.1, ce qui permet de conserver les avantages du typage tout en autorisant l'attaquant à utiliser la même constante pour instancier des variables de différents types. La section 3.2 présente le raisonnement pour parvenir au théorème principal de ce chapitre. Les preuves formelles, laissées de côté, sont fournies dans la section 3.3. Nous montrons la pertinence de nos hypothèses dans la section 3.4, en adaptant nos contre-exemples pour le nombre d'agents au cas des constantes.

Dans tout ce chapitre, nous conservons le modèle des théories quasi-linéaires (voir section 2.1). En particulier, il n'existe qu'une seule règle qui réduit chaque destructeur.

3.1 Traces quasiment typées

Dans le chapitre précédent, nous avons démontré que $P \not\sqsubseteq_{at} Q$ si, et seulement si, il existe une trace bien typée de P qui est un témoin de non-inclusion. Dans ce cas, comme l'attaquant respecte le système de types et comme il doit nécessairement être capable d'instancier chaque variable, il doit disposer au moins d'autant de constantes qu'il existe de types de variables différents dans le protocole. Or, nous voulons borner les constantes de l'attaquant indépendamment du protocole. Nous devons donc relâcher les contraintes de typage pour permettre à l'attaquant d'utiliser la même constante pour instancier des variables de types différents. De cette manière, nous conservons les avantages du typage, et en particulier nous pouvons toujours connaître la forme des termes attendus pour chaque variable, tout en réduisant le nombre de constantes considérées.

Étant donné un système de types structuré $(\mathcal{T}, \delta)$, nous notons $\mathcal{T}^\star = \mathcal{T} \uplus \{\tau^\star\}$ où $\tau^\star$ est un élément particulier ($\tau^\star \notin \mathcal{T}$). Nous définissons également une relation d'ordre $\preccurlyeq$ sur l'ensemble des types de la manière suivante.

— $\tau^\star \preccurlyeq \tau$ et $\tau \preccurlyeq \tau$ pour tout type τ (initial ou non).

— $\mathsf{f}(\tau_1, \ldots, \tau_k) \preccurlyeq \mathsf{f}(\tau'_1, \ldots, \tau'_k)$ quand $\tau_i \preccurlyeq \tau'_i$ pour chaque i et $\mathsf{f} \in \Sigma_c$.

Nous disons qu'une substitution σ est *quasiment typée* par rapport à $(\mathcal{T}, \delta)$ si pour toute variable $x \in dom(\sigma)$, $\delta(x\sigma) \preccurlyeq \delta(x)$. Informellement, la relation $\tau \preccurlyeq \tau'$ signifie donc qu'un élément de type τ peut instancier une variable de type τ'. Cette notion s'étend naturellement à une exécution : $\mathcal{P} \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi; \sigma; i)$ est *quasiment typée* par rapport à $(\mathcal{T}, \delta)$ si la substitution sous-jacente σ est quasiment typée par rapport à $(\mathcal{T}, \delta)$; et dans ce cas, nous disons également que la trace tr est une trace quasiment typée de $\mathcal{P}$, par rapport à $(\mathcal{T}, \delta)$. En l'absence d'ambiguïté, nous omettrons de préciser le système de types structuré par rapport auquel une trace ou une substitution sont quasiment typées. Nous supposons également l'existence de trois constantes spéciales $c_0^\star, c_1^\star \in \Sigma_{\mathsf{fresh}}^{\mathsf{bitstring}}$ et $c_+^\star \in \Sigma_{\mathsf{fresh}}^{\mathsf{atom}}$, toutes trois de type $\tau^\star$. Ces constantes peuvent donc être assignées à des variables de tous les types.

Le théorème principal de ce chapitre affirme qu'il existe un témoin de non-inclusion si, et seulement si, il existe un témoin quasiment typé contenant au plus les constantes $c_0^\star, c_1^\star$ et $c_+^\star$, en plus de celles qui apparaissent explicitement dans le protocole, et de plus toutes les recettes de la trace sont simples. Soient $\mathcal{K}_\mathcal{P}$ et $\mathcal{K}_\mathcal{Q}$ deux Σ_0^--configurations. Nous notons $\Sigma_\mathcal{P} \subset \Sigma_0^-$ (resp. $\Sigma_\mathcal{Q} \subset \Sigma_0^-$) l'ensemble fini des constantes qui apparaissent dans $\mathcal{K}_\mathcal{P}$ (resp. $\mathcal{K}_\mathcal{Q}$) ou dans les règles de réécriture. Autrement dit, $\Sigma_\mathcal{P}$ est le plus petit ensemble contenant les constantes apparaissant dans les règles de réécriture et tel que $\mathcal{K}_\mathcal{P}$ soit une $\Sigma_\mathcal{P}$-configuration.

Théorème 3.1. *Soit $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$ un système de types structuré, et $\mathcal{K}_\mathcal{P}$ une Σ_0^--configuration déterministe, initiale et conforme au système de types structuré $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$. Soit $\mathcal{K}_\mathcal{Q}$ une autre Σ_0-configuration déterministe. Posons $\Sigma_0^\star = (\Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q}) \uplus \{c_0^\star, c_1^\star, c_+^\star\}$. Nous avons $\mathcal{K}_\mathcal{P} \sqsubseteq_{at} \mathcal{K}_\mathcal{Q}$ par rapport à Σ_0^- si, et seulement si, il existe un témoin quasiment typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{K}_\mathcal{P})$ de cette non-inclusion, où toutes les recettes de tr sont simples.*

L'existence d'un témoin quasiment typé découle directement du chapitre précédent, puisqu'un témoin bien typé est quasiment typé. Le reste de ce chapitre est dédié à la démonstration des autres éléments de ce théorème, à savoir qu'il est possible de se restreindre à trois constantes et à des recettes simples. Illustrons le résultat sur l'exemple de référence.

Exemple 3.1. *Reprenons tr'' et ϕ'' comme à l'exemple 2.8, avec $(\mathsf{tr}'', \phi'') \in \mathsf{trace}_{\Sigma_0^+}(P'_{OR})$ (où le processus P'_{OR} a été défini à l'exemple 1.16). La Σ_0^+-trame ψ'' est obtenue en remplaçant $\mathsf{w}_4 \triangleright k_{as}$ par $\mathsf{w}_4 \triangleright k$ dans ϕ''. Si nous considérons la trace $\mathsf{tr}^\star = \mathsf{tr}''\{c_{fwd} \triangleright c_0^\star\}$, nous obtenons $(\mathsf{tr}^\star, \phi^\star) \in \mathsf{trace}_{\Sigma_0^\star}(P'_{OR})$, avec $\phi^\star = \phi''\{c_{fwd} \triangleright c_0^\star\}$. De même, nous pouvons définir $\psi^\star = \psi''\{c_{fwd} \triangleright c_0^\star\}$ et nous avons $(\mathsf{tr}^\star, \psi^\star) \in \mathsf{trace}_{\Sigma_0^\star}(P''_{OR})$ où P''_{OR} a été défini à l'exemple 1.16. Si nous considérons la recette $R = \mathsf{sdec}(\mathsf{proj}_4^4(\mathsf{w}_1), \mathsf{w}_4)$, nous avons toujours que $R\phi^\star\!\downarrow$ est un message, tandis que $R\psi^\star\!\downarrow$ n'est pas un message.*

Cet exemple peut sembler trivial, puisqu'une seule constante est remplacée, mais il résume le principe général, qui est de projeter toutes les constantes qui ne sont pas utiles dans le test final sur la constante $c_0^\star$.

3.2 Trois constantes suffisent

Cette section expose le raisonnement qui permet de prouver le théorème 3.1. Les preuves formelles, qui sont laissées de côté, se trouvent dans la section 3.3. Pour commencer, nous démontrons un corollaire du théorème 2.2 : il est possible de considérer une exécution bien typée qui ne contient que des recettes simples.

Corollaire 3.1. *Soient $\mathcal{K}_\mathcal{P}$ une Σ_0^--configuration conforme au système de types $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$ et $\mathcal{K}_\mathcal{Q}$ une Σ_0^--configuration. Nous avons $\mathcal{K}_\mathcal{P} \not\sqsubseteq_{at} \mathcal{K}_\mathcal{Q}$ par rapport à Σ_0^- si, et seulement si, il existe un témoin bien typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_\mathcal{P})$ de non-inclusion qui utilise seulement des recettes simples.*

Démonstration. Ce corollaire est une conséquence directe du théorème 2.2. Il reste à montrer que l'on peut choisir des recettes simples pour la trace tr. Puisque l'exécution et le typage sont déterminés exclusivement par la trace concrète $\mathsf{tr}\phi\!\downarrow$ (avec ϕ la trame associée à tr), il est clair que nous pouvons remplacer toute recette R de tr par n'importe quelle recette R' telle que $R\phi\!\downarrow = R'\phi\!\downarrow$. De plus, $R\phi\!\downarrow$ est un message, donc $R\!\downarrow\phi\!\downarrow = R\phi\!\downarrow$ d'après le lemme 2.2. $R\!\downarrow$ est simple car $R\!\downarrow\phi\!\downarrow$ est un message, d'après le lemme 2.3. Donc en remplaçant chaque recette R de tr par sa forme normale forcée $R\!\downarrow$, nous obtenons un témoin tr' bien typé et qui ne contient que des recettes simples. □

Il s'agit maintenant de démontrer que l'attaquant n'a besoin que de trois constantes. Nous allons procéder en renommant les constantes superflues. Plus formellement, étant donné un ensemble $\mathsf{A} \subseteq \Sigma_0^+$, un A-*renommage* est une fonction ρ telle que $dom(\rho) \cup img(\rho) \subseteq A$; $\rho(a)$ est de sorte atom si a est de sorte atom ; et si c est une constante apparaissant dans une règle de réécriture, alors $c \notin dom(\rho) \cup img(\rho)$ (même si $c \in \mathsf{A}$). Étant donné un système de types structuré $(\mathcal{T}, \delta)$, nous disons que le renommage ρ est *typé* si $\delta(\rho(a)) = \delta(a)$ pour tout $a \in dom(\rho)$.

Le lemme suivant démontre d'abord que lorsque deux termes se réduisent comme des messages, et sont égaux, alors aucun renommage ne peut modifier cette situation. Ensuite, lorsqu'un terme t ne se réduit pas comme un message, il existe une constante c_0 telle que $t\rho\!\downarrow$ ne sera pas un message, à moins que le renommage ρ ne modifie la constante c_0. De cette manière, nous savons que lorsque nous traitons le cas de non-inclusion statique où un certain terme n'est pas un message, il est possible de renommer toutes les constantes à part c_0, ce qui revient à dire que nous pouvons nous contenter d'utiliser deux constantes (il faudra en considérer une de plus pour préserver la sorte). Enfin, ce résultat est également valable pour une inégalité $t \neq t'$, c'est-à-dire qu'il existe une constante c_0 telle que $t\rho \neq t'\rho$ pour tout renommage ρ qui ne touche pas à c_0.

Lemme 3.1. *Soient t et t' deux termes dans $\mathcal{T}(\Sigma, \Sigma_0^+ \uplus \mathcal{N})$.*

1. *Si $t\!\downarrow$ est un Σ_0^+-message alors $(t\!\downarrow)\rho = (t\rho)\!\downarrow$ pour chaque Σ_0^+-renommage ρ.*
2. *Si $t\!\downarrow$ n'est pas un Σ_0^+-message, alors il existe une constante $c_0 \in \Sigma_0^+$ telle que pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$, le terme $t\rho\!\downarrow$ n'est pas un Σ_0^+-message.*
3. *Si $t\!\downarrow$ et $t'\!\downarrow$ sont des Σ_0^+-messages et $t\!\downarrow \neq t'\!\downarrow$, alors il existe une constante $c_0 \in \Sigma_0^+$ telle que pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$, nous avons $t\rho\!\downarrow \neq t'\rho\!\downarrow$.*

Illustrons ce résultat sur un exemple.

Exemple 3.2. *Donnons un exemple simple pour chaque point.*

1. *Soient $t = \mathsf{sdec}(\mathsf{senc}(a, k), k)$ et ρ un renommage, avec a et k des constantes atomiques. Si $k \notin dom(\rho)$, alors $t\rho = \mathsf{sdec}(\mathsf{senc}(a\rho, k), k)$ et donc $t\rho\!\downarrow = a\rho = t\!\downarrow\rho$; si $k \in dom(\rho)$, alors $t\rho = \mathsf{sdec}(\mathsf{senc}(a\rho, k\rho), k\rho)$, et comme ρ remplace les atomes par des atomes, $k\rho$ est un atome. Donc $t\rho\!\downarrow = a\rho = t\!\downarrow\rho$.*

2. *Soient* $t' = \mathsf{sdec}(\mathsf{senc}(a,k),k')$ *et* ρ *un renommage, avec* a, k *et* k' *des constantes.* $t'\downarrow = t'$ *n'est pas un message. Si* $k\rho = k'\rho$*, alors* $t'\rho\downarrow = a\rho$ *est un message. Cependant, pour tout renommage* ρ *tel que* $k \notin img(\rho) \cup dom(\rho)$*, nous avons* $k\rho = k \neq k'\rho$*, et donc* $t'\rho\downarrow = t'\rho$ *n'est pas un message.*
3. *Soient* a *et* b *deux constantes, et* ρ *un renommage. Pour que* $a\rho \neq b\rho$*, il suffit que* $a \notin dom(\rho) \cup img(\rho)$*.*

Il s'agit maintenant de démontrer que nous pouvons renommer des constantes sans perturber l'exécution : l'exécution après renommage suit l'exécution avant renommage. Ce résultat repose sur le fait que les constantes que nous remplaçons n'apparaissent pas dans le protocole au début de l'exécution. De plus, créer plus d'égalités entre constantes ne peut jamais empêcher un protocole de s'exécuter, mais seulement autoriser des exécutions supplémentaires.

Lemme 3.2. *Soient* ρ *un* A*-renommage typé avec* $\mathsf{A} \subseteq \Sigma_0^+ \setminus \Sigma_{\mathcal{P}}$*,* $\mathcal{K}_{\mathcal{P}}$ *une* Σ_0^+*-configuration conforme au système de types structuré* $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$*, et* $\mathcal{K}_{\mathcal{P}} \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi'; \sigma'; i')$ *une exécution quasiment typée. L'exécution* $\mathcal{K}_{\mathcal{P}} \xrightarrow{\mathsf{tr}\rho} (\mathcal{P}'; \phi'\rho; \sigma'\rho; i')$ *est quasiment typée.*

Enfin, nous en arrivons au résultat principal.

Théorème 3.1. *Soit* $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$ *un système de types structuré, et* $\mathcal{K}_{\mathcal{P}}$ *une* Σ_0^-*-configuration déterministe, initiale et conforme au système de types structuré* $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$*. Soit* $\mathcal{K}_{\mathcal{Q}}$ *une autre* Σ_0*-configuration déterministe. Posons* $\Sigma_0^\star = (\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}) \uplus \{c_0^\star, c_1^\star, c_+^\star\}$*. Nous avons* $\mathcal{K}_{\mathcal{P}} \sqsubseteq_{at} \mathcal{K}_{\mathcal{Q}}$ *par rapport à* Σ_0^- *si, et seulement si, il existe un témoin quasiment typé* $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{K}_{\mathcal{P}})$ *de cette non-inclusion, où toutes les recettes de* tr *sont simples.*

La réciproque découle du lemme 2.15. Concentrons-nous sur le sens direct. Nous commençons par appliquer le corollaire 3.1, pour obtenir un témoin bien typé de non-inclusion. Il existe alors deux cas principaux :

1. le témoin s'exécute dans $\mathcal{K}_{\mathcal{P}}$ mais pas dans $\mathcal{K}_{\mathcal{Q}}$;
2. le témoin s'exécute dans $\mathcal{K}_{\mathcal{P}}$ et dans $\mathcal{K}_{\mathcal{Q}}$, mais il existe un test d'équivalence statique qui passe dans $\mathcal{K}_{\mathcal{P}}$ mais pas dans $\mathcal{K}_{\mathcal{Q}}$.

Schéma de preuve. Dans le premier cas, considérons que $\mathsf{tr}.\mathsf{in}(c, R)$ s'exécute dans $\mathcal{K}_{\mathcal{P}}$ mais pas dans $\mathcal{K}_{\mathcal{Q}}$. Il existe un motif u, un processus P et une substitution σ tels que $\mathsf{in}(c,u).P \in \mathcal{K}_{\mathcal{P}}$ et $u\sigma = R\phi\downarrow$, avec ϕ la trame associée au témoin dans $\mathcal{K}_{\mathcal{P}}$. Soit il n'existe pas de processus $\mathsf{in}(c,v).Q \in \mathcal{K}_{\mathcal{Q}}$ (et alors le résultat est évident) soit il en existe un, et alors v ne s'unifie pas avec $R\psi\downarrow$, où ψ est la trame associée au témoin dans $\mathcal{K}_{\mathcal{Q}}$. Dans ce cas, le point 1 du lemme 3.1 montre que pour tout renommage ρ, $R\phi\downarrow\rho = (R\rho)(\phi\rho)\downarrow$. Le point 3 du lemme 3.1 donne une constante c_0 telle que, pour tout renommage qui ne touche pas c_0, $R\psi\downarrow\rho$ ne s'unifie pas avec v. Nous définissons le renommage ρ qui envoie toutes les constantes sur $c_0^\star$, sauf c_0 qui est envoyée sur $c_1^\star$ ou $c_+^\star$ suivant que c_0 est de sorte atom ou $\mathsf{bitstring}$. Le lemme 3.2 s'applique à ρ, donc il est possible de renommer le témoin et la réception ne passe toujours pas. Les cas de l'émission et du changement de phase sont plus simples.

Dans le second cas, considérons le témoin tr. Notons ϕ la trame résultante dans $\mathcal{K}_{\mathcal{P}}$, et ψ la trame résultante dans $\mathcal{K}_{\mathcal{Q}}$. Ou bien il existe une recette telle que $R\phi\downarrow$ est un message, mais $R\psi\downarrow$ n'est pas un message ; ou bien il existe deux recettes R_1 et R_2 telles que $R_1\phi\downarrow = R_2\phi\downarrow$ mais $R_1\psi\downarrow \neq R_2\psi\downarrow$. Le premier point du lemme 3.1 garantit que pour tout renommage ρ, $(R\rho)(\phi\rho)\downarrow$ est un message, et $(R_1\phi)\rho\downarrow = R_1\phi\downarrow\rho = R_2\phi\downarrow\rho = (R_2\phi)\rho\downarrow$. Le second point nous donne une constante c_0 telle que, pour tout renommage ρ qui ne touche pas à c_0, $(R\psi)\rho\downarrow$ n'est pas un message, ou (selon le cas) $(R_1\psi)\rho\downarrow \neq (R_2\psi)\rho\downarrow$. Nous considérons alors le renommage ρ qui envoie toutes les constantes sur $c_0^\star$, sauf c_0 qui est envoyée sur $c_1^\star$ ou $c_+^\star$ en fonction de sa sorte. Le lemme 3.2 s'applique à ρ, donc il est possible de répliquer le témoin et d'obtenir un test de non-inclusion statique.

3.3 Démonstrations

Cette section contient les preuves laissées de côté dans la section précédente. Commençons par démontrer le lemme suivant.

Lemme 3.1. *Soient t et t' deux termes dans $\mathcal{T}(\Sigma, \Sigma_0^+ \uplus \mathcal{N})$.*

1. *Si $t\!\downarrow$ est un Σ_0^+-message alors $(t\!\downarrow)\rho = (t\rho)\!\downarrow$ pour chaque Σ_0^+-renommage ρ.*
2. *Si $t\!\downarrow$ n'est pas un Σ_0^+-message, alors il existe une constante $c_0 \in \Sigma_0^+$ telle que pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$, le terme $t\rho\!\downarrow$ n'est pas un Σ_0^+-message.*
3. *Si $t\!\downarrow$ et $t'\!\downarrow$ sont des Σ_0^+-messages et $t\!\downarrow \neq t'\!\downarrow$, alors il existe une constante $c_0 \in \Sigma_0^+$ telle que pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$, nous avons $t\rho\!\downarrow \neq t'\rho\!\downarrow$.*

Démonstration. Nous démontrons chaque point séparément.

Premier point. Montrons que si $t\!\downarrow$ est un Σ_0^+-message alors $(t\!\downarrow)\rho = (t\rho)\!\downarrow$ pour chaque Σ_0^+-renommage ρ. Nous procédons par récurrence structurelle sur t. Si t est une donnée, alors $t\!\downarrow\rho = t\rho = t\rho\!\downarrow$, ce qui suffit à conclure. Supposons donc $t = \mathsf{f}(t_1, \ldots, t_n)$ pour un symbole f.

Cas où f est un constructeur. Si f est un symbole constructeur, alors :

$$\begin{aligned} t\rho\!\downarrow &= \mathsf{f}(t_1, \ldots, t_n)\rho\!\downarrow \\ &= \mathsf{f}(t_1\rho, \ldots, t_n\rho)\!\downarrow \\ &= \mathsf{f}(t_1\rho\!\downarrow, \ldots, t_n\rho\!\downarrow) \end{aligned}$$

Par hypothèse de récurrence $\mathsf{f}(t_1\rho\!\downarrow, \ldots, t_n\rho\!\downarrow) = \mathsf{f}(t_1\!\downarrow\rho, \ldots, t_n\!\downarrow\rho)$ et nous avons donc

$$\begin{aligned} t\rho\!\downarrow &= \mathsf{f}(t_1\rho\!\downarrow, \ldots, t_n\rho\!\downarrow) \\ &= \mathsf{f}(t_1\!\downarrow\rho, \ldots, t_n\!\downarrow\rho) \\ &= \mathsf{f}(t_1\!\downarrow, \ldots, t_n\!\downarrow)\rho \\ &= \mathsf{f}(t_1, \ldots, t_n)\!\downarrow\rho \\ &= t\!\downarrow\rho \end{aligned}$$

Cas où f est un destructeur. Si f est un symbole destructeur ($\mathsf{f} = \mathsf{des}$), alors il existe une unique règle $\mathsf{des}(t_1^{\mathsf{des}}, \ldots, t_n^{\mathsf{des}}) \to t_0^{\mathsf{des}}$ qui réduit des. Comme $t\!\downarrow$ est un Σ_0^+-message, il existe une substitution σ telle que pour chaque i, $t_i = t_i^{\mathsf{des}}\sigma$. Nous avons donc $t_i\rho = (t_i^{\mathsf{des}}\sigma)\rho = t_i^{\mathsf{des}}(\rho \circ \sigma)$. Comme ρ renomme les atomes par des atomes, $t = \mathsf{des}(t_1^{\mathsf{des}}\sigma, \ldots, t_n^{\mathsf{des}}\sigma)$ se réduit. Si t_0^{des} est un terme clos, alors, comme $c \notin dom(\rho) \cap img(\rho)$ pour toute constante c qui apparaît dans une règle de réécriture, nous avons $t_0^{\mathsf{des}} = t_0^{\mathsf{des}}\rho$. Donc $t\rho\!\downarrow = t_0^{\mathsf{des}} = t_0^{\mathsf{des}}\rho = t\!\downarrow\rho$. Si t_0^{des} n'est pas un terme clos, alors $\mathsf{des}(t_1^{\mathsf{des}}(\rho \circ \sigma), \ldots, t_2^{\mathsf{des}}(\rho \circ \sigma))$ se réécrit vers $t_0^{\mathsf{des}}(\rho \circ \sigma)$, car $c \notin dom(\rho) \cap img(\rho)$ pour toute constante c qui apparaît dans une règle de réécriture. De plus, nous avons également $t\!\downarrow = t_0^{\mathsf{des}}\sigma\!\downarrow$. Comme t_0^{des} est un sous-terme de t_1^{des}, $t_0^{\mathsf{des}}\sigma\!\downarrow$ est un sous-terme de $t_1^{\mathsf{des}}\sigma\!\downarrow = t_1\!\downarrow$. L'hypothèse de récurrence s'applique donc et nous avons $t_0^{\mathsf{des}}\sigma\!\downarrow\rho = t_0^{\mathsf{des}}(\rho\circ\sigma)\!\downarrow$, c'est-à-dire $t\!\downarrow\rho = t\rho\!\downarrow$, ce qui conclut la preuve du premier point.

Second point. Soit $t \in \mathcal{T}(\Sigma, \Sigma_0^+ \uplus \mathcal{N})$ un terme tel que $t\!\downarrow$ n'est pas un Σ_0^+-message. Dans le cas où $t \in \mathcal{T}(\Sigma_c, \Sigma_0^+ \uplus \mathcal{N})$, la seule possibilité pour que $t\rho\!\downarrow$ soit un message (tandis que $t\!\downarrow$ n'en est pas un) est qu'il existe une constante c de sorte $\mathsf{bitstring}$ en position de clé, et que $\rho(c)$ soit de sorte atom. Soient c_0 une telle constante, et ρ un Σ_0^+-renommage tel que $c_0 \notin dom(\rho) \cup img(\rho)$.

Nous avons que $t\rho\downarrow$ n'est pas un Σ_0^+-message puisque c_0 est en position de clé dans $t\rho$ et c_0 n'est pas de sorte atom.

Maintenant, t doit contenir au moins un symbole dans Σ_d. Soit p l'une des positions les plus basses telles que $t = C[\mathsf{g}(t_1, \ldots, t_k)]_p$ pour un certain $\mathsf{g} \in \Sigma_d$ et $t_1, \ldots, t_k \in \mathcal{T}(\Sigma_c, \Sigma_0^+ \uplus \mathcal{N})$. Soient $u = \mathsf{g}(t_1, \ldots, t_k)$, et ρ_0 un renommage particulier qui envoie toute constante dans Σ_0^+ vers $c_0 \in \Sigma_0^+$ (c'est-à-dire une constante arbitraire de sorte atom). Soit $\ell \to r$ la règle de réécriture associée à g. Ou bien la règle ne s'applique pas à $u\rho_0$, et alors elle ne s'applique à aucun $u\rho$ pour un Σ_0^+-renommage ρ ; ou bien, la règle peut s'appliquer à $u\rho_0$ alors qu'elle ne peut pas s'appliquer à u. Dans ce cas, il existe deux positions $p_1 \neq p_2$ telles que $\ell|_{p_1} = \ell|_{p_2} \in \mathcal{X}$, et donc $u|_{p_1} \neq u|_{p_2}$ tandis que $(u\rho_0)|_{p_1} = (u\rho_0)|_{p_2}$. De plus, nous savons que $u|_{p_1}$ et $u|_{p_2}$ sont toutes deux des constantes de Σ_0^+ (de sorte atom puisque sinon la réduction n'est pas possible). Soit $c_1 = u|_{p_1}$. Tout Σ_0^+-renommage ρ avec $c_1 \notin (dom(\rho) \cup img(\rho))$ empêchera la règle de réécriture $\ell \to r$ de s'appliquer à $u\rho$ et donc à $t\rho$. Ceci permet de conclure que $t\rho\downarrow$ n'est pas un Σ_0^+-message, ce qui prouve le second point.

Troisième point. Soit $t_1, t_2 \in \mathcal{T}(\Sigma, \Sigma_0^+ \uplus \mathcal{N})$ tels que $t_1\downarrow, t_2\downarrow$ sont des Σ_0^+-messages et $t_1\downarrow \neq t_2\downarrow$. D'après le premier point, nous avons $(t_1\downarrow)\rho = (t_1\rho)\downarrow$ et $(t_2\downarrow)\rho = (t_2\rho)\downarrow$. Donc, donc nous pouvons simplement montrer que si t_1, t_2 sont des Σ_0^+-messages et $t_1 \neq t_2$ alors il existe $c_0 \in \Sigma_0^+$ tel que $t_1\rho \neq t_2\rho$ pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$. Pour ce faire, nous procédons par récurrence sur le nombre de symboles apparaissant dans t_1 et t_2.

Initialisation. La seule situation non triviale correspond au cas où t_1 et t_2 sont toutes deux dans Σ_0^+. Supposons, sans perte de généralité, que $t_2 = c_0$. Puisque $t_1 \neq t_2$, nous avons $t_1\rho \neq t_2\rho$ pour chaque Σ_0^+-renommage ρ tel que $c_0 \notin dom(\rho) \cup img(\rho)$. Les autres cas, où t_1 ou t_2 est dans $\mathcal{N}$, sont triviaux puisque nous pouvons choisir n'importe quel Σ_0^+-renommage ρ.

Hérédité. Maintenant, dans le cas où t_1 et t_2 ne sont pas des données, nous distinguons deux cas. Ou bien ils n'ont pas le même symbole de fonction comme racine, et alors tout Σ_0^+-renommage ρ préserve l'inégalité $t_1\rho \neq t_2\rho$; ou bien ils ont le même symbole pour racine : supposons alors que $t_1 = \mathsf{f}(u_1, \ldots, u_k)$ et $t_2 = \mathsf{f}(v_1, \ldots, v_k)$ avec $\mathsf{f} \in \Sigma_c$. Nous savons que $u_i \neq v_i$ pour un certain $i \in \{1, \ldots, k\}$. L'hypothèse de récurrence s'applique, et nous en déduisons qu'il existe c_0 tel que $u_i\rho \neq v_i\rho$ pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$. Donc nous concluons en considérant n'importe quel Σ_0^+-renommage satisfaisant cette condition. □

Nous pouvons passer à la démonstration du lemme 3.2.

Lemme 3.2. *Soient ρ un A-renommage typé avec $\mathsf{A} \subseteq \Sigma_0^+ \setminus \Sigma_\mathcal{P}$, $\mathcal{K}_\mathcal{P}$ une Σ_0^+-configuration conforme au système de types structuré $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$, et $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}} (\mathcal{P}'; \phi'; \sigma'; i')$ une exécution quasiment typée. L'exécution $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}\rho} (\mathcal{P}'; \phi'\rho; \sigma'\rho; i')$ est quasiment typée.*

Démonstration. Soient $\mathcal{K}_\mathcal{P}$ une Σ_0^+-configuration conforme au système de types structuré $(\mathcal{T}, \delta)$, et $K'_\mathcal{P} = (\mathcal{P}'; \phi'; \sigma'; i')$. Nous montrons ce résultat par récurrence sur la longueur n de la trace d'exécution $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}} \mathcal{K}'_\mathcal{P}$.
Initialisation. Nous avons $\mathcal{K}'_\mathcal{P} = \mathcal{K}_\mathcal{P}$, et le résultat est trivialement vrai.
Hérédité. Nous avons $\mathsf{tr} = \mathsf{tr}^- \cdot \alpha$, et :

$$\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}^-} (\mathcal{P}^-; \phi^-; \sigma^-; i^-) \xrightarrow{\alpha} (\mathcal{P}'; \phi'; \sigma'; i').$$

D'après notre hypothèse de récurrence, nous savons que $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}^-} (\mathcal{P}^-\rho; \phi^-\rho; \sigma^-\rho; i^-)$. Nous distinguons trois cas suivant α.

— Cas $\alpha = \mathsf{phase}\ i'$. Dans ce cas, nous avons $\mathcal{P}' = \mathcal{P}^-$, $\sigma' = \sigma^-$, $\phi' = \phi^-$, et $i^- < i'$. De plus, $(\mathcal{P}^-; \phi^-\rho; \sigma^-\rho; i^-) \xrightarrow{\mathsf{phase}\ i'} (\mathcal{P}^-; \phi^-\rho; \sigma^-\rho; i')$, et nous pouvons conclure puisque

$$(\mathcal{P}^-; \phi^-\rho; \sigma^-\rho; i') = (\mathcal{P}'; \phi'\rho; \sigma'\rho; i')$$

— Cas $\alpha = \mathsf{out}(c, \mathsf{w})$. Dans ce cas $\mathcal{P}^- = \{i^-{:}\mathsf{out}(c, u).P_0\} \cup \mathcal{P}_0$ pour un certain u, P_0, et $\mathcal{P}_0$. Nous avons également $\sigma' = \sigma^-$, $\phi' = \phi^- \cup \{\mathsf{w} \triangleright u\sigma^-\}$ et $i' = i^-$. Pour conclure que

$$(\mathcal{P}^-; \phi^-\rho; \sigma^-\rho; i^-) \xrightarrow{\mathsf{out}(c,\mathsf{w})} (\mathcal{P}'; \phi'\rho; \sigma'\rho; i')$$

il suffit de montrer que $(u\sigma^-)\rho = u(\sigma^-\rho)$. En fait, $(u\sigma^-)\rho = (u\rho)(\sigma^-\rho)$, et comme $dom(\rho) \subseteq \Sigma_0^+ \setminus \Sigma_{\mathcal{P}}$, nous déduisons $u\rho = u$, et comme $\rho(a)$ est un atome si a est un atome, nous pouvons conclure.

— Cas $\alpha = \mathsf{in}(c, R)$. Dans ce cas $\mathcal{P}^- = \{i^-{:}\mathsf{in}(c, u).P_0\} \cup \mathcal{P}_0$ pour certains u, P_0, et $\mathcal{P}_0$. Nous avons aussi $\phi' = \phi^-$, $\sigma' = \sigma^- \uplus \sigma_0$ pour une certaine substitution σ_0 telle que $R\phi^-\!\downarrow = (u\sigma^-)\sigma_0$ et $R\phi^-\!\downarrow$ est un message. De plus, $i^- = i'$. Pour conclure que $(\mathcal{P}^-; \phi^-\rho; \sigma^-\rho; i^-) \xrightarrow{\mathsf{in}(c,R\rho)} (\mathcal{P}'; \phi'\rho; \sigma'\rho; i')$, il reste à montrer que $(R\rho)(\phi^-\rho)\!\downarrow = u(\sigma^-\rho)\sigma_0'$ et $\sigma'\rho = \sigma^-\rho \uplus \sigma_0'$ pour une certaine substitution σ_0'. Comme $R\phi^-\!\downarrow = (u\sigma^-)\sigma_0$, nous déduisons $(R\phi^-\!\downarrow)\rho = ((u\sigma^-)\sigma_0)\rho$, et grâce au lemme 3.1 (premier point), nous avons $(R\phi^-)\rho\!\downarrow = ((u\rho)(\sigma^-\rho))(\sigma_0\rho)$. Enfin, puisque ρ est un A-renommage et $\mathsf{A} \subseteq (\Sigma_0^- \uplus \{c_0^\star, c_1^\star, c_+^\star\}) \setminus \Sigma_{\mathcal{P}}$, nous savons que $u\rho = u$, et donc nous avons que $(R\rho)(\phi^-\rho)\!\downarrow = (u(\sigma^-\rho))(\sigma_0\rho)$. De plus, comme $\sigma' = \sigma^- \uplus \sigma_0$, nous avons $\sigma'\rho = \sigma^-\rho \uplus \sigma_0\rho$. Ainsi, nous concluons en choisissant $\sigma_0' = \sigma_0\rho$.

Comme le renommage ρ est typé, l'exécution résultante est quasiment typée lorsque l'exécution d'origine est quasiment typée. □

Nous concluons par la démonstration du théorème 3.1.

Théorème 3.1. *Soit $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$ un système de types structuré, et $\mathcal{K}_{\mathcal{P}}$ une Σ_0^--configuration déterministe, initiale et conforme au système de types structuré $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$. Soit $\mathcal{K}_{\mathcal{Q}}$ une autre Σ_0-configuration déterministe. Posons $\Sigma_0^\star = (\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}) \uplus \{c_0^\star, c_1^\star, c_+^\star\}$. Nous avons $\mathcal{K}_{\mathcal{P}} \sqsubseteq_{at} \mathcal{K}_{\mathcal{Q}}$ par rapport à Σ_0^- si, et seulement si, il existe un témoin quasiment typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{K}_{\mathcal{P}})$ de cette non-inclusion, où toutes les recettes de tr sont simples.*

Démonstration. La direction ($\Leftarrow$) est facile à établir. Il s'agit d'une conséquence directe du lemme 2.15. Nous considérons donc l'autre direction ($\Rightarrow$).

Pour commencer, nous appliquons le corollaire 3.1 pour obtenir un témoin quasiment typé $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{K}_{\mathcal{P}})$ de cette non-inclusion qui utilise seulement des recettes simples.

En considérant $\Sigma^{\star\star} = (\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}) \uplus \{c \in \Sigma_0^+ \mid \delta_{\mathcal{P}}(c) = \tau^\star\}$ (supposé infini), et en appliquant un alpha-renommage sur la dérivation pour renommer les constantes de $\Sigma_0^+ \setminus \Sigma^{\star\star}$ vers des constantes de $\Sigma^{\star\star}$, nous obtenons facilement un témoin bien typé dont toutes les recettes sont simples, construit en utilisant des constantes de $\Sigma^{\star\star}$.

Nous considérons un tel témoin de non-inclusion minimal (en longueur), c'est-à-dire une trace $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma^{\star\star}}(\mathcal{K}_{\mathcal{P}})$ telle que :

1. ou bien $(\mathsf{tr}, \psi) \notin \mathsf{trace}_{\Sigma^{\star\star}}(\mathcal{K}_{\mathcal{Q}})$ pour tout ψ ;
2. ou bien $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma^{\star\star}}(\mathcal{K}_{\mathcal{Q}})$ mais $\phi \not\sqsubseteq_s \psi$.

Nous allons appliquer un renommage sur ce témoin pour supprimer les constantes de $\Sigma^{\star\star} \setminus (\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}})$.

Ce renommage ρ devra vérifier que $dom(\rho) \subseteq \Sigma^{\star\star} \setminus (\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}})$, et $img(\rho) \subseteq \{c_0^\star, c_1^\star, c_+^\star\}$. De plus, si c est de sorte atom, alors $\rho(c)$ sera de sorte atom.

Dans la suite, nous aurons à considérer différents renommages, en particulier, ρ_0 qui envoie toute constante de $\Sigma^{\star\star}$ vers $c_0^\star$, ainsi que ρ_1 qui envoie toute constante de $\Sigma^{\star\star}$ de sorte atom vers $c_0^\star$, et toute constante de $\Sigma^{\star\star}$ de sorte $\mathsf{bitstring}$ vers $c_+^\star$.

Cas 1 : $\mathsf{tr} = \mathsf{tr}^- \cdot \alpha$ ne passe pas dans $\mathcal{K}_\mathcal{Q}$. Dans ce cas, nous avons alors :

— $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}^-} (\mathcal{P}^-; \phi^-; \sigma_\mathcal{P}^-; i^-) \xrightarrow{\alpha} (\mathcal{P}; \phi; \sigma_\mathcal{P}; i)$;

— $\mathcal{K}_\mathcal{Q} \xrightarrow{\mathsf{tr}^-} (\mathcal{Q}^-; \psi^-; \sigma_\mathcal{Q}^-; i^-)$;

— $\phi^- \sqsubseteq_s \psi^-$.

En nous appuyant sur le lemme 3.2, nous avons :

— $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}^-\rho} (\mathcal{P}^-; \phi^-\rho; \sigma_\mathcal{P}^-\rho; i^-) \xrightarrow{\alpha\rho} (\mathcal{P}; \phi\rho; \sigma_\mathcal{P}\rho; i)$;

— $\mathcal{K}_\mathcal{Q} \xrightarrow{\mathsf{tr}^-\rho} (\mathcal{Q}^-; \psi^-\rho; \sigma_\mathcal{Q}^-\rho; i^-)$

pour chaque A-renommage ρ tel que $\mathsf{A} \subseteq \Sigma_0^+ \setminus (\Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q})$. Donc, pour conclure, il reste à justifier que $\alpha\rho$ ne peut pas être exécutée à partir de $(\mathcal{Q}^-; \psi^-\rho; \sigma_\mathcal{Q}^-; i^-)$ pour un Σ_0^+-renommage ρ tel que $img(\rho) \subseteq \{c_0^\star, c_1^\star, c_+^\star\}$. Nous considérons trois cas suivant l'action α.

1. *Cas* $\alpha = \mathsf{phase}\ i$. Dans ce cas, l'instruction $\xrightarrow{\mathsf{phase}\ i}$ ne s'exécute pas dans $(\mathcal{Q}^-; \psi^-; \sigma_\mathcal{Q}^-; i^-)$, ce qui est impossible puisque α peut être exécuté à partir de $(\mathcal{P}^-; \phi^-; \sigma_\mathcal{P}^-; i^-)$, et donc $i^- < i$.
2. *Cas* $\alpha = \mathsf{out}(c, \mathsf{w})$. Dans ce cas, nous avons que $\mathcal{P}^- = \mathsf{out}(c, u).P_0 \uplus \mathcal{P}_0$. Ou bien $\mathcal{Q}^-$ ne peut pas exécuter une émission sur le canal c, et alors c'est le cas pour $(\mathcal{Q}^-; \psi^-\rho_0; \sigma_\mathcal{Q}^-\rho_0; i^-)$, ce qui prouve le résultat ; ou bien nous avons $\mathcal{Q}^- = \mathsf{out}(c, v).Q_0 \uplus \mathcal{Q}_0$ mais $v\sigma_\mathcal{Q}^-$ n'est pas un $\Sigma^{\star\star}$-message. Donc $(v\sigma_\mathcal{Q}^-)\rho_1 = v(\sigma_\mathcal{Q}^-\rho_1)$ n'est toujours pas un message, ce qui permet de conclure.
3. *Cas* $\alpha = \mathsf{in}(c, R)$. Dans ce cas, nous avons $\mathcal{P}^- = \mathsf{in}(c, u).P_0 \uplus \mathcal{P}_0$. Ou bien $\mathcal{Q}^-$ ne peut pas exécuter de réception sur le canal c, et donc $(\mathcal{Q}^-; \psi^-; \sigma_\mathcal{Q}^-; i^-)$ n'est pas prêt à exécuter une réception sur le canal c, ce qui démontre le résultat ; ou bien, puisque $\phi^- \sqsubseteq_s \psi^-$, et $R\phi^-\!\downarrow$ est un message, nous déduisons que $R\psi^-\!\downarrow$ est un message, et nous avons que $R\psi^-\!\downarrow\rho_0 = (R\psi^-)\rho_0\!\downarrow = (R\rho_0)(\psi^-\rho_0)\!\downarrow$ est un message (premier point du lemme 3.1). Donc, puisque la réception ne peut pas être exécutée, il s'agit d'un problème de filtrage, c'est-à-dire qu'il n'existe pas de substitution τ telle que $(v\sigma_\mathcal{Q}^-)\tau = R\psi^-\!\downarrow$. Si $R\psi^-\!\downarrow$ et $v\sigma_\mathcal{Q}^-$ ne s'unifient pas à cause de leur structure, alors aucun renommage ne changera cette situation. Nous considérons donc le renommage ρ_0, et les termes $(R\rho_0)(\psi^-\rho_0)\!\downarrow$ et $v(\sigma^-\mathcal{Q}\rho_0)$ ne s'unifient pas. Sinon, la seule possibilité est qu'il existe deux positions p_1 et p_2 dans $(v\sigma_\mathcal{Q}^-)$ telles que $(v\sigma_\mathcal{Q}^-)|_{p_1} = (v\sigma_\mathcal{Q}^-)|_{p_2} \in \mathcal{X}$, mais $(R\psi^-\!\downarrow)|_{p_1} \neq (R\psi^-\!\downarrow)|_{p_2}$. Grâce au lemme 3.1 (troisième point), il existe une constante c_0 telle que pour tout Σ_0^+-renommage ρ avec $c_0 \notin dom(\rho) \cup img(\rho)$, $t_1\rho \neq t_2\rho$. Si $c_0 \in \Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q}$, alors nous pouvons simplement considérer ρ_0 qui laisse la constante c_0 inchangée. Sinon, supposons sans perte de généralité que c_0 est $c_0^\star$ (ou $c_+^\star$) (quitte à réaliser un alpha-renommage), et nous considérons le renommage ρ qui envoie toute constante de $\Sigma^{\star\star}$ (sauf $c_0^\star$ ou $c_+^\star$) sur $c_1^\star$.

Cas 2 : Nous avons $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}} (\mathcal{P}; \phi; \sigma_\mathcal{P}; i)$; $\mathcal{K}_\mathcal{Q} \xrightarrow{\mathsf{tr}} (\mathcal{Q}; \psi; \sigma_\mathcal{Q}; i)$; et $\phi \not\sqsubseteq_s \psi$. D'après le lemme 3.2, nous avons également :

— $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}\rho} (\mathcal{P}; \phi\rho; \sigma_\mathcal{P}\rho; i)$; et

— $\mathcal{K}_\mathcal{Q} \xrightarrow{\mathsf{tr}\rho} (\mathcal{Q}; \psi\rho; \sigma_\mathcal{Q}\rho; i)$

pour tout A-renommage ρ tel que $\mathsf{A} \subseteq \Sigma_0^+ \setminus (\Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}})$. Pour conclure, il reste à justifier que $\phi\rho \not\sqsubseteq \psi\rho$ pour un Σ_0^+-renommage ρ tel que $img(\rho) \subseteq \{c_0^\star, c_1^\star, c_+^\star\}$. Nous distinguons deux cas suivant la forme du test.

1. *Cas où le test d'équivalence statique est un message.* Il existe une $\Sigma^{\star\star}$-recette R telle que $R\phi\downarrow$ est un Σ_0^+-message tandis que $R\psi\downarrow$ n'en est pas un. Donc, d'après le lemme 3.1 (deuxième point), il existe une constante $c_0 \in \Sigma_0^+$ telle que pour tout Σ_0^+-renommage ρ vérifiant $c_0 \notin dom(\rho) \cup img(\rho)$, nous avons que $(R\psi)\rho\downarrow$ n'est pas un message. Dans le cas où $c_0 \in \Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}$, nous considérons le renommage ρ_0. D'après le lemme 3.1 (premier point) $(R\rho_0)(\phi\rho_0)\downarrow = (R\phi)\rho_0\downarrow = R\phi\downarrow\rho_0$ est un message. De plus, nous avons que $(R\rho_0)(\psi\rho_0)\downarrow = (R\psi)\rho_0\downarrow$ n'est pas un message. Ceci nous permet de conclure. Maintenant, si $c_0 \notin \Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}$, nous supposons sans perte de généralité que c_0 est $c_0^\star$, ou $c_+^\star$ si c_0 est de sorte $\mathsf{bitstring}$ (quitte à effectuer un alpha-renommage), et nous considérons le renommage ρ qui envoie toutes les constantes de $\Sigma^{\star\star}$ (sauf $c_0^\star$ or $c_+^\star$) vers $c_1^\star$. D'après le lemme 3.1 (premier point), $(R\rho)(\phi\rho)\downarrow = (R\phi)\rho\downarrow = R\phi\downarrow\rho$ est un message. De plus, nous avons vu que $(R\rho)(\psi\rho)\downarrow = (R\psi)\rho\downarrow$ n'est pas un message. Donc, nous obtenons un témoin de non-inclusion.
2. *Cas où le test d'équivalence statique est un test d'égalité.* Il existe deux $\Sigma^{\star\star}$-recettes R_1 et R_2 telles que $R_1\phi\downarrow$, $R_2\phi\downarrow$ sont des messages, et $R_1\phi\downarrow = R_2\phi\downarrow$. De plus, nous avons que $R_1\psi\downarrow$ et $R_2\psi\downarrow$ sont des messages, mais $R_1\psi\downarrow \neq R_2\psi\downarrow$. D'après le lemme 3.1 (troisième point), il existe une constante $c_0 \in \Sigma_0^+$ telle que pour tout Σ_0^+-renommage ρ avec $c_0 \notin dom(\rho) \cup img(\rho)$, nous avons $(R_1\psi)\rho\downarrow \neq (R_2\psi)\rho\downarrow$. Dans le cas où $c_0 \in \Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}$, nous considérons le renommage ρ_0. D'après le lemme 3.1 (premier point) $(R_i\rho_0)(\phi\rho_0)\downarrow = (R_i\phi)\rho_0\downarrow = R_i\phi\downarrow\rho_0$ est un message (pour chaque $i \in \{1,2\}$). Similairement, nous avons $(R_i\rho_0)(\psi\rho_0)\downarrow = (R_i\psi)\rho_0\downarrow = R_i\psi\downarrow\rho_0$ avec $i \in \{1,2\}$. Nous pouvons en conclure que $R_1\rho_0 \stackrel{?}{=} R_2\rho_0$ est un test qui tient dans $\phi\rho_0$ mais pas dans $\psi\rho_0$. Maintenant, dans le cas $c_0 \notin \Sigma_{\mathcal{P}} \cup \Sigma_{\mathcal{Q}}$, nous supposons sans perte de généralité que c_0 est $c_0^\star$ (ou $c_+^\star$) (quitte à effectuer un alpha-renommage), et nous considérons le renommage ρ qui envoie chaque constante de $\Sigma^{\star\star}$ (sauf $c_0^\star$ ou $c_+^\star$) vers $c_1^\star$. D'après le lemme 3.1 (premier point) $(R_i\rho)(\phi\rho)\downarrow = (R_i\phi)\rho\downarrow = R_i\phi\downarrow\rho$ est un message (pour chaque $i \in \{1,2\}$). De même, nous avons $(R_i\rho)(\psi\rho)\downarrow = (R_i\psi)\rho\downarrow = R_i\psi\downarrow\rho$ pour chaque $i \in \{1,2\}$. Nous pouvons en conclure que $R_1\rho \stackrel{?}{=} R_2\rho$ est un test qui tient dans $\phi\rho$ mais pas dans $\psi\rho$. Nous obtenons donc un témoin de non-inclusion.

Nous avons traité tous les cas, et nous obtenons donc le résultat. □

3.4 Contre-exemples

Notre classe de protocoles est relativement limitée : en particulier, nous sommes contraints par l'hypothèse de déterminisme, par l'absence de branches *else* et par la notion de théories à constructeurs. L'objectif de cette section est de montrer que chacune de ces hypothèses est nécessaire, c'est-à-dire que si nous enlevons une seule d'entre elles, il n'est plus possible de calculer une borne indépendante du protocole. Cependant, dans un modèle non typé et sans sorte, l'attaquant peut remplacer les constantes $c_0, \ldots, c_n$ par $c_0, \mathsf{hash}(c_0), \ldots, \mathsf{hash}(\ldots \mathsf{hash}(c_0) \ldots)$ et il lui suffit donc d'une seule constante. Tous les exemples qui suivent supposent donc que les exécutions sont quasiment typées.

Pour commencer, en section 3.4.1, nous présentons le codage d'un problème indécidable sous forme d'équivalence de protocoles conformes à un système de types structuré, parce que ce codage sera adapté successivement dans les deux sections suivantes. Remarquons que ce codage permet de démontrer que l'hypothèse de conformité ne suffit pas pour obtenir un résultat de décidabilité. La section 3.4.2 montre, en adaptant le codage précédent, que si nous ajoutons des branches *else*, ou simplement des conditionnelles portant sur des conditions négatives (aussi simples que $x \neq y$),

aucune borne n'est calculable (au sens de la théorie de la calculabilité). Ensuite, en modifiant le codage de la section 3.4.2, nous obtenons un exemple analogue pour des protocoles sans branches *else*, mais non-déterministes, dans la section 3.4.3. Dans ces deux situations, aucune borne, même dépendante du protocole, n'est calculable. La section 3.4.4 donne, plus modestement, un exemple où la borne dépend du protocole considéré, en présence d'une théorie équationnelle simple. Il n'a pas été démontré qu'il était possible, dans tous les cas, de calculer une telle borne à partir du protocole, mais l'exemple utilise des termes de taille arbitraire, qu'il semble difficile d'obtenir dans un modèle d'exécution typé.

La plupart des énoncés de cette section sont admis, et toutes les démonstrations sont informelles.

3.4.1 Indécidabilité dans le modèle typé.

Ce codage a été proposé par Durgin, Lincoln, Mitchell et Scedrov [70]. Il permet de montrer que le problème de vérification reste indécidable même en présence d'un modèle typé, c'est-à-dire pour des protocoles conformes à un système de types structuré, et des exécutions quasiment typées.

Nous allons procéder en codant le problème de correspondance de Post (PCP) [90] dans notre algèbre de processus, de sorte que borner le nombre de constantes nécessaires pour vérifier le protocole revienne à borner la taille des solutions de PCP. Comme PCP n'est pas décidable, une telle borne n'est pas calculable. Cette démarche suppose que les exécutions du protocole soient de longueur arbitraire, puisque l'équivalence est décidable sans réplication, ce qui interdit tout espoir de coder PCP. Pourtant, ce résultat s'applique également aux protocoles sans réplication : dire qu'aucune borne sur le nombre de constantes nécessaires à l'attaquant pour attaquer $!P$ n'est calculable revient à affirmer qu'il n'existe pas de borne (indépendante du protocole considéré) sur le nombre de constantes nécessaires à l'attaquant pour vérifier les protocoles de la famille des P_i où chaque P_i représente i sessions de P en parallèle.

Nous étendons donc l'algèbre de processus considérée à la grammaire suivante :

$$
\begin{array}{rcll}
P, Q & := & 0 & \text{processus nul} \\
 & | & \mathsf{in}(c, u).P & \text{entrée} \\
 & | & \mathsf{out}(c, u).P & \text{sortie} \\
 & | & (P \mid Q) & \text{composition parallèle} \\
 & | & i : P & \text{phase} \\
 & | & !\mathsf{new}\, c'.\mathsf{out}(c, c').P & \text{réplication} \\
 & | & \mathsf{new}\, n.P & \text{renommage}
\end{array}
$$

La règle supplémentaire $!\mathsf{new}\, c'.\mathsf{out}(c, c').P$ permet de créer une nouvelle instance de P après avoir déclaré un canal frais c' dédié à cette nouvelle instance. Cette nouvelle construction permet au protocole de rester déterministe en présence de réplications. La construction $\mathsf{new}\, n$ crée un nonce frais pour remplacer toutes les occurences de n dans P.

Les règles de sémantique associées à ces constructions sont les suivantes, les autres étant inchangées :

$$(i{:}\mathsf{new}\ n.P \cup \mathcal{P}; \phi; \sigma; i) \xrightarrow{\tau} (i{:}P\{n \triangleright n'\} \cup \mathcal{P}; \phi; \sigma; i) \quad \text{avec } n' \text{ un nom frais de } \mathcal{N}.$$

$$(i{:}!\,\mathsf{new}\, c'.\mathsf{out}(c, c').P \cup \mathcal{P}; \phi; \sigma; i) \xrightarrow{\mathsf{sess}(c,ch)} (i{:}P\{c' \triangleright ch\} \cup i{:}!\,\mathsf{new}\, c'.\mathsf{out}(c, c').P \cup \mathcal{P}; \phi; \sigma; i)$$
avec ch un canal frais de $\mathcal{C}h$.

Une instance de PCP sur un alphabet A est donnée par deux ensembles de tuiles $U = \{u_i \mid 1 \leqslant i \leqslant n\}$ et $V = \{v_i \mid 1 \leqslant i \leqslant n\}$ où $u_i, v_i \in \mathsf{A}^*$. Une solution de PCP est une séquence non-vide

$i_1, \ldots, i_p$ sur $\{1, \ldots, n\}$ telle que $u_{i_1} \ldots u_{i_p} = v_{i_1} \ldots v_{i_p}$. Décider si une instance de PCP admet une solution est un problème indécidable. Étant donné un mot $u = \alpha_1 \ldots . \alpha_k$ de $\mathsf{A}^\star$, nous notons $u^i = \alpha_i$. En particulier, u_i^j (resp. v_i^j) désigne la j^e lettre de la tuile u_i (resp. v_i). Par ailleurs, k_i représente la longueur de u_i et j_i la longueur de v_i.

L'idée consiste à se servir de nonces pour construire des listes chaînées : le codage

$$\langle n_0, a_0, n_1\rangle, \langle n_1, a_1, n_2\rangle \ldots \langle n_{k-1}, a_{k-1}, n_k\rangle, \langle n_k, a_k, n_{k+1}\rangle$$

permet de représenter une liste $a_0, \ldots, a_k$ de longueur arbitraire, en n'utilisant que des termes de profondeur bornée, si chaque a_i est une constante.

Le processus principal, sur le canal c_i, reçoit deux pointeurs représentés par des nonces, puis ajoute une tuile de chaque côté et renvoie les deux nouveaux pointeurs. U, V et K sont des noms inconnus de l'attaquant. Il s'agit du processus suivant :

$$\begin{aligned}
P_i(c_i) = {} & \mathsf{in}(c_i, \mathsf{senc}(\langle x, y\rangle, K).\\
& \mathsf{new}\, n_2.\mathsf{out}(c_i, \mathsf{senc}(\langle x, u_i^1, n_2\rangle, U)).\\
& \ldots\\
& \mathsf{new}\, n_{k_i}.\mathsf{out}(c_i, \mathsf{senc}(\langle n_{k_i-1}, u_i^{k_i}, n_{k_i}\rangle, U)).\\
& \mathsf{new}\, m_2.\mathsf{out}(c_i, \mathsf{senc}(\langle y, v_i^1, m_2\rangle, V)).\\
& \ldots\\
& \mathsf{new}\, m_{j_i}.\mathsf{out}(c_i, \mathsf{senc}(\langle m_{j_i-1}, v_i^{j_i}, m_{j_i}\rangle, V)).\\
& \mathsf{out}(c_i, \mathsf{senc}(\langle n_{k_i}, m_{j_i}\rangle, K)
\end{aligned}$$

Le motif $\langle x, y\rangle$ de la première ligne sert à instancier les nonces qui représentent des pointeurs vers la partie précédente des mots construits. Nous avons besoin de deux pointeurs puisque u_i n'est pas toujours de la même taille que v_i (sinon PCP est décidable et même trivial). Les six lignes suivantes ajoutent les lettres de u_i, puis celles de v_i, aux deux listes chaînées, tandis que la dernière émet les deux nouveaux pointeurs, qui permettront de boucler avec la première ligne. Pour ce qui est du système de type, pour tout i nous avons $\delta(n_i) = \delta(m_i) = \delta(x) = \delta(y) = \tau_n \in \mathcal{T}_0$, et par ailleurs, $\delta(u_i^1) = \delta(u_i^2) = \cdots = \tau_u \in \mathcal{T}_0$, $\delta(v_i^1) = \delta(v_i^2) = \cdots = \tau_u \in \mathcal{T}_0$. Les types de U, V et K n'ont pas d'intérêt, mais pour fixer les idées nous pouvons poser $\delta(K) = \delta(U) = \delta(V) = \tau_k \in \mathcal{T}_0$.

Ensuite, il faut considérer un processus qui nous permet de commencer. $\bot_U$ et $\bot_V$ sont des constantes (avec $\delta(\bot_U) = \delta(\bot_V) = \tau_m$) qui représentent la liste vide :

$$B = \mathsf{out}(c_0, \mathsf{senc}(\langle \bot_U, \bot_V\rangle, K))$$

Puis nous devons disposer d'un moyen de vérifier la solution, en remontant les deux listes chaînées et en vérifiant qu'elles sont identiques (nous utilisons deux fois la même variable z_1).

$$\begin{aligned}
C(c_j) = {} & \mathsf{in}(c_j, \mathsf{senc}(\langle x', y'\rangle, K).\\
& \mathsf{in}(c_j, \mathsf{senc}(\langle x', z_1, z_p\rangle, U).\\
& \mathsf{in}(c_j, \mathsf{senc}(\langle y', z_1, z_p'\rangle, V).\\
& \mathsf{out}(c_j, \mathsf{senc}(\langle z_p, z_p'\rangle, K)
\end{aligned}$$

Le système de types vérifie $\delta(x') = \delta(y') = \delta(z_p) = \delta(z_p') = \tau_n$, et $\delta(z_1) = \tau_u$.

Enfin, les deux processus qui permettent de garantir que nous sommes remontés jusqu'au bout :

$$\begin{aligned} F_P = \mathsf{in}(c_f, \mathsf{senc}(\langle x'', y''\rangle, K).\\ \mathsf{in}(c_f, \mathsf{senc}(\langle x'', z_2, \bot_U\rangle, U).\\ \mathsf{in}(c_f, \mathsf{senc}(\langle y'', z_2, \bot_V\rangle, V).\\ \mathsf{out}(c_f, a) \end{aligned}$$

$$\begin{aligned} F_Q = \mathsf{in}(c_f, \mathsf{senc}(\langle x'', y''\rangle, K).\\ \mathsf{in}(c_f, \mathsf{senc}(\langle x'', z_2, \bot_U\rangle, U).\\ \mathsf{in}(c_f, \mathsf{senc}(\langle y'', z_2, \bot_V\rangle, V).\\ \mathsf{out}(c_f, b) \end{aligned}$$

avec a, b des constantes ($\delta(a) = \delta(b) = \tau_a$), $\delta(x'') = \delta(y'') = \tau_n$ et $\delta(z_2) = \tau_u$.

Les protocoles que nous considérons sont les suivants :

$$\begin{aligned} \mathcal{P} = \{B, F_P, !\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).P_1(c'_1), \ldots, !\mathsf{new}\, c'_n.\mathsf{out}(c_n, c'_n).P_n(c'_n),\\ !\mathsf{new}\, c'_{n+1}.\mathsf{out}(c_{n+1}, c'_{n+1}).C(c'_{n+1})\}\\ \mathcal{Q} = \{B, F_Q, !\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).P_1(c'_1), \ldots, !\mathsf{new}\, c'_n.\mathsf{out}(c_n, c'_n).P_n(c'_n),\\ !\mathsf{new}\, c'_{n+1}.\mathsf{out}(c_{n+1}, c'_{n+1}).C(c'_{n+1})\} \end{aligned}$$

$\mathcal{P}$ et $\mathcal{Q}$ sont conformes au système de types structuré $(\mathcal{T}, \delta)$ puisque les sous-termes chiffrés sont répartis entre trois catégories :

— les termes de la forme $\mathsf{senc}(\_, K)$, qui sont tous de type $\mathsf{senc}(\langle \tau_n, \tau_n\rangle, \tau_k)$.

— les termes de la forme $\mathsf{senc}(\_, U)$, qui sont tous de type $\mathsf{senc}(\langle \tau_n, \tau_u, \tau_n\rangle, \tau_k)$.

— les termes de la forme $\mathsf{senc}(\_, V)$, qui sont tous de type $\mathsf{senc}(\langle \tau_n, \tau_u, \tau_n\rangle, \tau_k)$.

Pourtant, les inclusions entre $\mathcal{P}$ et $\mathcal{Q}$ sont indécidables, puisqu'il existe un témoin de non-inclusion si, et seulement si, il existe une solution à l'instance de PCP considérée. Cette situation ne démontre pas que l'attaquant a besoin d'un nombre arbitraire de constantes, mais elle constitue un point de départ pour la suite. Nous admettons donc la propriété suivante :

Proposition 3.1. *Il existe une trace de non-inclusion $\mathcal{P} \not\sqsubseteq_{at} \mathcal{Q}$ si, et seulement si, l'instance de* PCP *considérée admet une solution.*

3.4.2 Conditionnelles

Une extension naturelle consiste à considérer des processus avec des branches *else*. Cependant, lorsque les messages émis sur ces branches peuvent dépendre, directement ou indirectement, de certaines constantes, il devient nécessaire que l'attaquant dispose d'un nombre arbitraire de constantes. Ce résultat négatif est déjà valable pour les propriétés d'accessibilité.

Étant donné un entier n arbitraire, nous notons Σ^n un ensemble de cardinal n de constantes de type $\tau^\star$. Formellement, nous montrons que nous pouvons associer, pour chaque instance de PCP, deux protocoles $\mathcal{P}$ et $\mathcal{Q}$ (qui utilisent exclusivement le chiffrement symétrique et la paire) tels que $\mathcal{P} \not\sqsubseteq_{at} \mathcal{Q}$ avec n constantes, si, et seulement si, l'instance de PCP a une solution de longueur plus petite que n. Il s'agit donc de modifier les protocoles de la section précédente pour que l'attaquant ne puisse vérifier sa solution que s'il dispose d'autant de constantes que la longueur (en nombre de tuiles) de la solution de PCP.

L'étape la plus nouvelle consiste à établir un moyen de vérifier que l'attaquant dispose d'une liste de constantes toutes différentes. Le nom D (avec $\delta(D) = \tau_k$) représente une clé inconnue de

l'attaquant, la variable x_c^0 représente une clé ajoutée par l'attaquant, de type $\delta(x_c^0) = \tau_c \in \mathcal{T}$, et les nonces n_c, n'_c, ainsi que la constante $\bot_{nc}$ et les variables x_{nc}, x'_{nc}, sont de type τ_{nc}. $\bot_{nc}$ permet de représenter la liste vide.

$$V_0(c) = \mathsf{in}(c, x_c^0).\mathsf{new}\, n_c.\mathsf{out}(c, \mathsf{senc}(\langle n_c, x_c^0, \bot_{nc}\rangle, D))$$

$$\begin{aligned} V_1(c) = {} & \mathsf{in}(c, \mathsf{senc}(\langle x_{nc}, x_c^1, x'_{nc}\rangle, D)).\\ & \mathsf{in}(c, \mathsf{senc}(\langle y_{nc}, y_c^1, x'_{nc}\rangle, D)).\\ & \mathsf{if}\ x_c^1 \neq y_c^1\ \mathsf{then}\\ & \mathsf{new}\, n'_c.\mathsf{out}(c, \mathsf{senc}(\langle n'_c, x_c^1, y_{nc}\rangle, D)) \end{aligned}$$

Le processus V_0 permet d'obtenir des jetons $\langle n_c, x_c^0, \bot_{nc}\rangle$. Ensuite, le principe de V_1 est que les éléments de la liste $a :: b :: \ell$ sont deux à deux distincts si, et seulement si :

— les éléments des listes $a :: \ell$ et $b :: \ell$ sont deux à deux distincts ;

— et $a \neq b$.

À ce titre, la construction $\mathsf{if}\ x_c^1 \neq y_c^1\ \mathsf{then}$ a la sémantique attendue : elle continue de s'exécuter seulement si x_c^1 et y_c^1 sont instanciées par des valeurs différentes. Cette instruction peut se coder comme $\mathsf{if}\ x_c^1 = y_c^1\ \mathsf{then}\, 0\, \mathsf{else} \ldots$. Si nous conservions chaque fois le même pointeur de tête, il serait facile d'obtenir $\langle n_1, a, n_2\rangle$ puis $\langle n_2, b, n_1\rangle$, et ainsi de disposer d'une liste chaînée infinie avec seulement deux constantes. Le changement de pointeur, grâce au nonce n'_c, résout ce problème.

Exemple 3.3. *Si l'attaquant dispose de trois constantes a, b, c, il peut obtenir les termes suivants après trois utilisations de V_0 :*

$$\begin{aligned} t_a &= \mathsf{senc}(\langle n_1, a, \bot_{nc}\rangle, D)\\ t_b &= \mathsf{senc}(\langle n_2, b, \bot_{nc}\rangle, D)\\ t_c &= \mathsf{senc}(\langle n_3, c, \bot_{nc}\rangle, D) \end{aligned}$$

Il peut donc en déduire par exemple $t'_b = \mathsf{senc}(\langle n'_2, b, n_1\rangle, D)$ et $t'_c = \mathsf{senc}(\langle n'_3, c, n_1\rangle, D)$ en appliquant V_1 à t_b, t_a puis à t_c, t_a. Enfin, il peut obtenir $t''_c = \mathsf{senc}(\langle n''_3, c, n'_2\rangle, D)$ en appliquant V_1 à t'_c, t'_b. Il pourra alors certifier qu'il possède trois constantes distinctes en fournissant t''_c, t'_b et t_a, dont les nonces se suivent. Si il tente de tricher et d'utiliser deux fois la même constante, et récupère par exemple $\mathsf{senc}(\langle n_4, c, \bot_{nc}\rangle, D)$, il peut arriver à obtenir $\mathsf{senc}(\langle n''_4, c, n'_2\rangle, D)$, $\mathsf{senc}(\langle n''_2, b, n'_4\rangle, D)$ ou $\mathsf{senc}(\langle n'_1, a, n_4\rangle, D)$, mais à chaque fois ces termes représentent la tête d'une liste de trois constantes seulement. Il ne peut pas obtenir de liste plus longue sans utiliser une constante supplémentaire.

Plus formellement, démontrons le lemme suivant :

Lemme 3.3. *Soit $E = \{\mathsf{senc}(\langle n_1, a_0, n_0\rangle, D), \mathsf{senc}(\langle n_2, a_1, n_1\rangle, D) \ldots, \mathsf{senc}(\langle n_{k+1}, a_k, n_k\rangle, D)\}$ un ensemble de termes. Posons $\mathcal{P} = \{(\,!\,\mathsf{new}\, c'_0.\mathsf{out}(c_0, c'_0).V_0(c'_0) \mid !\,\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).V_1(c'_1))\}$ et supposons $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^+}(\mathcal{P})$, tels que $E \subseteq img(\phi)$. Alors pour tout $i \neq j$, $a_i \neq a_j$.*

Démonstration informelle. Nous procédons par récurrence sur k. Si $k = 1$, le résultat est évident. Supposons que le résultat soit vrai pour un k quelconque, et démontrons-le pour $k + 1$. Posons $t_i = \mathsf{senc}(\langle n_{i+1}, a_i, n_i\rangle, D)$ pour tout i. Si l'attaquant a réussi à connaître l'ensemble $\{t_1, \ldots, t_{k+1}\}$ alors il a réussi à connaître $\{t_1, \ldots, t_k\}$ et $\{t_1, \ldots, t_{k-1}, t_{k+1}\}$. L'hypothèse de récurrence s'applique, et nous avons $a_i \neq a_j$ pour tout $i \neq j$ tels que $i, j \leqslant k$. Or, comme le nom D n'est pas déductible (puisqu'il n'apparaît qu'en position de clé), l'attaquant a nécessairement appris t_{k+1} après avoir

exécuté V_1. Donc, il connaissait nécessairement $\mathsf{senc}(\langle n'_{k+2}, a_{k+1}, n'_k\rangle, D)$ et $\mathsf{senc}(\langle n_{k+1}, a, n'_k\rangle, D)$ avec n'_{k+2}, n'_k des noms et $a \neq a_{k+1}$. Pour un nonce n donné, il n'existe qu'un seul terme de la forme $\mathsf{senc}(\langle n, \_, \_\rangle, D)$, donc $a = a_k$ et $n'_k = n_k$ et $a_{k+1} \neq a_k$. En appliquant l'hypothèse de récurrence à $\{t_1, \ldots, t_{k-1}, \mathsf{senc}(\langle n'_{k+2}, a_{k+1}, n_k\rangle, D)\}$, nous obtenons de plus que $a_{k+1} \neq a_i$ pour chaque $i < k$, ce qui démontre le résultat. □

À partir de ce point, il suffit d'adapter les processus de la section précédente pour vérifier l'existence d'une liste chaînée de constantes toutes différentes au moins aussi longue que la solution.

Nous modifions B pour que l'attaquant doive fournir un jeton $\mathsf{senc}(\langle p_1, c, p_2\rangle, D)$: charge à lui de pouvoir fournir les jetons suivants au moment de construire la solution. Les variables x_p et x'_p vérifient $\delta(x_p) = \delta(x_p) = \tau_{nc}$, et la variable x_0 satisfait $\delta(x_0) = \tau_c$:

$$B = \mathsf{in}(c_0, \mathsf{senc}(\langle x_p, x_0, x'_p\rangle, D)).\mathsf{out}(c_0, \mathsf{senc}(\langle \bot_U, \bot_V, x_p\rangle, K))$$

Nous modifions $P_i(c_i)$ en demandant à l'attaquant de fournir un jeton qui pointe vers la bonne adresse. À la fin, nous mettons à jour l'adresse cible pour la prochaine étape (le nonce n_1 du terme $\mathsf{senc}(\langle n_1, m_1, y'_p\rangle, K)$). Les données $y_p, y_0, y_{p'}$ sont des variables, et $\delta(y_p) = \delta(y'_p) = \tau_{nc}$ tandis que $\delta(y_0) = \tau_c$. De plus, les jetons chiffrés par K contiennent maintenant une troisième information : en plus des pointeurs x et y vers les solutions en construction, ils contiennnent également un pointeur y_p vers le dernier élément de la liste de constantes construite : de cette manière, l'attaquant doit fournir une constante différente pour chaque nouvelle tuile.

$$\begin{aligned}
P_i(c_i) = {} & \mathsf{in}(c_i, \mathsf{senc}(\langle y'_p, y_0, y_p\rangle, D)).\mathsf{in}(c_i, \mathsf{senc}(\langle x, y, y_p\rangle, K).\\
& \mathsf{new}\, n_2.\mathsf{out}(c_i, \mathsf{senc}(\langle x, u_i^1, n_2\rangle, U)).\\
& \ldots\\
& \mathsf{new}\, n_{k_i}.\mathsf{out}(c_i, \mathsf{senc}(\langle n_{k_i-1}, u_i^{k_i}, n_{k_i}\rangle, U)).\\
& \mathsf{new}\, m_2.\mathsf{out}(c_i, \mathsf{senc}(\langle y, v_i^1, m_2\rangle, V)).\\
& \ldots\\
& \mathsf{new}\, m_{j_i}.\mathsf{out}(c_i, \mathsf{senc}(\langle m_{j_i-1}, v_i^{j_i}, m_{j_i}\rangle, V)).\\
& \mathsf{out}(c_i, \mathsf{senc}(\langle n_{k_i}, m_{j_i}, y'_p\rangle, K)
\end{aligned}$$

Ces changements suffisent à garantir que l'attaquant ne peut obtenir la solution que s'il dispose d'autant de constantes que la taille de la solution, en nombre de tuiles. Les autres modifications (avec z', z'' des variables de type τ_{nc}) sont triviales :

$$\begin{aligned}
C(c_j) = {} & \mathsf{in}(c_j, \mathsf{senc}(\langle x', y', z'\rangle, K).\\
& \mathsf{in}(c_j, \mathsf{senc}(\langle x', z_1, z_p\rangle, U).\\
& \mathsf{in}(c_j, \mathsf{senc}(\langle y', z_1, z'_p\rangle, V).\\
& \mathsf{out}(c_j, \mathsf{senc}(\langle z_p, z'_p, z'\rangle, K)
\end{aligned}$$

$$\begin{aligned}
F_P = {} & \mathsf{in}(c_f, \mathsf{senc}(\langle x'', y'', z''\rangle, K).\\
& \mathsf{in}(c_f, \mathsf{senc}(\langle x'', z_2, \bot_U\rangle, U).\\
& \mathsf{in}(c_f, \mathsf{senc}(\langle y'', z_2, \bot_V\rangle, V).\\
& \mathsf{out}(c_f, a)
\end{aligned}$$

$$\begin{aligned} F_Q = &\mathsf{in}(c_f, \mathsf{senc}(\langle x'', y'', z''\rangle, K).\\ &\mathsf{in}(c_f, \mathsf{senc}(\langle x'', z_2, \bot_U\rangle, U).\\ &\mathsf{in}(c_f, \mathsf{senc}(\langle y'', z_2, \bot_V\rangle, V).\\ &\mathsf{out}(c_f, b) \end{aligned}$$

Nous considérons les protocoles $\mathcal{P}$ et $\mathcal{Q}$ suivants :

$$\begin{aligned} \mathcal{P} = \{&B, F_P, !\,\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).P_1(c'_1), \ldots, !\,\mathsf{new}\, c'_n.\mathsf{out}(c_n, c'_n).P_n(c'_n),\\ &!\,\mathsf{new}\, c'_{n+1}.\mathsf{out}(c_{n+1}, c'_{n+1}).C(c'_{n+1}), !\,\mathsf{new}\, c'_{n+2}.\mathsf{out}(c_{n+2}, c'_{n+2}).V_0(c'_{n+2}),\\ &!\,\mathsf{new}\, c'_{n+3}.\mathsf{out}(c_{n+3}, c'_{n+3}).V_1(c'_{n+3})\} \end{aligned}$$

$$\begin{aligned} \mathcal{Q} = \{&B, F_Q, !\,\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).P_1(c'_1), \ldots, !\,\mathsf{new}\, c'_n.\mathsf{out}(c_n, c'_n).P_n(c'_n),\\ &!\,\mathsf{new}\, c'_{n+1}.\mathsf{out}(c_{n+1}, c'_{n+1}).C(c'_{n+1}), !\,\mathsf{new}\, c'_{n+2}.\mathsf{out}(c_{n+2}, c'_{n+2}).V_0(c'_{n+2}),\\ &!\,\mathsf{new}\, c'_{n+3}.\mathsf{out}(c_{n+3}, c'_{n+3}).V_1(c'_{n+3})\} \end{aligned}$$

Il est clair que $\mathcal{P}$ et $\mathcal{Q}$ sont conformes au système de type $(\mathcal{T}, \delta)$. Concernant les ensembles de constantes, nous avons $\Sigma_\mathcal{P} = \Sigma_\mathcal{Q} = \{\bot_V, \bot_U, \bot_{nc}\}$.

Proposition 3.2. *Il existe un témoin quasiment typé de non-inclusion $\mathcal{P} \not\sqsubseteq_{at} \mathcal{Q}$ par rapport à $\Sigma^n \uplus (\Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q})$ si, et seulement si, l'instance de* PCP *considérée admet une solution de longueur au plus n.*

Démonstration informelle. Si l'instance de PCP admet une solution de longueur au plus n, il est facile de construire un témoin quasiment typé, en utilisant les constantes différentes pour obtenir une liste chaînée (par les nonces) de constantes.

Réciproquement, supposons qu'il existe un témoin quasiment typé de $\mathcal{P} \not\sqsubseteq_{at} \mathcal{Q}$ par rapport à $\Sigma^n \uplus \{\bot_V, \bot_U, \bot_{nc}\}$. Nous admettons, comme pour la proposition 3.1, qu'il existe une solution de l'instance de PCP. Il reste à montrer que cette solution est de longueur au plus n.

La solution a été construite à travers les processus P_i et donc pour chaque tuile, l'attaquant a dû fournir $\mathsf{senc}(\langle y_p, y_0, y'_p\rangle, D)$ avec $y_{p'}$ un pointeur vers une constante précédente. De plus, comme le témoin est quasiment typé, la variable y_0 a dû être instanciée par un terme t tel que $\delta(t) \preccurlyeq \delta(y_0) = \tau_c$: nécessairement t est une donnée. Comme l'attaquant ne connaît aucune autre donnée que les constantes de $\Sigma_n \uplus \{\bot_V, \bot_U, \bot_{nc}\}$, il a utilisé chaque fois l'une de ces constantes. D'après le lemme 3.3, les constantes utilisées sont deux à deux distinctes. Nécessairement, l'attaquant dispose de plus de constantes de type $\tau^\star$ qu'il n'y a de tuiles dans la solution, donc $n \geqslant \ell$ où ℓ est la longueur de la solution. □

3.4.3 Non-déterminisme

Comme dans la section précédente, la principale difficulté consiste à trouver un moyen de vérifier que les éléments d'une liste sont deux à deux distincts. Nous utilisons deux constantes supplémentaires true et false qui représentent des booléens. Nous commençons par initialiser la liste chaînée, comme à la section précédente :

$$V_0(c) = \mathsf{in}(c, x_c^0).\mathsf{new}\, n_c.\mathsf{out}(c, \mathsf{senc}(\langle n_c, x_c^0, \mathsf{false}, \bot_{nc}\rangle, D))$$

Le booléen est initialisé à false, puisque dans la suite, le défenseur peut toujours choisir de garder, côté Q, le booléen true s'il l'a obtenu une seule fois (true signifie que l'attaquant a utilisé au moins deux fois la même constante).

L'algorithme utilisé est le même que dans la section précédente, à savoir que si $a :: \ell$ et $b :: \ell$ sont des listes de termes deux à deux distincts, et $a \neq b$, alors $a :: b :: \ell$ est une liste d'éléments deux à deux distincts. Cependant, nous nous appuyons cette fois-ci sur le non-déterminisme : nous utilisons le fait que, côté $\mathcal{Q}$, il suffit qu'il existe *une* exécution qui imite celle de $\mathcal{P}$ pour que $\mathcal{P}$ soit inclus dans $\mathcal{Q}$. Le processus V_1^P que nous utilisons côté $\mathcal{P}$, pour remplacer le processus V_1 de la section précédente, est relativement simple, et son exécution ne dépend pas de la valeur des constantes reçues (en particulier, le fait qu'elles soient différentes n'apporte rien). Les variables x_{bool}, y_{bool} sont de type $\delta(x_{bool}) = \delta(y_{bool}) = \delta(\mathsf{true}) = \delta(\mathsf{false}) = \tau_{bool}$.

$$\begin{aligned} V_1^P(c) = \mathsf{in}(c, \mathsf{senc}(\langle x_{nc}, x_c^1, x_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{in}(c, \mathsf{senc}(\langle y_{nc}, y_c^1, y_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{new}\, n'_c.\mathsf{out}(c, \mathsf{senc}(\langle n'_c, x_c^1, \mathsf{true}, y_{nc}\rangle, D)) \end{aligned}$$

La subtilité se trouve côté Q : il faut donner la possibilité de passer par une branche qui imite P si, et seulement si, les constantes qui instancient les variables x_c^1 et y_c^1 sont égales. Plus précisément, les processus sont équivalents si, et seulement si, pour chaque exécution de P, il existe une exécution de Q qui l'imite. Informellement, le défenseur peut choisir l'exécution qui lui convient le mieux côté Q : nous allons donc faire en sorte que si l'attaquant utilise plusieurs fois la même constante, le défenseur puisse toujours choisir l'un des processus V_2^Q, V_3^Q, V_4^Q qui imitent P ; si l'attaquant utilise des constantes deux-à-deux distincts, le défenseur sera forcé de choisir V_1^Q, qui ne permet pas d'imiter V_1^P. Ce processus V_1^Q est toujours accessible au défenseur, mais il n'a jamais intérêt à le choisir. Définissons donc les processus suivants (pour améliorer la lisibilité, les variables n'ont pas été renommées). Le processus V_1^Q correspond à celui que le défenseur n'a pas intérêt à choisir.

$$\begin{aligned} V_1^Q(c) = \mathsf{in}(c, \mathsf{senc}(\langle x_{nc}, x_c^1, x_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{in}(c, \mathsf{senc}(\langle y_{nc}, y_c^1, y_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{new}\, n'_c.\mathsf{out}(c, \mathsf{senc}(\langle n'_c, x_c^1, \mathsf{false}, y_{nc}\rangle, D)) \end{aligned}$$

Le processus V_2 peut être utilisé par le défenseur quand x_c^1 apparaît deux fois, c'est-à-dire chaque fois que l'attaquant utilise la même constante deux fois. À la différence du processus précédent, l'attaquant obtient donc un jeton contenant true.

$$\begin{aligned} V_2^Q(c) = \mathsf{in}(c, \mathsf{senc}(\langle x_{nc}, x_c^1, x_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{in}(c, \mathsf{senc}(\langle y_{nc}, x_c^1, y_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{new}\, n'_c.\mathsf{out}(c, \mathsf{senc}(\langle n'_c, x_c^1, \mathsf{true}, y_{nc}\rangle, D)) \end{aligned}$$

Les deux autres processus servent à traiter le cas où l'attaquant a utilisé un jeton contenant true, c'est-à-dire qu'il a déjà utilisé deux constantes identiques dans le passé. Dans ce cas, le défenseur a également la possibilité d'imiter P, ce qui force l'attaquant à utiliser des constantes toutes différentes.

$$\begin{aligned} V_3^Q(c) = \mathsf{in}(c, \mathsf{senc}(\langle x_{nc}, x_c^1, \mathsf{true}, x'_{nc}\rangle, D)).\\ \mathsf{in}(c, \mathsf{senc}(\langle y_{nc}, y_c^1, y_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{new}\, n'_c.\mathsf{out}(c, \mathsf{senc}(\langle n'_c, x_c^1, \mathsf{true}, y_{nc}\rangle, D)) \end{aligned}$$

$$\begin{aligned} V_4^Q(c) = \mathsf{in}(c, \mathsf{senc}(\langle x_{nc}, x_c^1, x_{bool}, x'_{nc}\rangle, D)).\\ \mathsf{in}(c, \mathsf{senc}(\langle y_{nc}, y_c^1, \mathsf{true}, x'_{nc}\rangle, D)).\\ \mathsf{new}\, n'_c.\mathsf{out}(c, \mathsf{senc}(\langle n'_c, x_c^1, \mathsf{true}, y_{nc}\rangle, D)) \end{aligned}$$

L'idée est d'utiliser $V^P = !\,\mathsf{new}\, c'.\mathsf{out}(c, c').V_1^P(c')$ côté P et, côté Q, le processus :

$$\begin{aligned} V^Q = {} & !\,\mathsf{new}\, c'.\mathsf{out}(c, c').V_1^Q(c') \mid !\,\mathsf{new}\, c'.\mathsf{out}(c, c').V_2^Q(c') \mid !\,\mathsf{new}\, c'.\mathsf{out}(c, c').V_3^Q(c') \\ & \mid !\,\mathsf{new}\, c'.\mathsf{out}(c, c').V_4^Q(c') \end{aligned}$$

Ces deux processus s'exécutent différemment, mais l'attaquant ne peut pas savoir de quel côté il se trouve. S'il utilise plusieurs fois la même constante, ou s'il a utilisé plusieurs fois la même constante à l'une des étapes précédentes de la vérification, il obtient systématiquement un jeton $\langle n'_c, x_c^1, \mathsf{true}, y_{nc}\rangle$ (chiffré sous la clé D) des deux côtés.

Ensuite, nous adaptons la vérification finale. Il est important que la vérification du booléen se fasse à l'avant-dernière étape, sinon l'attaquant pourrait utiliser cette vérification sans avoir trouvé de solution au préalable.

$$\begin{aligned} F = {} & \mathsf{in}(c_f, \mathsf{senc}(\langle x'', z_2, \bot_U\rangle, U).\\ & \mathsf{in}(c_f, \mathsf{senc}(\langle y'', z_2, \bot_V\rangle, V).\\ & \mathsf{in}(c_f, \mathsf{senc}(\langle x'', y'', \mathsf{true}, z''\rangle, K).\\ & \mathsf{out}(c_f, \mathsf{true}) \end{aligned}$$

L'idée est que, si l'attaquant a utilisé plusieurs fois la même constante, il a nécessairement obtenu le booléen true des deux côtés, et ce processus s'exécute donc de la même manière des deux côtés. Si, au contraire, il possède un jeton contenant false côté Q, qui ne passe pas, et il est capable de distinguer entre P et Q.

Les processus P_i doivent également tenir compte du booléen, et garder chaque fois le dernier obtenu, représenté par la variable z'_{bool} (z_{bool}, z'_{bool} sont des variables de type τ_{bool}).

$$\begin{aligned} P_i(c_i) = {} & \mathsf{in}(c_i, \mathsf{senc}(\langle y'_p, y_0, z'_{bool}, y_p\rangle, D)).\mathsf{in}(c_i, \mathsf{senc}(\langle x, y, z_{bool}, y_p\rangle, K).\\ & \mathsf{new}\, n_2.\mathsf{out}(c_i, \mathsf{senc}(\langle x, u_i^1, n_2\rangle, U)).\\ & \dots\\ & \mathsf{new}\, n_{k_i}.\mathsf{out}(c_i, \mathsf{senc}(\langle n_{k_i-1}, u_i^{k_i}, n_{k_i}\rangle, U)).\\ & \mathsf{new}\, m_2.\mathsf{out}(c_i, \mathsf{senc}(\langle y, v_i^1, m_2\rangle, V)).\\ & \dots\\ & \mathsf{new}\, m_{j_i}.\mathsf{out}(c_i, \mathsf{senc}(\langle m_{j_i-1}, v_i^{j_i}, m_{j_i}\rangle, V)).\\ & \mathsf{new}\, n_c.\mathsf{out}(c_i, \mathsf{senc}(\langle n_{k_i}, m_{j_i}, z'_{bool}, y'_p\rangle, K) \end{aligned}$$

Nous devons également adapter le processus C de la même manière, avec z''_{bool} une variable de

type τ_{bool} :

$$\begin{aligned} C(c_j) =& \mathsf{in}(c_j, \mathsf{senc}(\langle x', y', z''_{bool}, z' \rangle, K).\\ & \mathsf{in}(c_j, \mathsf{senc}(\langle x', z_1, z_p \rangle, U).\\ & \mathsf{in}(c_j, \mathsf{senc}(\langle y', z_1, z'_p \rangle, V).\\ & \mathsf{out}(c_j, \mathsf{senc}(\langle z_p, z'_p, z''_{bool}, z' \rangle, K) \end{aligned}$$

Le processus B est également modifié de la même manière :

$$B = \mathsf{in}(c_0, \mathsf{senc}(\langle x'_p, x_0, z_{bool}, x_p \rangle, D)).\mathsf{out}(c_0, \mathsf{senc}(\langle \bot_U, \bot_V, z_{bool}, x_p \rangle, K))$$

Nous considérons donc les protocoles $\mathcal{P}$ et $\mathcal{Q}$ suivants :

$$\begin{aligned} \mathcal{P} =& \{B, F, V_P, !\,\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).P_1(c'_1), \ldots, !\,\mathsf{new}\, c'_n.\mathsf{out}(c_n, c'_n).P_n(c'_n),\\ & !\,\mathsf{new}\, c'_{n+1}.\mathsf{out}(c_{n+1}, c'_{n+1}).C(c'_{n+1}), !\,\mathsf{new}\, c'_{n+2}.\mathsf{out}(c_{n+2}, c'_{n+2}).V_0(c'_{n+2})\} \end{aligned}$$

$$\begin{aligned} \mathcal{Q} =& \{B, F, V_Q, !\,\mathsf{new}\, c'_1.\mathsf{out}(c_1, c'_1).P_1(c'_1), \ldots, !\,\mathsf{new}\, c'_n.\mathsf{out}(c_n, c'_n).P_n(c'_n),\\ & !\,\mathsf{new}\, c'_{n+1}.\mathsf{out}(c_{n+1}, c'_{n+1}).C(c'_{n+1}), !\,\mathsf{new}\, c'_{n+2}.\mathsf{out}(c_{n+2}, c'_{n+2}).V_0(c'_{n+2})\} \end{aligned}$$

$\mathcal{P}$ et $\mathcal{Q}$ sont conformes au système de types $(\mathcal{T}, \delta)$. Nous avons $\Sigma_\mathcal{P} = \Sigma_\mathcal{Q} = \{\bot_V, \bot_U, \bot_{nc}, \mathsf{true}, \mathsf{false}\}$ Nous admettons la proposition suivante :

Proposition 3.3. *Il existe un témoin quasiment typé de non-inclusion $\mathcal{P} \not\sqsubseteq_{at} \mathcal{Q}$ par rapport à $\Sigma^n \uplus (\Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q})$ si, et seulement si, l'instance de* PCP *considérée admet une solution de longueur au plus n.*

Remarquons que $\mathcal{P}$ est déterministe et $\mathcal{Q}$ est déterminé (voir section 1.4), donc la borne n'est pas calculable, y compris pour des protocoles déterminés.

3.4.4 Théories équationnelles

Avec ce dernier exemple, nous expliquons pourquoi il n'est pas possible de borner le nombre de constantes de l'attaquant dans le cadre des théories équationnelles, c'est-à-dire quand nous ne faisons plus de différence entre constructeurs et destructeurs, et quand nous remplaçons les règles $\mathsf{sdec}(\mathsf{senc}(x, y), y) \to x$ par des égalités $\mathsf{sdec}(\mathsf{senc}(x, y), y) = x$, ce qui a principalement trois conséquences :

1. Tous les termes peuvent être échangés sur le réseau.
2. Tous les symboles peuvent apparaître dans les processus.
3. Il n'y a plus de restriction sur les clés atomiques.

Pour cet exemple, une borne est calculable sur chaque protocole, mais il est possible de la faire croître arbitrairement. Nous n'avons pas énoncé de résultat de typage en présence d'une théorie équationnelle, donc nous supposons seulement que le système de types nous permet de forcer une variable à être instanciée par une donnée. Il est à noter que cette hypothèse n'est pas nécessairement vérifiée par tout système de type, par exemple si nous donnons le même type à $\mathsf{sdec}(\mathsf{senc}(a, k), k')$ et à $a = \mathsf{sdec}(\mathsf{senc}(a, k), k)$. Nous utiliserons exclusivement le chiffrement symétrique senc, le déchiffrement symétrique sdec, la paire et les projections.

Étant donné une liste ℓ de paires de constantes, nous construisons deux termes $t^P(\ell)$ et $t^Q(\ell)$ utilisant les symboles de fonction senc, sdec, et des constantes $c_1, \ldots, c_n$. Les termes $t^P(\ell)$ et $t^Q(\ell)$ sont égaux dès qu'une paire d'éléments de ℓ contient deux constantes identiques. Autrement dit, nous voulons que ces termes vérifient la propriété suivante :

Propriété 3.1. *Nous avons $t^P(\ell)\!\downarrow = t^Q(\ell)\!\downarrow$ si, et seulement si, il existe une paire (a, b) dans ℓ telle que $a = b$.*

Les termes $t^P(\ell)$ et $t^Q(\ell)$ sont définis récursivement comme suit.

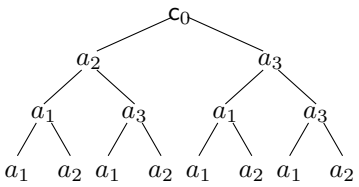
— $t^P(\ell) = \mathsf{sdec}(\mathsf{senc}(c_0, a), b)$ et $t^Q(\ell) = \mathsf{sdec}(\mathsf{senc}(c_0, b), a)$ quand $\ell = [(a, b)]$;

— Si $\ell = (a, b) :: \ell'$ avec ℓ' non vide, nous avons

$$t^X(\ell) = \mathsf{sdec}(\mathsf{senc}(\mathsf{c}_0, \mathsf{sdec}(\mathsf{senc}(a, t_1), t_2)), \mathsf{sdec}(\mathsf{senc}(b, t_1), t_2))$$

où m, t_1 et t_2 vérifient que $t^X(\ell') = \mathsf{sdec}(\mathsf{senc}(\mathsf{c}_0, t_1), t_2)$ et $X \in \{P, Q\}$.

Pour illustrer le résultat, le terme $t^P(\ell_0)$ pour $\ell_0 = [(a_2, a_3), (a_1, a_3), (a_1, a_2)]$ est décrit ci-dessous. Un sous-arbre dont la racine est étiquetée par n et dont les enfants sont t_1 et t_2 représente le terme $\mathsf{sdec}(\mathsf{senc}(n, t_1), t_2)$. Le terme $t^Q(\ell_0)$ est le même que $t^P(\ell_0)$ après permutation des étiquettes des feuilles.



Remarquons que $t^P(\ell_0) = t^Q(\ell_0)$ quand $a_1 = a_2$. Maintenant, si $a_1 = a_3$, nous obtenons $t^P(\ell_0)\!\downarrow = t^Q(\ell_0)\!\downarrow = \mathsf{sdec}(\mathsf{senc}(\mathsf{c}_0, a_2), a_3)$ et nous avons $t^P(\ell_0) = t^Q(\ell_0) = \mathsf{c}_0$ quand $a_2 = a_3$. Ce sont les seuls cas où $t^P(\ell_0)\!\downarrow = t^Q(\ell_0)\!\downarrow$. Plus généralement, il serait possible de montrer que $t^P(\ell)$ et $t^Q(\ell)$ satisfont la propriété 3.1.

Soient n et n_0 deux entiers. Nous nous appuyons sur ces termes pour construire deux processus P_n et Q_n tels que $(P_n; \emptyset; \emptyset; 0) \not\approx_{at} (Q_n; \emptyset; \emptyset; 0)$ par rapport à Σ^{n_0} si, et seulement si, $n_0 \geqslant n$. Ces processus sont construits de la manière suivante :

$$\begin{array}{rcl} P_n & = & \mathsf{in}(c, z_1) \ldots \mathsf{in}(c, z_n).\mathsf{out}(c, t^P(\ell)) \\ Q_n & = & \mathsf{in}(c, z_1) \ldots \mathsf{in}(c, z_n).\mathsf{out}(c, t^Q(\ell)) \end{array}$$

où ℓ est la liste de longueur $n(n-1)/2$ qui contient toutes les paires de la forme (z_i, z_j) avec $i < j$, et $z_1, \ldots, z_n$ sont des variables qui ne peuvent être instanciées que par des données. Remarquons que dans le cas où $n_0 < n$, dans chaque exécution, l'attaquant est forcé d'utiliser plusieurs fois la même constante, et donc les instances résultantes de $t^P(\ell)$ et $t^Q(\ell)$ seront égales (modulo la théorie équationnelle). Dans le cas où nous avons suffisamment de constantes distinctes, les instances résultantes de $t^P(\ell)$ et $t^Q(\ell)$ correspondent à des termes publics distincts. Donc, dans un tel cas, l'équivalence de protocoles ne tient pas.

Ainsi, cet exemple montre qu'en présence de théories équationnelles, la borne dépend du protocole considéré. La question de savoir si cette borne est calculable reste ouverte. En effet, si la construction des termes t^P et t^Q de taille arbitraire semble peu compatible avec l'utilisation d'un modèle typé, le même exemple s'adapte pour démontrer que la borne sur le nombre d'agents n'est pas calculable en présence de théories équationnelles [60].

3.5 Conclusion

Dans ce chapitre, nous avons démontré qu'il était possible de restreindre le nombre de constantes de l'attaquant à trois (en plus des constantes explicitement présentes dans le protocole de départ), en relâchant les contraintes de typage. De plus, nous pouvons nous contenter de rechercher des témoins de non-inclusion dont toutes les recettes sont simples. Concernant la borne sur le nombre de constantes, des résultats analogues existent dans un cadre beaucoup plus général pour le nombre d'agents [60] : en particulier, l'algèbre de processus peut contenir la construction *let* et de petites branches *else*. De plus, au prix d'une augmentation du nombre de constantes, il est possible d'autoriser les paires critiques, et en fait toutes les théories à constructeurs, y compris avec une notion de messages paramétrique (le résultat est donc vrai que l'on considère ou non les sortes et les contours). Les sortes ne sont utiles, à travers l'hypothèse des clés atomiques, que pour démontrer que l'attaquant ne perd pas de puissance en se limitant aux recettes simples.

Cependant, les résultats de ce chapitre sont relativement évidents dans un modèle non typé et sans sorte, et toute extension nécessiterait donc d'obtenir un résultat de typage adéquat auparavant. Il serait alors envisageable d'unifier les résultats portant sur les nonces, les agents et les constantes dans le cadre le plus général possible, d'autant que les hypothèses ont été justifiées par les contre-exemples que nous avons rappelés. La question de savoir s'il est possible de borner le nombre de constantes à utiliser pour un modèle typé et en présence de théories équationnelles reste également ouverte.

À l'issue de ce chapitre, et grâce au chapitre précédent, nous disposons d'un résultat de petite attaque (le théorème 3.1) qui nous permet de considérer un nombre fini de traces concrètes pour vérifier chaque protocole. Nous nous intéressons maintenant à la traduction de l'équivalence de protocoles sous forme de problèmes de planification, que nous allons présenter dans le chapitre suivant.

4 Problèmes de planification

Les problèmes de planification ont été introduits par Fikes et Nilsson [75] dans le but principal de guider les actions d'un robot pour organiser des objets dans l'espace et se déplacer, en tenant compte des relations de dépendance entre les actions. Dans ce contexte, le robot a besoin de connaître l'état du monde qui l'entoure et les effets de ses actions élémentaires sur le monde, afin de trouver quelle séquence de ces actions il doit combiner pour réaliser un objectif donné. Il s'agit donc d'une classe de problèmes génériques, dans laquelle le problème d'équivalence de protocoles sera traduit au chapitre 5.

Plus récemment, il a été démontré par Bylander [41] que les problèmes de planification étaient PSPACE-difficiles. Dans certains cas particuliers, si les préconditions sont exclusivement positives, et en bornant la taille des règles, la complexité diminue : le problème devient NP-complet ou polynomial. Deux méthodes efficaces de résolution de ces problèmes ont été proposées.

D'une part, Blum et Furst [39] ont introduit un graphe, dit graphe de planification, qui peut être calculé assez rapidement, et qui résume toutes les séquences possibles d'actions. Afin de gagner en efficacité, le graphe de planification est une surapproximation : certains faits sont représentés dans le graphe sans être réellement accessibles. Si les objectifs ne sont pas atteints dans le graphe, nous avons alors la garantie qu'ils ne sont pas atteignables. Sinon il faut s'assurer que ces objectifs soient réalisables en vérifiant chaque chemin possible sur le graphe.

D'autre part, Kautz et Selman [82] ainsi que Ernst, Millstein et Weld [72] traduisent le problème de planification sous forme de formules SAT. Kautz et Selman [81] ont ensuite associé cette approche avec la précédente : après la construction du graphe de planification, qui permet de limiter la taille de l'espace de recherche, l'accessibilité des objectifs est ensuite vérifiée par un codage SAT, comme le font également Armando et Compagna (dans [21]).

Ce chapitre commence par définir les problèmes de planification ainsi que les chemins et solutions de ces problèmes, avant de présenter informellement l'algorithme utilisé pour les résoudre en section 4.2. Cet algorithme sera décrit formellement en section 4.3, où sa correction sera également démontrée. Toujours dans cette section, un exemple montrera que le graphe réalise une surapproximation, ce qui justifiera d'utiliser un codage SAT pour éliminer les fausses solutions, ainsi qu'il sera expliqué en section 4.4.

4.1 Définitions

Nous définirons d'abord les systèmes de plannification comme des ensembles de faits munis de règles d'évolution, puis l'effet de séquences de règles, appelées chemins, dans la section 4.1.1. Comme nous étudions des systèmes potentiellement infinis, nous présenterons ensuite, dans la section 4.1.2, une hypothèse qui nous permettra de conserver des représentations finies. Enfin les problèmes de planifications seront définis dans la section 4.1.3, ce qui permettra de nous intéresser à leur résolution dès la section suivante. Les définitions formelles de système et de problème de planification sont adaptées de celles de Blum et Furst (voir [39]).

4.1.1 Système de planification

Un système de planification permet de représenter l'évolution d'un monde, envisagé comme ensemble de faits, sous l'action d'opérateurs. Pour le décrire, il suffit donc de définir l'ensemble des faits possibles, les règles qui permettent de changer d'état, ainsi qu'un point de départ. Plus formellement :

Définition 4.1. *On appelle* système de planification *la donnée d'un triplet* $\langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ *où*

- F *est un ensemble dénombrable (potentiellement infini) de formules atomiques closes, appelées* faits.
- $\mathcal{I} \subset \mathsf{F}$ *est un ensemble fini de faits initiaux.*
- Rule *est un ensemble de règles de la forme*

$$\mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$$

où $\mathsf{Pre}, \mathsf{Add}, \mathsf{Del}$ *sont des ensembles finis de faits, avec* $\mathsf{Add} \cap \mathsf{Del} = \emptyset$ *et* $\mathsf{Del} \subseteq \mathsf{Pre}$.

Par soucis de clarté, l'ensemble Del sera omis s'il est vide. On notera alors $\mathsf{Pre} \to \mathsf{Add}$.

Dans une règle $r = \mathsf{Pre} \to \mathsf{Add}; \mathsf{Del} \in \mathsf{Rule}$, $\mathsf{Pre}(r) = \mathsf{Pre}$ représente les préconditions de r, c'est-à-dire les faits qui doivent être présents pour que la règle puisse s'appliquer, $\mathsf{Add}(r) = \mathsf{Add}$ décrit les faits qui doivent être ajoutés à l'état du monde suite à l'application de cette règle et $\mathsf{Del}(r) = \mathsf{Del}$ ceux qui doivent être supprimés.

Plus précisément, considérant $S \subseteq \mathsf{F}$, la règle r est *applicable en* S si $\mathsf{Pre}(r) \subseteq S$, et l'on note alors $S \xrightarrow{r} S'$ où $S' = (S \cup \mathsf{Add}(r)) \setminus \mathsf{Del}(r)$.

Exemple 4.1. *Regardons d'abord un exemple très simple de système de planification* $\langle \mathsf{F}_0, \mathcal{I}_0, \mathsf{Rule}_0 \rangle$ *qui servira d'illustration dans la suite. Posons* $\mathsf{F}_0 = \{a, b, c, d\}$, $\mathcal{I}_0 = \{a\}$ *et* $\mathsf{Rule}_0 = \{rb, rc, rd\}$ *avec* $rb = a \to b$, $rc = a \to c$ *et* $rd = d \to \emptyset; d$. *Les règles* rb *et* rc *sont applicables en* $\mathcal{I}_0$ *car* $\mathsf{Pre}(rb) = \mathsf{Pre}(rc) = a \in \mathcal{I}_0$. *Au contraire,* rd *n'est pas applicable en* $\mathcal{I}_0$.

Exemple 4.2. *Nous pouvons également considérer un exemple plus proche des protocoles cryptographiques. On suppose que notre coffre fort ne s'ouvre que si l'on entre le mot de passe, à savoir les trois chiffrements* $\{a\}_k$,$\{b\}_k$ *et* $\{c\}_k$. *L'attaquant n'a pas la clé* k, *mais il connaît* a, b *et* c *et dispose de trois jetons qui lui permettent de chiffrer chacun l'une des constantes* a, b, c. *Nous utilisons* Jeton *pour les jetons, qui peuvent être consommés, et* att *pour les faits qui modélisent la connaissance de l'attaquant et qui ne peuvent pas disparaître, car l'attaquant n'oublie pas. Le système de planification qui correspond est* $\langle \mathsf{F}_1, \mathcal{I}_1, \mathsf{Rule}_1 \rangle$ *avec :*

- $\mathsf{F}_1 = \{\mathsf{att}(a), \mathsf{att}(b), \mathsf{att}(c), \mathsf{att}(\{a\}_k), \mathsf{att}(\{b\}_k), \mathsf{att}(\{c\}_k), \mathsf{Jeton}(1), \mathsf{Jeton}(2), \mathsf{Jeton}(3), \mathsf{att}(s)\}$, *où les faits de la forme* $\mathsf{att}(x)$ *représentent la connaissance de l'attaquant, les faits* $\mathsf{Jeton}(i)$ *(*$i \in \{1, 2, 3\}$*) représentent les jetons inutilisés et* s *est le secret caché dans le coffre fort.*
- $\mathcal{I}_1 = \{\mathsf{att}(a), \mathsf{att}(b), \mathsf{att}(c), \mathsf{Jeton}(1), \mathsf{Jeton}(2), \mathsf{Jeton}(3)\}$.
- $\mathsf{Rule}_1 = \{r(a,1), r(a,2), r(a,3), r(b,1), r(b,2), r(b,3), r(c,1), r(c,2), r(c,3), rs\}$ *avec :*

$$r(x,i) = \mathsf{att}(x), \mathsf{Jeton}(i) \to \mathsf{att}(\{x\}_k); \mathsf{Jeton}(i) \text{ pour tout } i \in \{1;2;3\} \text{ et tout } x \in \{a,b,c\}.$$
$$rs = \mathsf{att}(\{a\}_k), \mathsf{att}(\{b\}_k), \mathsf{att}(\{c\}_k) \to \mathsf{att}(s)$$

Les règles $r(x,i)$ *(pour* $i \in \{1,2,3\}$ *et* $x \in \{a,b,c\}$*) sont toutes applicables dans l'état initial* $\mathcal{I}_0$. *Par exemple,* $\mathsf{Pre}(r(a,1)) = \{\mathsf{att}(a), \mathsf{Jeton}(1)\}$ *est inclus dans* $\mathcal{I}_1$, *et*

$$\mathcal{I}_1 \xrightarrow{r(a,1)} S_1^a := \{\mathsf{att}(a), \mathsf{att}(b), \mathsf{att}(c), \mathsf{Jeton}(2), \mathsf{Jeton}(3), \mathsf{att}(\{a\}_k)\}$$

Dans S_1^a, seules les règles $r(x,2), r(x,3)$ avec $x \in \{a,b,c\}$ sont applicables. Les règles $r(a,2)$ et $r(a,3)$ consomment un jeton sans rien apprendre à l'attaquant, tandis que les quatre autres lui permettent d'obtenir un chiffrement supplémentaire.

Considérons maintenant l'état $S = \{\mathsf{att}(a), \mathsf{att}(b), \mathsf{att}(c), \mathsf{att}(\{a\}_k), \mathsf{att}(\{b\}_k), \mathsf{att}(\{c\}_k)\}$. Aucune des neuf règles de chiffrement $r(i,x)$ (avec $i \in \{1,2,3\}$ et $x \in \{a,b,c\}$) n'est applicable en S, car S ne contient aucun $\mathsf{Jeton}(i)$ et donc les préconditions des règles $r(i,x)$ ne sont pas remplies. Au contraire, la règle supplémentaire rs s'applique en S, car S contient $\mathsf{Pre}(rs) = \mathsf{att}(\{a\}_k), \mathsf{att}(\{b\}_k), \mathsf{att}(\{c\}_k)$. On a donc $S \xrightarrow{rs} S \cup \{\mathsf{att}(s)\}$.

Arrivés à ce point, l'application des règles en séquence semble naturelle. Cependant, les règles peuvent également être appliquées simultanément. En effet, supposons que l'on dispose de deux règles r et r' avec $\mathsf{Del}(r) = \mathsf{Del}(r') = \emptyset$. Alors, dans tout état S tel que $\mathsf{Pre}(r) \cup \mathsf{Pre}(r') \subseteq S$, on a $S \xrightarrow{r} S \cup \mathsf{Add}(r) \xrightarrow{r'} S \cup \mathsf{Add}(r) \cup \mathsf{Add}(r')$ et $S \xrightarrow{r'} S \cup \mathsf{Add}(r') \xrightarrow{r} S \cup \mathsf{Add}(r) \cup \mathsf{Add}(r')$. Il est donc possible d'appliquer r et r' en une seule étape, puisque le résultat obtenu ne dépend pas de l'ordre des opérations.

Formellement, soient un système de plannification $\langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ et p un ensemble de règles $\{r_1, \ldots, r_k\}$. p est un *pas* si c'est un singleton ou si $\mathsf{Del}(r_i) = \emptyset$ pour chaque i. On note alors $\mathsf{Pre}(p) = \cup_i \mathsf{Pre}(r_i)$, $\mathsf{Add}(p) = \cup_i \mathsf{Add}(r_i)$. $\mathsf{Del}(p) = \mathsf{Del}(r_1)$ si r est le singleton $\{r_1\}$ et $\mathsf{Del}(p) = \emptyset$ si p n'est pas un singleton. Un pas p est *applicable* dans un état S si $\mathsf{Pre}(p) \subseteq S$. On note alors $S \xrightarrow{p} S'$ avec $S' = (S \setminus \mathsf{Del}(p)) \cup \mathsf{Add}(p)$.

Un *chemin de longueur n* est une séquence de pas $c = c_1, \ldots, c_n$. Soit c un chemin. La longueur de c est notée $\mathsf{len}(c)$. On dit que c est *applicable* dans un état $S_0 \subseteq \mathsf{F}$ donné si il existe des états $S_1, \ldots, S_n$ tels que pour chaque i $(1 \leqslant i \leqslant n)$, $\mathsf{Pre}(c_i) \subseteq S_{i-1}$ et $S_i = (S_{i-1} \cup \mathsf{Add}(c_i)) \setminus \mathsf{Del}(c_i)$.

On note alors $S_0 \xrightarrow{c} S_n$, et on dira aussi que S_n est l'*état résultant* de c sur S_0.

On dit qu'un chemin est *linéaire* si chaque c_i est un singleton. Il est clair qu'il existe un chemin entre deux états S et S' si, et seulement si, il existe un chemin linéaire.

Exemple 4.3. *Revenons à l'exemple 4.1 et au système $\langle \mathsf{F}_0, \mathcal{I}_0, \mathsf{Rule}_0 \rangle$. La séquence rb, rc est un chemin linéaire de $\mathcal{I}_0$ à $S = \{a,b,c\}$. On démontrerait facilement qu'une séquence s de règles rb et rc, appliquée en $\mathcal{I}_0$, aboutirait à un état $S' \subseteq S$: puisqu'aucune règle ne permet d'ajouter d, il est facile de voir que rd ne peut s'appliquer dans un état obtenu à partir de $\mathcal{I}_0$ qui ne contient pas d. Ainsi, rd n'est applicable dans aucun état atteint à partir de l'état initial.*

Par ailleurs, comme les règles rb et rc n'effacent rien, le chemin $\{rb, rc\}$ de longueur 1 est un chemin de $\mathcal{I}_0$ à S.

Exemple 4.4. *Reprenons le système $\langle \mathsf{F}_1, \mathcal{I}_1, \mathsf{Rule}_1 \rangle$ de l'exemple 4.2. La règle rs est la seule à ne rien effacer ($\mathsf{Del}(rs) = \emptyset$), donc on ne peut pas mettre de règles en parallèle. La séquence des règles $r(a,1), r(b,2), r(c,3), rs$ est un chemin linéaire de $\mathcal{I}_0$ à S_f avec :*

$$S_f = \{\mathsf{att}(a), \mathsf{att}(b), \mathsf{att}(c), \mathsf{att}(\{a\}_k), \mathsf{att}(\{b\}_k), \mathsf{att}(\{c\}_k), \mathsf{att}(s)\}$$

4.1.2 Systèmes bornés

Soit $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ un système de planification. Nous n'avons pas exclu jusqu'ici que F et Rule soient des ensembles infinis. Néanmoins, pour pouvoir effectuer des calculs sur un système de planification, une hypothèse supplémentaire sera nécessaire.

Ainsi, nous disons que Π est un système de planification *borné* si l'ensemble des règles applicables dans tout état fini $S \subset \mathsf{F}$ est lui-même fini et si l'on dispose d'un oracle de planification $\mathcal{O}$, c'est-à-dire d'une fonction qui, étant donné S (fini), renvoie, en temps fini, l'ensemble $\mathcal{O}(S)$ des règles applicables *utiles*, c'est-à-dire de la forme $\mathsf{Pre} \rightarrow \mathsf{Add}; \mathsf{Del}$ avec $\mathsf{Add} \cup \mathsf{Del} \neq \emptyset$. Plus formellement, l'oracle est une fonction respectant la propriété suivante :

Propriété 4.1. *Soit $\langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ un système de planification, et $\mathcal{O}$ une fonction qui à toute partie finie S de F associe une partie finie $\mathcal{O}(S)$ de Rule. La fonction $\mathcal{O}$ vérifie la* propriété de l'oracle de planification *si, pour tout ensemble fini $S \subseteq \mathsf{F}$, $\mathcal{O}(S)$ est l'ensemble des règles applicables utiles dans S.*

Sous ces conditions, nous pouvons montrer que si l'on considère des chemins de longueur bornée, alors le nombre d'états accessibles est fini, et chacun de ces états est lui-même fini.

Fait 4.1. *Soient $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ un système de planification borné, et k un entier. Soit $S_0 \subseteq \mathsf{F}$ un ensemble fini. Soit $\Gamma_k = \{c \mid S_0 \xrightarrow{c} S$ avec $\mathsf{len}(c) = k\}$. Γ_k est fini, et pour tout $c \in \Gamma_k$, l'état S tel que $S_0 \xrightarrow{c} S$ est fini.*

Démonstration. Procédons par récurrence sur k. Si $k = 0$, le résultat est évident. Il reste donc à traiter l'hérédité. Soit $k > 0$ un entier tel que Γ_{k-1} soit fini, et que les états résultants de ces chemins soient eux-mêmes finis.

Soit $c \in \Gamma_k$. $c = c_1.c_2$ où c_1 est de longueur $k-1$. Soient S_1 l'état résultant du chemin c_1 sur S_0 et S_2 l'état résultant de c sur S_1. S_1 est fini par hypothèse d'induction, donc $\mathcal{O}(S_1)$ existe et est fini. Comme c_2 est un ensemble de règles applicables dans S_1, $c_2 \subseteq \mathcal{O}(S_1)$ et c_2 est donc fini. D'où $c \in \{c_1.c_2 \mid c_1 \in \Gamma_{k-1}$ et $c_2 \subseteq \mathcal{O}(S_1)\}$ pour tout $c \in \Gamma_k$. Autrement dit, $\Gamma_k \subseteq \{c_1.c_2 \mid c_1 \in \Gamma_{k-1}$ et $c_2 \subseteq \mathcal{O}(S_1)\}$ et donc Γ_k est fini.

De plus, on a :

$$\begin{aligned} S_2 &= (S_1 \cup \mathsf{Add}(c_2)) \setminus \mathsf{Del}(c_2) \\ &\subseteq S_1 \cup (\cup_{r \in c_2} \mathsf{Add}(r)) \end{aligned}$$

Comme, pour toute règle $r \in \mathsf{Rule}$, $\mathsf{Add}(r)$ est fini, S_2 est fini.

Nous avons ainsi démontré que les chemins de longueur k sont en nombre fini, et que les états résultants sont eux-mêmes finis. □

Dans la suite, il sera démontré que, pour un nombre borné des sessions d'un protocole cryptographique, il suffit de considérer des chemins bornés dans des systèmes bornés. Ainsi, le nombre d'états accessibles restera toujours fini.

4.1.3 Problèmes de planification

Après la description des systèmes de planification, qui fournissent une représentation de l'état d'un monde et de ses changements, ainsi que les chemins qui permettent de passer d'un état à un autre, il reste à définir quelles questions poser sur ces systèmes. Étant donné un ensemble d'objectifs Θ, il s'agit de savoir s'il existe un chemin de l'état initial jusqu'à un état contenant Θ.

Définition 4.2. *On appelle* problème de planification *la donnée d'une paire* $\langle \Pi, \Theta \rangle$ *où* $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ *est un système de planification et* $\Theta \subseteq \mathsf{F}$ *un ensemble fini de faits appelés* buts *(ou* objectifs*).*

Soit $S \subseteq \mathsf{F}$*. Un chemin* c *de* $\mathcal{I}$ *à* S *est une* solution *de* $\langle \Pi, \Theta \rangle$ *si* $\Theta \subseteq S$*. Nous dirons aussi que* c *est un* plan *de* Θ *dans* Π*.*

Par extension, nous dirons qu'un problème de planification est *borné* quand le système de planification sous-jacent est lui-même borné.

Les buts doivent seulement être inclus dans l'état final, et ne coïncident donc pas nécessairement avec lui. En effet, dans le cas qui nous occupe, il sera question de savoir si l'attaquant parvient à faire la différence entre deux protocoles, indépendamment des connaissances qu'il aurait pu acquérir par ailleurs.

Notons également qu'il est possible de résoudre les problèmes de planification par composition. Plus précisément, soient $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ et $\Pi' = \langle \mathsf{F}', \Theta, \mathsf{Rule}' \rangle$ deux systèmes de planification avec $\Theta \subseteq \mathsf{F} \cap \mathsf{F}'$, et $\Theta' \subset \mathsf{F}'$ un ensemble d'objectifs. Si c est une solution de $\langle \Pi, \Theta \rangle$ et c' est une solution de $\langle \Pi', \Theta' \rangle$, alors le chemin $c.c'$ est une solution de $\langle \Pi'', \Theta' \rangle$ où $\Pi'' = \langle \mathsf{F} \cup \mathsf{F}', \mathcal{I}, \mathsf{Rule} \cup \mathsf{Rule}' \rangle$.

Exemple 4.5. *Dans l'exemple 4.3, nous avions vu que, dans le système* $\Pi_0 = \langle \mathsf{F}_0, \mathcal{I}_0, \mathsf{Rule}_0 \rangle$*,* $\{rb, rc\}$ *était un chemin de l'état initial* $\mathcal{I}_0$ *à l'état* $S = \{a, b, c\}$*. Le problème de planification* $\langle \Pi_0, \{b, c\} \rangle$ *admet donc une solution* $\{ra, rb\}$*.*

Comme tous les états accessibles à partir de l'état initial $\mathcal{I}_0$ *sont inclus dans* S*, et comme* S *ne contient pas* d*, le problème de planification* $\langle \Pi_0, \{d\} \rangle$ *n'admet donc aucune solution.*

Exemple 4.6. *L'exemple 4.2 modélise un coffre-fort contenant un document secret* s*. Le problème de planification* $\langle \Pi_1, \{\mathsf{att}(s)\} \rangle$ *correspond donc à la question de savoir si l'attaquant a accès à* s*. D'après l'exemple 4.4, il existe un chemin* c *de* $\mathcal{I}_0$ *à* S_f *avec* $\mathsf{att}(s) \in S_f$*, l'attaquant peut ouvrir le coffre. Cependant, avec un jeton de moins (par exemple* $\mathsf{Jeton}(3)$*), l'attaquant ne peut plus chiffrer que deux des trois constantes* a, b, c *et ne peut donc pas ouvrir le coffre. Le système de planification correspondant est* $\Pi_1' = \langle \mathsf{F}_1, \mathcal{I}_1 \setminus \{\mathsf{Jeton}(3)\}, \mathsf{Rule}_1 \rangle$*. Ainsi, le problème* $\langle \Pi_1', \{\mathsf{att}(s)\} \rangle$ *n'admet pas de solution.*

Maintenant que nous avons défini les problèmes de planification, nous devons les résoudre. Quand nous traduirons le problème d'équivalence de protocoles, nous nous demanderons si l'attaquant peut arriver à un état où les deux protocoles ont déjà eu un comportement distinct. En d'autres termes, les protocoles seront équivalents si le problème *n'*a *pas* de solution.

4.2 Survol informel

Le graphe de planification est une technique classique de résolution pour les problèmes de planification, introduite par Blum et Furst [39] et qu'Armando et Compagna [21] ont appliqué les premiers à la vérification de protocoles cryptographiques pour les propriétés de trace. À l'origine, elle ne s'applique qu'aux cas où l'ensemble des faits est fini. Dans cette thèse, nous l'adaptons au cas des systèmes bornés, c'est-à-dire pour lesquels le nombre de règles applicables est fini à tout moment. Nous n'avons trouvé aucune mention de cette variante du problème dans la littérature.

Son rôle est à la fois de calculer une surapproximation des états accessibles et de résumer les dépendances entre faits et règles pour guider le codage en formule SAT. Si le calcul du graphe s'arrête sans que les objectifs soient présents dans l'état final, on peut être sûr qu'il ne seront pas atteints. Si, au contraire, nos buts sont présents à la fin, il faut encore vérifier qu'ils sont effectivement accessibles par un chemin. Pour cette dernière étape, les informations présentes dans le graphe servent à coder une formule SAT qui sera ensuite vérifiée par un solveur.

Le graphe de planification que nous calculons est une version compacte de celui qui est défini par Blum et Furst (dans [39]). Plus précisément, l'algorithme de Blum et Furst n'utilise pas les propriétés de monotonie, c'est-à-dire que certains ensembles soient croissants, et les recopie à chaque étape. Une telle utilisation de la mémoire est inutile, car il suffit de garder une copie de chaque élément d'un ensemble croissant. Cependant, plutôt qu'une preuve que ces deux algorithmes donnent des résultats similaires, ce chapitre montrera uniquement la correction du nôtre.

Cette section décrit informellement le graphe, qui sera défini, ainsi que ses propriétés, dans la section 4.3. Les nœuds du graphe représentent des faits ou des règles, tandis que les arêtes indiquent les liens entre ces nœuds. Par exemple, on dessine des arêtes entre une règle et chacune de ses préconditions. Une relation spéciale, dite d'*exclusion mutuelle* (ou plus simplement *mutex*), signale que deux faits (ou deux règles) ne peuvent pas être obtenus (ou s'appliquer) simultanément.

4.2.1 Algorithme

Soit $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ un système de planification borné. Le graphe associé à Π contient deux catégories de nœuds, qui correspondent aux faits et aux règles, et deux catégories d'arêtes. Une partie des arêtes représente les relations entre faits et règles (préconditions, ajouts et suppression) tandis que l'autre partie représente les exclusions entre des nœuds de même catégorie.

Ce graphe se calcule par étapes, ainsi que la fonction node qui associe des nœuds aux faits et aux règles représentés dans le graphe. Chaque nœud, qu'il représente un fait ou une règle, est indexé par l'étape à laquelle à laquelle il a été ajouté au graphe. Cette fonction node sera explicitée au fur et à mesure. On peut donc voir $\mathsf{node}(f)$ comme une manière succinte désigné le nœud du graphe associé à f.

Au départ, les seuls nœuds du graphe sont les faits initiaux indexés par leur date d'apparition 0 (f_0 pour $f \in \mathcal{I}_0$). On notera donc $\mathsf{node}(f) = f_0$. Puis, à l'étape i, à partir des faits du graphe, on fait appel à l'oracle. Comme les faits sont en nombre fini, l'appel à l'oracle est bien défini. Il renvoie l'ensemble fini des règles applicables. Nous ôtons de ces règles toutes celles qui sont déjà présentes dans le graphe (rappelons que les règles sont concrètes, et donc qu'une règle identique à une règle existante n'apporte rien de nouveau).

Parmi les règles restantes, celles dont plusieurs préconditions s'excluent mutellement sont éliminées. Pour les autres règles r, r_i est ajouté au graphe. On notera $\mathsf{node}(r) = r_i$.

On ajoute ensuite des arêtes pour représenter les relations entre les faits et les règles, par exemple des arêtes $\mathsf{node}(f) \longrightarrow_{pre} \mathsf{node}(r)$ pour chaque $f \in \mathsf{Pre}(r)$. On ajoute également les nouveaux nœuds f_i au graphe (et alors $\mathsf{node}(f) = f_i$). Remarquons que dans tous les cas, on n'obtient qu'un nombre fini de nouveaux faits puisque les nouvelles règles sont en nombre fini.

On ajoute ensuite les mutex comme expliqué en 4.2.2, puis on inspecte toutes les exclusions mutuelles. On indique la date de fin de celles qui ne sont plus valides par $n \overset{mutex}{\longleftrightarrow}_j^i n'$.

Si aucune mutex n'a été enlevée, et si aucun fait et aucune règle n'a été ajouté au cours de l'étape, l'algorithme s'arrête. Il s'arrête également si l'on a complété le bon nombre d'étapes. Sinon, on commence l'étape suivante.

4.2.2 Exclusions mutuelles

Les exclusions mutuelles sont de deux sortes : soit elles signalent que deux règles ne peuvent structurellement pas avoir lieu en même temps, et dans ce cas elles sont définitives ; soit elles existent comme conséquences des premières, et peuvent évoluer avec le graphe : on parle de mutex provisoires.

Deux règles représentées par des nœuds n et m sont définitivement en mutex lorsque les préconditions ou les ajouts de l'une sont supprimés par l'autre. On note alors $n \overset{mutex}{\longleftrightarrow}_i^\infty m$.

Ces exclusions se propagent et créent les exclusions mutuelles provisoires. Ainsi, deux règles sont en mutex dès que des nœuds représentant leurs préconditions sont eux-même en mutex. De même, des faits s'excluent mutellement dès que toutes les règles qui les ajoutent s'excluent mutuellement. Chacune de ces exclusions peut prendre fin. Par exemple, dans le cas des faits, si l'on découvre de nouvelles règles qui les ajoutent sans être elles-mêmes mutellement exclusives. On note $n \overset{mutex}{\longleftrightarrow}_i^+ m$, où i est l'étape à laquelle cette exclusion commence. Si ces mutex ne sont plus valables à partir de l'étape j, on note $n \overset{mutex}{\longleftrightarrow}_i^j m$.

4.3 Algorithme

Cette section décrit formellement l'algorithme du graphe de planification et démontre certaines de ses propriétés.

La fonction L_Π^i, présentée dans la section 4.3.1, constitue le cœur de l'algorithme et correspond à une itération. L'algorithme complet pourra en être déduit en section 4.3.2. Les propriétés du graphe seront présentées dans les sections suivantes.

Le graphe sera représenté comme (F, R, A, M) où F, R sont des ensembles de noeuds et A, M des ensembles de règles. F représente les faits dans le graphe, et R les règles. Les nœuds de F et R sont indexés par un entier qui correspond à leur date d'apparition dans le graphe. A contient les arrêtes entre F et R, tandis que M est l'ensemble des mutex. Les arêtes de A ($\longrightarrow_{pre}$, $\longrightarrow_{add}$,$\longrightarrow_{del}$) sont orientées uniquement pour marquer la différence entre les faits et les règles, et les mutex ($\overset{mutex}{\longleftrightarrow}$) sont des arêtes non-orientées. Comme expliqué en section 4.2.1, node associe les faits et les règles aux nœuds qui correspondent. S'il y a un fait f_k pour une certaine itération k dans F, alors $\mathsf{node}(f) = f_k$.

Le graphe contiendra des règles $\mathsf{nop}(f)$ (de la forme $f \rightarrow f$) pour chaque fait f. Elles ne correpondent pas à des règles du problème, dont elles ne modifient pas la nature. Elles servent uniquement d'intermédiaires de calcul pour la propagation des mutex des faits aux règles puis des règles aux faits.

4.3.1 Fonction itérée L^i_Π

Soient $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ un système de planification borné et i un entier. Le graphe de planification associé à Π est un quadruplet (F, R, A, M) où :

— $F \subseteq \{f_k \mid f \in \mathsf{F}, k \in \mathbb{N}\}$ est un ensemble fini.

— $R \subseteq \{r_k \mid r \in \mathsf{Rule}, k \in \mathbb{N}\}$ est un ensemble fini.

— $A \subseteq \{n \longrightarrow_{pre} m \mid n \in F, m \in R\} \cup \{n \longrightarrow_{add} m \mid n \in R, m \in F\} \cup \{n \longrightarrow_{del} m \mid n \in R, m \in F\}$

— $M \subseteq \{n \xleftarrow{mutex}{}^{\star}_{k} m \mid (n,m) \in F^2 \cup R^2, \star \in \{+, \infty\} \cup \mathbb{N}\}$.

La fonction L^i_Π se calcule comme suit, à partir de son entrée $(F, R, A, M, \mathsf{new}_f)$, où (F, R, A, M) est un graphe et new_f un ensemble fini de faits. On suppose que la fonction node a été partiellement définie et on l'étendra au cours du calcul de L^i_Π.

1. On initialise $\mathsf{new}_r := \emptyset$, et pour chaque $f \in \mathsf{new}_f$, on fait $\mathsf{new}_r := \mathsf{new}_r \cup \{\mathsf{nop}(f)\}$.
2. On fait $\mathsf{new}_f := \emptyset$. On pose $L_f := \{f \mid \mathsf{node}(f) \in F\}$ et $L_r := \{r \mid \mathsf{node}(r) \in R\}$.
3. On fait appel à l'oracle sur L_f : $\mathcal{O}(L_f)$ est l'ensemble des règles applicables dans L_f.
4. On considère chaque règle $r \in \mathcal{O}(L_f) \setminus L_r$. On vérifie s'il existe une paire de faits $f, f' \in \mathsf{Pre}(r)$ tels que, pour un certain m :
$$\mathsf{node}(f) \xleftarrow{mutex}{}^{+}_{m} \mathsf{node}(f') \in M$$
Si il existe une telle paire, on élimine la règle. Si la règle n'a pas été éliminée, on fait $\mathsf{new}_r := \mathsf{new}_r \cup \{r\}$.
5. (Nouveaux faits) Pour chaque règle $r \in \mathsf{new}_r$, pour chaque $f \in \mathsf{Add}(r)$, si $\mathsf{node}(f)$ n'est pas défini, alors on fait $\mathsf{new}_f := \{f\} \cup \mathsf{new}_f$.
6. On ajoute les nœuds que l'on a calculés :
   — Pour chaque $r \in \mathsf{new}_r$, on fait $R := \{r_i\} \cup R$ et on définit $\mathsf{node}(r) = r_i$.
   — Pour chaque $f \in \mathsf{new}_f$, on fait $F := \{f_i\} \cup F$ et on définit $\mathsf{node}(f) = f_i$.
7. Pour chaque $r \in \mathsf{new}_r$:
   — Pour chaque $f \in \mathsf{Pre}(r)$ on fait $A := A \cup \{\mathsf{node}(f) \longrightarrow_{pre} \mathsf{node}(r)\}$.
   — Pour chaque $f \in \mathsf{Add}(r)$ on fait $A := A \cup \{\mathsf{node}(r) \longrightarrow_{add} \mathsf{node}(f)\}$.
   — Pour chaque $f \in \mathsf{Del}(r)$ on fait $A := A \cup \{\mathsf{node}(r) \longrightarrow_{del} \mathsf{node}(f)\}$.
8. (Interférences) Pour chaque $r \in \mathsf{new}_r$, pour chaque $\mathsf{node}(r')$ dans R :
   — Pour chaque $f \in \mathsf{Del}(r)$, si $\mathsf{node}(r') \longrightarrow_{add} \mathsf{node}(f) \in A$ ou $\mathsf{node}(f) \longrightarrow_{pre} \mathsf{node}(r') \in A$, on fait $M := M \cup \{\mathsf{node}(r) \xleftarrow{mutex}{}^{\infty}_{i} \mathsf{node}(r')\}$.
   — Pour chaque $f \in \mathsf{Del}(r')$, si $\mathsf{node}(r) \longrightarrow_{add} \mathsf{node}(f) \in A$ ou $\mathsf{node}(f) \longrightarrow_{pre} \mathsf{node}(r) \in A$, on fait $M := M \cup \{\mathsf{node}(r) \xleftarrow{mutex}{}^{\infty}_{i} \mathsf{node}(r')\}$.
9. (Besoins concurrents) Pour chaque $r \in \mathsf{new}_r$, pour chaque $\mathsf{node}(r')$ dans R, pour chaque $f \in \mathsf{Pre}(r)$, pour chaque $f' \in \mathsf{Pre}(r')$, si $f \xleftarrow{mutex}{}^{+}_{m} f' \in M$ pour un certain m, alors $M := M \cup \{r \xleftarrow{mutex}{}^{+}_{i} r'\}$.
10. Pour chaque $f \in \mathsf{new}_f$, pour chaque $\mathsf{node}(f') \in F$, si pour toute paire $\mathsf{node}(r), \mathsf{node}(r')$ telle que $\mathsf{node}(r) \longrightarrow_{add} \mathsf{node}(f) \in A$ et $\mathsf{node}(r') \longrightarrow_{add} \mathsf{node}(f') \in A$ on a $\mathsf{node}(r) \xleftarrow{mutex}{}^{+}_{m} \mathsf{node}(r') \in M$ ou $\mathsf{node}(r) \xleftarrow{mutex}{}^{\infty}_{m} \mathsf{node}(r') \in M$ pour un certain m, alors on note $M := M \cup \{\mathsf{node}(f) \xleftarrow{mutex}{}^{+}_{i} \mathsf{node}(f')\}$.

11. On regarde les mutex provisoires dans M : notons $m = n_1 \xleftarrow{mutex}{}^{+}_{\ell} n_2 \in M$.
    - Si n_1 et n_2 sont des faits, si il existe une paire $\mathsf{node}(r), \mathsf{node}(r')$ telle que $\mathsf{node}(r) \longrightarrow_{add} \mathsf{node}(f) \in A$ et $\mathsf{node}(r') \longrightarrow_{add} \mathsf{node}(f') \in A$ mais $\mathsf{node}(r)$ et $\mathsf{node}(r')$ ne sont pas en mutex (c'est-à-dire $\mathsf{node}(r) \xleftarrow{mutex}{}^{+}_{p} \mathsf{node}(r') \notin M$ et $\mathsf{node}(r) \xleftarrow{mutex}{}^{\infty}_{p} \mathsf{node}(r') \notin M$ pour tout p), on remplace m par $m' = n_1 \xleftarrow{mutex}{}^{i-1}_{\ell} n_2$, c'est-à-dire $M := (M \setminus \{m\}) \cup \{m'\}$.
    - Si n_1 et n_2 sont des règles, pour chaque $f \in \mathsf{Pre}(r)$, pour chaque $f' \in \mathsf{Pre}(r')$, si $f \xleftarrow{mutex}{}^{+}_{p} f' \notin M$ pour un certain p, on remplace m par $m' = n_1 \xleftarrow{mutex}{}^{i-1}_{\ell} n_2$, c'est-à-dire $M := (M \setminus \{m\}) \cup \{m'\}$.
12. On renvoie $(F, R, A, M, \mathsf{new}_f)$.

Á l'étape 8, un exclusion mutuelle est déclarée si deux règles ne peuvent pas être actives en même temps, soit parce que l'une efface les préconditions de l'autre (et alors celle dont les préconditions sont effacées n'est plus applicable) soit parce que les deux ont des effets contradictoires. L'étape 9 permet de propager les exclusions mutuelles aux règles à partir des faits, quand les préconditions de deux règles ne peuvent pas être satisfaites simultanément. L'étape 10 propage les exclusions aux faits à partir des règles, quand deux faits ne peuvent pas être obtenus simultanément.

La première chose à démontrer est que cet algorithme est bien défini. Plus précisément :

Lemme 4.1. *Soient (F^-, R^-, A^-, M^-) un graphe fini et new_f^- un ensemble fini de faits. Le calcul de $(F^+, R^+, A^+, M^+, \mathsf{new}_f^+) = \mathsf{L}^i_\Pi(F^-, R^-, A^-, M^-, \mathsf{new}_f^-)$ s'arrête et $(F^+, R^+, A^+, M^+, \mathsf{new}_f^+)$ est fini.*

Démonstration. La seule étape difficulté potentielle consiste à démontrer que l'appel à l'oracle de l'étape 3 est bien défini. À l'étape 2, L_f est fini car F^- est fini. Donc l'appel à l'oracle $\mathcal{O}(L_f)$ est bien défini et son résultat est fini.

□

Cette fonction possède plusieurs propriétés de monotonie qui seront utiles.

Lemme 4.2. *Soient (F^-, R^-, A^-, M^-) un graphe et new_f^- un ensemble fini de faits. Soit i un entier. Posons $(F^+, R^+, A^+, M^+, \mathsf{new}_f^+) = \mathsf{L}^i_\Pi(F^-, R^-, A^-, M^-, \mathsf{new}_f^-)$. Alors :*

- $F^- \subseteq F^+$
- $R^- \subseteq R^+$
- $A^- \subseteq A^+$
- *Si $(F^-, R^-, A^-, M^-) = (F^+, R^+, A^+, M^+)$, alors pour tout $j \geqslant i$*

$$\mathsf{L}^j_\Pi(F^-, R^-, A^-, M^-, \mathsf{new}_f^-) = (F^-, R^-, A^-, M^-, \mathsf{new}_f^-) = (F^-, R^-, A^-, M^-, \emptyset)$$

Démonstration. Les seules modifications de F, R et A interviennent aux étapes 6 et 7. On a donc clairement $F^- \subseteq F^+$, $R^- \subseteq R^+$ et $A^- \subseteq A^+$. Supposons que $(F^-, R^-, A^-, M^-) = (F^+, R^+, A^+, M^+)$. En particulier, d'après l'étape 6, new_f^+ est vide (les étapes suivantes ne modifient pas new_f) et new_r est vide. Comme new_r ne fait que croître entre l'étape 1 et l'étape 6, $\mathsf{new}_r = \emptyset$ à l'étape 6 implique $\mathsf{new}_r = \emptyset$ après l'étape 1 et donc $\mathsf{new}_f^- = \emptyset$.

On a donc bien $\mathsf{L}^i_\Pi(F^-, R^-, A^-, M^-, \mathsf{new}_f^-) = (F^-, R^-, A^-, M^-, \mathsf{new}_f^-) = (F^-, R^-, A^-, M^-, \emptyset)$. Montrons que c'est le cas pour tout $j \geqslant i$.

Les étapes 1 à 4 ne dépendent pas de j, donc on arrive à l'étape 6 avec les mêmes valeurs de $F, R, A, M, \mathsf{new}_f, \mathsf{new}_r$ que pour i. En particulier, $\mathsf{new}_f = \mathsf{new}_r = \emptyset$. Ainsi, on ne fait rien aux

étapes 6 à 10. Les conditions de l'étape 11 ne dépendent pas de i, et elles sont fausses pour i (car les instructions correspondantes ne s'exécutent pas, étant donné que $M^+ = M^-$) donc elles sont fausses aussi pour j et les instructions correspondantes ne s'exécutent pas. Ainsi, on arrive à la dernière étape avec $(F, R, A, M, \mathsf{new}_f) = (F^-, R^-, A^-, M^-, \mathsf{new}_f^-)$ quelque soit la valeur de $j \geqslant i$. C'est ce qu'il restait à démontrer. □

4.3.2 Algorithme complet

Soient $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ un système de planification borné et k un entier représentant la taille maximale du plan.

Maintenant que la fonction itérée L^i_Π a été définie, la structure de l'algorithme est relativement simple.

(A) On initialise $(F, R, A, M) := (\emptyset, \emptyset, \emptyset, \emptyset)$, et $\mathsf{new}_f = \emptyset$, $\mathsf{new}_r = \emptyset$.

(B) Pour chaque $f \in \mathcal{I}$, on fait $F := F \cup \{f_0\}$, $\mathsf{new}_f = \mathsf{new}_f \cup \{f\}$ et on pose $\mathsf{node}(f) = f_0$.

(C) Si $k = 0$, on s'arrête et on renvoie $(F, \emptyset, \emptyset, \emptyset)$. Sinon, on pose $i := 1$.

(D) On fait $(F^-, R^-, A^-, M^-) := (F, R, A, M)$. On calcule :

$$(F, R, A, M, \mathsf{new}_f) = \mathsf{L}^i_\Pi(F, R, A, M, \mathsf{new}_f)$$

(E) (Saturation) Si $(F, R, A, M) = (F^-, R^-, A^-, M^-)$, on s'arrête et on renvoie (F, R, A, M).

(F) Si $i = k$, on s'arrête et on renvoie (F, R, A, M). Sinon, $i := i+1$ et on retourne à l'étape (D).

Nous appelons *graphe de planification* associé à Π, noté $\mathsf{PG}_k(\Pi)$, le résultat de cet algorithme. Si $(F, R, A, M) = \mathsf{PG}_k(\Pi)$ et $i \leqslant k$, on notera $F_i = \{f_j \in F \mid j \leqslant i\}$, $R_i = \{r_j \in R \mid j \leqslant i\}$ et A_i les arêtes entre ces nœuds, c'est-à-dire

$$\begin{aligned} A_i = &\{f \longrightarrow_{pre} r \text{ avec } f \in F_{i-1} \text{ et } r \in R_i\} \\ &\cup \{r \longrightarrow_{add} f \text{ avec } f \in F_i \text{ et } r \in R_i\} \\ &\cup \{r \longrightarrow_{del} f \text{ avec } f \in F_i \text{ et } r \in R_i\} \end{aligned}$$

$$\begin{aligned} M_i = &\{n \xleftarrow{mutex}{}^{q}_{p} m \text{ avec } n, m \in F \cup R \text{ et } p \leqslant i \leqslant q\} \\ &\cup \{n \xleftarrow{mutex}{}^{\star}_{p} m \text{ avec } n, m \in F \cup R, p \leqslant i \text{ et } \star \in \{+, \infty\}\} \end{aligned}$$

D'après le lemme 4.2, on a $F_i \subseteq F_{i+1}$, $R_i \subseteq R_{i+1}$ et $A_i \subseteq A_{i+1}$. Intuitivement, le graphe (F_i, R_i, A_i, M_i) correspond à toutes éléments actifs à l'étape i : tous les faits et les règles qui ont été atteints, et toutes les exclusions mutuelles qui sont déjà apparues mais n'ont pas été supprimées. Le graphe calculé par Blum et Furst [39] correspond essentiellement à la séquence $F_0, R_1, M_1 \uplus A_1, F_2, R_2, M_2 \uplus A_2, \ldots, F_k$ et contient simplement plus de répétitions.

Dans la plupart des cas que nous rencontrerons en pratique, c'est l'étape (E) qui permettra à l'algorithme de s'arrêter. La borne k permet d'assurer la terminaison dans tous les cas restants.

Pour démontrer le bien fondé de l'arrêt à l'étape (E), nous considérerons l'algorithme du graphe de planification sans cette étape que nous appellerons *algorithme simplifié*. Son résultat sera noté $\mathsf{PG}'_k(\Pi)$. L'algorithme du graphe de planification sera appelé *algorithme complet*.

Grâce au lemme 4.1, il est clair que ces deux algorithmes sont bien définis et s'arrêtent car i augmente à chaque passage à l'étape (F) et les algorithmes s'arrêtent quand $i = k$. Le lemme 4.3 en découle directement.

Lemme 4.3. *Soit* Π *un système de planification. Alors* $\mathsf{PG}_k(\Pi) = \mathsf{PG}'_k(\Pi)$.

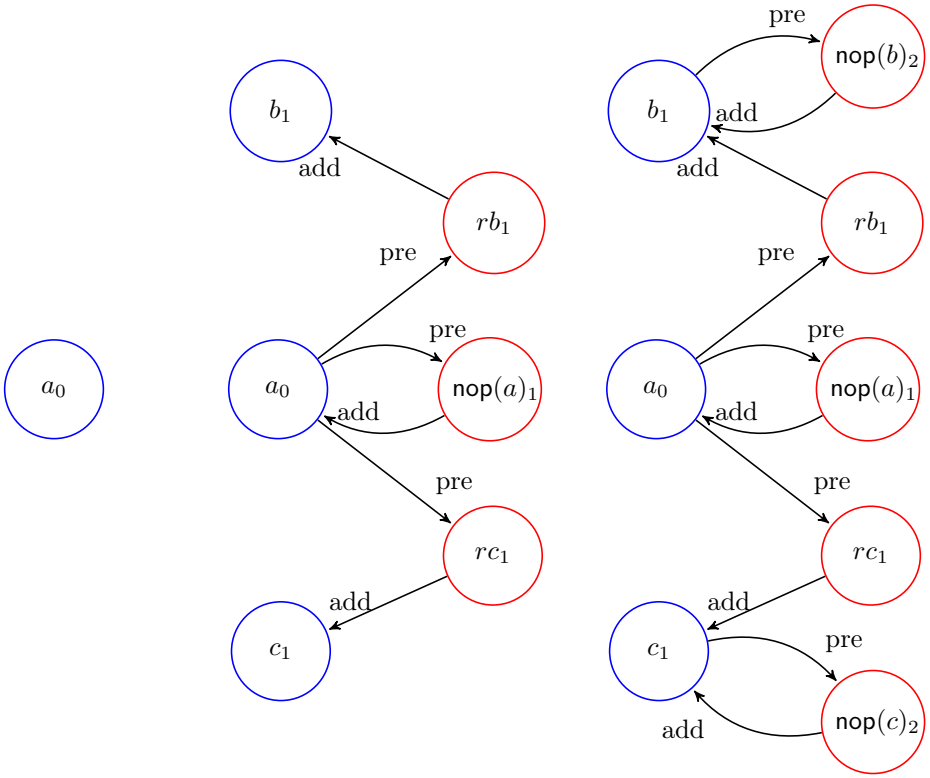


FIGURE 4.1 – Graphe de planification $\mathsf{PG}_k(\Pi_0)$ avec $k = 0$ (à gauche), $k = 1$ (au centre) et $k = 2$ (à droite). Le graphe pour $k = 2$ est un point fixe. Les faits sont entourés en bleu et les règles en rouge.

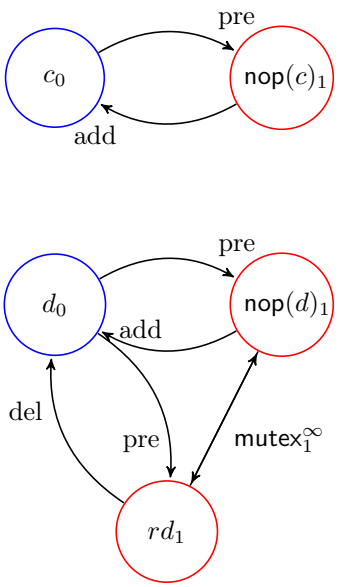


FIGURE 4.2 – Graphe de planification $\mathsf{PG}_1(\Pi'_0)$. Ce graphe est un point fixe de l'algorithme.

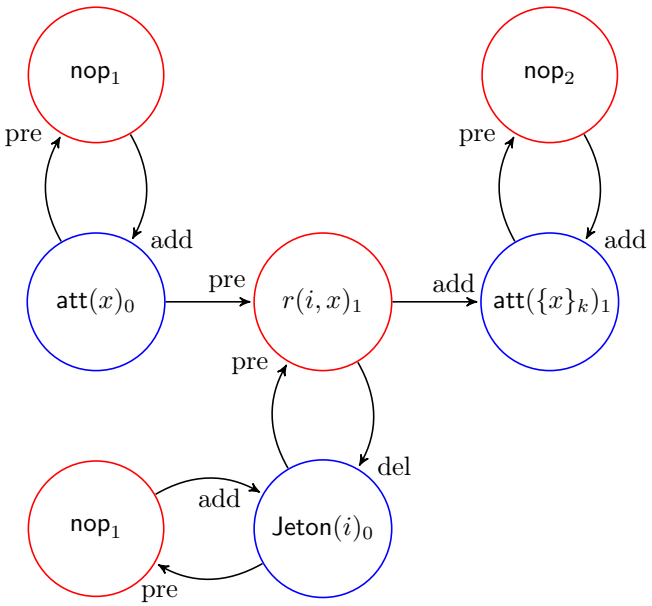


FIGURE 4.3 – La partie du graphe de planification $\mathsf{PG}_2(\Pi_1)$ reliée à chaque $r(x,i)$ avec $i \in \{1,2,3\}$ et $x \in \{a,b,c\}$ (sans les mutex). Les nop_i (pour i entier) doivent se lire comme $\mathsf{nop}(f)_i$ où f est l'unique fait auxquels ils sont reliés par des arrêtes de A.

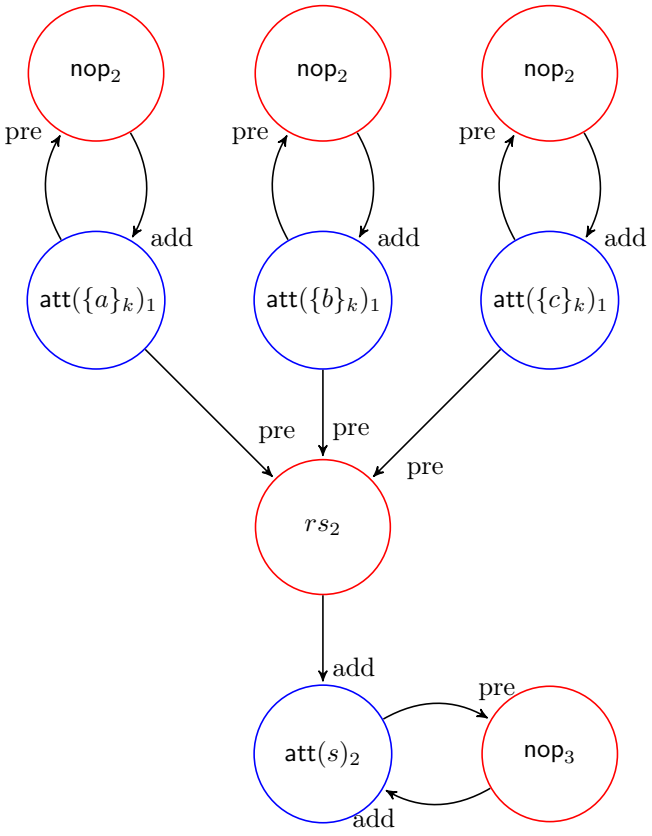


FIGURE 4.4 – Partie du graphe de planification $\mathsf{PG}_2(\Pi_1)$ qui ajoute $\mathsf{att}(s)_2$, sans les mutex. Les nop_i (pour i entier) doivent se lire comme $\mathsf{nop}(f)_i$ où f est l'unique fait auxquels ils sont reliés par des arrêtes de A.

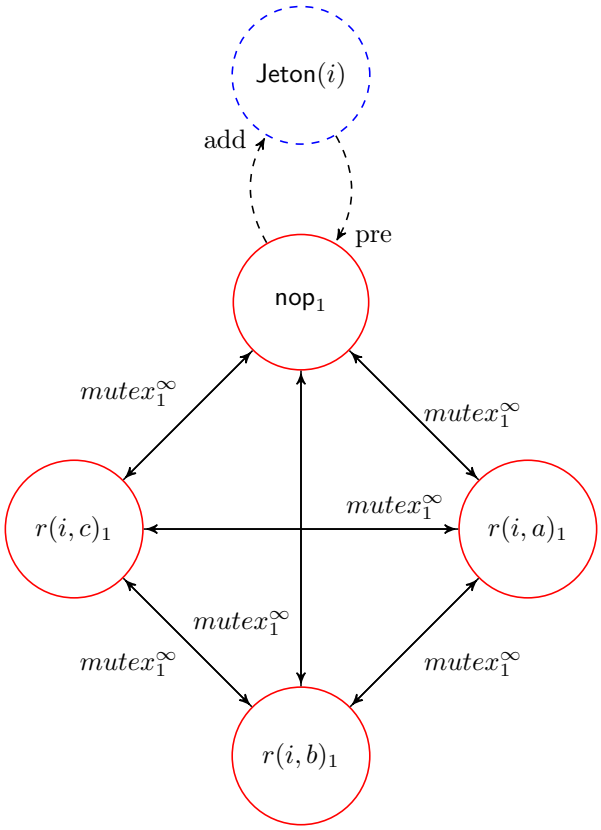


FIGURE 4.5 – Représentation d'une partie P_i des mutex du graphe de planification $\mathsf{PG}_2(\Pi_1)$ pour $i \in \{1;2;3\}$, où nop_1 représente $\mathsf{nop}(\mathsf{Jeton}(i))_1$. L'ensemble des mutex est obtenu par $P_1 \cup P_2 \cup P_3$. Dans ce graphe il n'y a pas de mutex entre les noeuds.

Démonstration. C'est une conséquence directe du lemme 4.2. □

Exemple 4.7. *Retournons au système de planification* $\Pi_0 = \langle \mathsf{F}_0, \mathcal{I}_0, \mathsf{Rule}_0 \rangle$ *de l'exemple 4.1. L'algorithme de calcul du graphe de planification atteint son point fixe à partir de* $k = 2$*. Le graphe complet final est donc* $\mathsf{PG}_2(\Pi_0) = (F, R, A, M)$ *avec* $F = \{a_0, b_1, c_1\}$, $R = \{rb_1, rc_1, \mathsf{nop}(a)_1, \mathsf{nop}(b)_2, \mathsf{nop}(c)_2\}$, $M = \emptyset$ *et*

$$\begin{aligned} A = \{ & a_0 \longrightarrow_{pre} rb_1 \ ; \ a_0 \longrightarrow_{pre} rc_1 \ ; \ rb_1 \longrightarrow_{add} b_1 \ ; \ rc_1 \longrightarrow_{add} c_1 \ ; \\ & a_0 \longrightarrow_{pre} \mathsf{nop}(a)_1 \ ; \ b_1 \longrightarrow_{pre} \mathsf{nop}(b)_2 \ ; \ c_1 \longrightarrow_{pre} \mathsf{nop}(c)_2 \ ; \\ & \mathsf{nop}(a)_1 \longrightarrow_{add} a_0 \ ; \ \mathsf{nop}(b)_2 \longrightarrow_{add} b_1 \ ; \ \mathsf{nop}(c)_2 \longrightarrow_{add} c_1 \} \end{aligned}$$

Les différentes étapes du calcul du graphe sont représentées sur la figure 4.1.

Si nous considérons maintenant $\Pi_0' = \langle \mathsf{F}_0, \{c, d\}, \mathsf{Rule}_0 \rangle$ *l'algorithme atteint son point fixe pour* $k = 1$*. Il s'agit du graphe représenté sur la figure 4.2.*

Exemple 4.8. *Le graphe de planification du système* Π_1 *(à saturation) est plus complexe, et il est donc représenté par morceaux, sous forme de trois schémas. La figure 4.5 représente les mutex pour chaque valeur de* $i \in \{1; 2; 3\}$*. Les figures 4.3 et 4.4 ne contiennent donc pas les mutex. La figure 4.3 représente seulement un motif qui est répété pour chaque valeur de* $i \in \{1, 2, 3\}$ *et* $x \in \{a, b, c\}$*. À l'exception des* nop *(dont les noms ne sont pas indiqués car il n'y a pas d'ambiguité), les noms des nœuds sont identiques sur les différents schémas si et seulement si ils représentent le même nœud. Pour obtenir le graphe complet, il faut répéter trois fois le motif de la figure 4.5, neuf fois celui de la figure 4.3 et une seule fois la figure 4.4.*

Les mutex du graphe ne portent que sur les règles. En particulier, $\mathsf{att}(\{a\}_k)_1, \mathsf{att}(\{b\}_k)_2, \mathsf{att}(\{c\}_k)_2$ *ne sont pas en mutex, ce qui permet à l'attaquant de déduire* s *dans la figure 4.4. Le motif 4.3 représente l'utilisation du jeton* $\mathsf{Jeton}(i)$ *pour chiffrer* x *et obtenir* $\{x\}_k$*.*

4.3.3 Correction

Nous démontrerons dans la section 4.3.4 que le graphe de planification est une surapproximation, c'est-à-dire qu'il contient des faits qui ne sont pas réellement accessibles par un chemin de longueur k. Pour le moment, nous voulons montrer que, si un fait n'apparaît pas dans le graphe, alors il n'est dans aucun état résultant d'un chemin de longueur k. Ainsi, étant donné un problème de planification borné $\langle \Pi, \Theta \rangle$ et le graphe $(F, R, A, M) = \mathsf{PG}_k(\Pi)$, si $\Theta \not\subseteq F$, alors le problème de planification n'a pas de solution de longueur k. Dans le chapitre suivant, lorsque nous traduirons l'équivalence de protocoles en problème de planification, nous pourrons en déduire directement que les protocoles sont en équivalence.

Proposition 4.1. *Soient* $\Pi = \langle \mathsf{F}, \mathcal{I}, \mathsf{Rule} \rangle$ *un système de planification,* k *un entier et* $(F, R, A, M) = \mathsf{PG}_k(\Pi)$*. Soit* $c = c_1 \dots c_j$ *un chemin dans* Π *de longueur* $j \leqslant k$*, applicable sur* $\mathcal{I}$*. Soit* S *tel que* $\mathcal{I} \xrightarrow{c} S$*. Alors* $S \subseteq F$ *et* $c_i \subseteq R$ *pour chaque* $1 \leqslant i \leqslant j$*.*

Démonstration. Pour simplifier la preuve, nous démontrons le résultat sur l'algorithme simplifié. Le lemme 4.3 nous permet ensuite d'en déduire le résultat pour l'algorithme complet.

Nous avons démontré qu'au cours de l'algorithme, F,R et A croissent (voir lemme 4.2). Au contraire, après l'apparition des faits et la définition initiale des mutex à l'étape 8, 9 et 10, les mutex peuvent seulement être arrêtés (à l'étape 11).

Nous allons montrer par récurrence sur la longueur $\ell \leqslant k$ de c qu'au bout de ℓ itérations, $S \subseteq F$ et que pour tout $f, f' \in S$, f et f' ne sont pas en mutex. De plus on arrive à l'étape (F) avec $i = \ell$, sauf si $\ell = 0$ auquel cas on arrive à l'étape (C).

Si $\ell = 0$, le résultat est évident. Sinon, supposons qu'on ait ℓ tel que le résultat soit vrai. Si $\ell \geqslant k$, nous avons démontré le résultat. Sinon, $\ell < k$.

On arrive à l'étape (F) avec $i = \ell$. Comme $\ell < k$, on ne s'arrête pas à cette étape.

Soit c un chemin de longueur $\ell + 1 : c = c_0.c_1$ avec c_0 un chemin de longueur ℓ. Il existe S_0 et S tels que $\mathcal{I} \xrightarrow{c_0} S_0 \xrightarrow{c_1} S$. L'hypothèse de récurrence s'applique et on arrive à l'étape (F) avec $i = \ell$, et $S_0 \subseteq F$ et aucune paire de faits de S_0 n'est en mutex à partir de cette étape.

Les règles $r \in c_1$ sont applicables à partir de S_0, donc $\mathsf{Pre}(r) \subseteq S_0 \subseteq F$. Ainsi pour chaque $r \in c_1$, $r \in O(L_f)$ à l'étape 3 du calcul de L^i_Π. Mais comme les faits de S_0 ne sont pas en mutex, soit $r \in R$, soit r n'est pas éliminée à l'étape 4 car $\mathsf{Pre}(r) \subseteq S_0$. Donc $r \in \mathsf{new}_r \cup R$. Comme $S \subseteq S_0 \cup \mathsf{Add}(c_1)$, d'après l'étape 5 du calcul de L^i_Π puis l'étape 6 du calcul de L^i_Π, $S \subseteq F$.

Il reste à montrer que deux faits quelconques de S ne sont pas en mutex. Soit $f \in S$. Si f appartient à S_0, alors on note $r_f = \mathsf{nop}(f)$. Sinon, il y a au moins une règle r de c_1 telle que $f \in \mathsf{Add}(r)$. Dans ce cas, on note $r_f = r$ (resp. $r_{f'} = r$).

Appelons $c'_1 = \{r_f \mid f \in S\}$ et $\mathsf{Pre}(c'_1) = \cup_{f \in S} \mathsf{Pre}(r_f)$. Hormis les règles de c_1, l'ensemble c'_1 ne contient que des nop donc $\mathsf{Add}(c'_1 - c_1) = \mathsf{Pre}(c'_1 - c_1)$.

Si c_1 est un singleton, il ne peut être en mutex avec lui-même ; il ne peut pas être non plus en mutex avec les règles de c'_1 car $\mathsf{Del}(c_1) \cap S = \emptyset$ et $\mathsf{Pre}(c'_1) \subseteq S$ (ce qui exclut le cas de l'étape 8 du calcul de L^i_Π, car $\mathsf{Add}(c'_1 \setminus c_1) = \mathsf{Pre}(c'_1 \setminus c_1) \subseteq S$) et $\mathsf{Pre}(c'_1) \subseteq S_0$, qui ne contient pas de paire de faits en mutex (ce qui exclut le cas de l'étape 9 du calcul de L^i_Π, ou au moins implique que la mutex ait été levée à l'étape 11 du calcul de L^i_Π). Les règles de $c'_1 - c_1$ ne peuvent être déclarées en mutex entre elles ni à l'étape 8 du calcul de L^i_Π (car elles n'effacent rien : pour tout $r \in c'_1 \setminus c_1, \mathsf{Del}(r) = \emptyset$), ni à l'étape 9 du calcul de L^i_Π car leurs préconditions sont incluses dans S_0 où il n'y a pas de mutex (ou, dans ce second cas, elles ont été levées à l'étape 11 du calcul de L^i_Π).

Sinon c'_1 est un ensemble de règles r avec $\mathsf{Del}(r) = \emptyset$, et dans ce cas ces règles ne peuvent être déclarées en mutex à l'étape 8 du calcul de L^i_Π. Elles ne peuvent pas non plus être déclarées en mutex à l'étape 9 du calcul de L^i_Π car leurs préconditions sont toutes incluses dans S_0 où il n'y a pas de mutex (ou, dans ce second cas, elles ont été levées à l'étape 11 du calcul de L^i_Π). Ainsi les règles de c'_1 ne sont pas en mutex entre elles.

Donc soient $f, f' \in S$. Il existe $r_f, r_{f'} \in c'_1$ tel que $f \in \mathsf{Add}(r_f)$ et $f' \in \mathsf{Add}(r_{f'})$. r_f et $r_{f'}$ ne sont pas en mutex, donc f et f' ne peuvent pas être déclarés en mutex à l'étape 10 du calcul de L^i_Π.

On arrive finalement à l'étape (F), ce qui permet de conclure la récurrence. □

Il paraît important d'insister sur le fait que la proposition 4.1 tisse un lien entre le nombre d'itérations k du graphe et la taille des chemins : tout chemin de longueur inférieure à k ne contient que des règles du graphe $\mathsf{PG}_k(\Pi)$ et les faits accessibles par ce chemin sont aussi inclus dans ce graphe. C'est ce lien qui nous permettra de démontrer l'arrêt de SAT-Equiv dans le chapitre 5.

4.3.4 Surapproximation

Le graphe de planification déduit des faits potentiellement accessibles, mais certains faits ne sont pas réellement accessibles, car les mutex ne tiennent compte que des conflits entre deux nœuds, et non de tous les conflits possibles. En particulier, si l'on dispose de trois faits f_1, f_2, f_3 qui ne sont pas accessibles tous ensemble, mais dont toutes les paires (f_1, f_2), (f_2, f_3) et (f_1, f_3) sont accessibles, les mutex ne peuvent pas exclure que l'on atteigne l'ensemble $\{f_1, f_2, f_3\}$. Si la règle $f_1, f_2, f_3 \to f_4$ existe, elle sera déclenchée dans l'algorithme du graphe de planification, alors qu'elle n'apparaît dans aucun chemin à partir de l'état initial. Ainsi le graphe de planification contiendra le fait inaccessible f_4. L'exemple 4.9 présente un cas où le graphe réalise une surapproximation stricte de l'ensemble des états accessibles, en même temps qu'il démontre que le cas se produit réellement. Il a été adapté d'un exemple existant (voir [39]).

Exemple 4.9. *Considérons le système de planification* $\Pi'_1 = \langle \mathsf{F}_1, \mathcal{I}_1 \setminus \{\mathsf{Jeton}(3)\}, \mathsf{Rule}_1 \rangle$ *présenté à l'exemple 4.6. Nous avions énoncé que le problème* $\langle \Pi'_1, \{\mathsf{att}(s)\} \rangle$ *n'admettait pas de solution. Pourtant, soit* $(F, R, A) = \mathsf{PG}_2(\Pi'_1)$. *On a* $\mathsf{att}(s) \in F$, *car les trois faits* $\mathsf{att}(\{a\}_k), \mathsf{att}(\{b\}_k), \mathsf{att}(\{c\}_k)$ *ne sont pas en mutex. En effet, l'attaquant peut obtenir n'importe quelle paire de ces faits grâce à ses deux jetons* $\mathsf{Jeton}(1)$ *et* $\mathsf{Jeton}(2)$, *mais il ne peut pas avoir les trois en même temps.*

En d'autres termes, le graphe de $\mathsf{PG}_2(\Pi'_1)$ *est identique à celui de* $\mathsf{PG}_2(\Pi_1)$ *pourvu qu'on supprime tous les motifs avec* $i = 3$.

Puisque le graphe de planification est une surapproximation, même quand l'ensemble d'objectifs Θ est inclus dans les faits du graphe, et même quand ces objectifs ne sont pas en mutex entre eux, il est nécessaire de vérifier qu'ils sont réellement accessibles. En pratique, nous allons utiliser le codage SAT décrit dans la section suivante.

4.4 Codage SAT

Étant donné un graphe de planification et un chemin, il est facile de vérifier dans le graphe que le chemin est un vrai chemin, et d'en calculer l'état résultant. Cependant, il faudrait alors énumérer tous les chemins compatibles avec le graphe de planification et les tester un à un. Une autre approche, proposée par Kautz et Selman (voir [81]) et suivie par Armando et Compagna pour SATMC (voir [21]), consiste à coder l'accessibilité de Θ sous forme de formule SAT, pour la fournir ensuite à un solveur SAT. C'est cette approche que nous retenons, et que nous exposons ici.

4.4.1 Problèmes SAT

On considère un ensemble de *variables propositionnelles* $\mathcal{V}$. On appelle *littéral* une formule de la forme x (littéral positif) ou $\neg x$ (littéral négatif) où $x \in \mathcal{V}$. Si ℓ est un littéral, il existe x une variable tel que $\ell = x$ ou $\ell = \neg x$. On notera $\neg\ell$ le littéral qui vaut $\neg x$ si $\ell = x$ et x si $\ell = \neg x$.

Nous appellerons *clause* une formule $\ell_1 \vee \ell_2 \vee \ldots \ell_n$ où $\ell_1, \ldots, \ell_n$ sont des littéraux et *formule SAT* un ensemble de clauses $c_1, \ldots, c_n$ noté $c_1 \wedge \cdots \wedge c_n$. Nous appelerons $B(\mathcal{V})$ l'ensemble des formules SAT ayant leurs variables dans $\mathcal{V}$.

Une *valuation* est une fonction $\mathcal{V} \to \{0, 1\}$. On peut étendre toute valuation sur $\mathcal{V}$ en une fonction $B(\mathcal{V}) \to \{0, 1\}$ avec les règles

$$\begin{aligned} v(\neg x) &= 1 - v(x) \\ v(\ell_0 \vee \ldots \ell_n) &= 1 \text{ s'il existe un } i \text{ tel que } v(\ell_i) = 1 \text{ et 0 sinon.} \\ v(c_0 \wedge \ldots c_n) &= 1 \text{ si pour tout } i,\ v(c_i) = 1 \text{ et 0 sinon.} \end{aligned}$$

Soit ϕ une formule de $B(\mathcal{V})$. Le *problème de satisfiabilité* de ϕ est le problème de savoir s'il existe une valuation v sur $\mathcal{V}$ telle que $v(\phi) = 1$ (on dit que v est un *modèle* de ϕ). On parle aussi de *problème SAT* associé à ϕ. Le théorème de Cook-Levin démontre que le problème SAT est NP-complet [24]. Il est également connu que le problème dit 3-SAT, où chaque clause est de taille au plus 3, est lui-même NP-complet. Au contraire, 2-SAT appartient à la classe des problèmes polynomiaux.

Récemment, des efforts ont été entrepris pour résoudre efficacement les problèmes SAT. La NP-complétude signifie qu'il existe des instances difficiles, mais dans la plupart des cas, le problème peut être résolu très rapidement grâce à l'algorithme CDCL (*Conflict-Driven Clause Learning*, voir par exemple [85] et [32]), ce qui est démontré par le développement d'outils très performants, tels que minisat (voir [71]) pour une version minimale, ou Glucose et Lingeling (pour une description, une comparaison, et une liste plus complète de SAT-solveurs récents, voir [28]), qui permettent aujourd'hui de traiter des formules de l'ordre de millions de clause et de variables.

4.4.2 Formule SAT associée à un graphe de planification

Nous allons déduire la formule SAT du graphe de planification. Les informations que nous avons gardées et qui ont pu sembler inutiles jusque là, comme la date d'apparition des mutex et des faits, nous aideront dans cette tâche.

Dans la suite, $\langle \Pi, \Theta \rangle$, est un problème de planification borné, k est un entier et $(F, R, A) = \mathsf{PG}_k(\Pi)$.

Variables. À chaque fait f (resp. règle r), on associe une variable f^j (resp. r^j) pour chaque $i \leqslant j \leqslant k$ où $f_i = \mathsf{node}(f)$ (resp. $r_i = \mathsf{node}(r)$). Pour synthétiser, à chaque fait $f \in F$ qui est accessible depuis l'état initial I en $j \leqslant k$ étapes correspond un nœud f_j et des variables SAT $f^j, \ldots, f^k$.

Ainsi, on passe facilement d'un chemin à une valuation ou d'une valuation à un chemin : au chemin $\mathcal{I} \xrightarrow{r} S$ correspond intuitivement la valuation v avec $v(f^0) = 1$ pour chaque $f \in \mathcal{I}$, $v(f^1) = 1$ pour chaque $f \in S$, $v(r^1) = 1$ et $v(x) = 0$ pour toute autre variable $x \in \mathcal{V}$.

Plus précisément, pour chaque $0 \leqslant i \leqslant k$ on notera $\mathcal{V}_i^{\mathsf{fact}} = \{f^i \mid \exists k, f_k \in F_i\}$, $\mathcal{V}_i^{\mathsf{rule}} = \{r^i \mid \exists k, r_k \in R_i\}$, $\mathcal{V}_i = \mathcal{V}_i^{\mathsf{fact}} \uplus \mathcal{V}_i^{\mathsf{rule}}$ et $\mathcal{V} = \cup_i \mathcal{V}_i$.

Donnons maintenant l'ensemble de clauses qui constitue notre formule SAT.

Faits initiaux. Tous les faits initiaux doivent être vrais, ce qui signifie que la formule suivante doit être satisfiable :

$$\Xi_{\mathcal{I}} = \bigwedge_{f \in \mathsf{node}^{-1}(F_0)} f^0$$

De cette manière, nous obtenons autant de clauses que de faits initiaux.

Axiomes de règles. Chaque $r_i \in R$ ne peut être active que si ses préconditions sont actives à l'étape précédente, et dans ce cas ses conséquences doivent être activées aussi.

On obtient que les formules suivantes doivent être satisfaites :

$$\begin{array}{rcl}
\Xi_i^{pre}(F,R,A,M) = & \bigwedge\{ & \neg r^i \ \vee \ f^{i-1} \ \mid \ \mathsf{node}(f) \longrightarrow_{pre} \mathsf{node}(r) \in A_i\} \\
\Xi_i^{add}(F,R,A,M) = & \bigwedge\{ & \neg r^i \ \vee \ f^{i} \ \mid \ \mathsf{node}(r) \longrightarrow_{add} \mathsf{node}(f) \in A_i\} \\
\Xi_i^{del}(F,R,A,M) = & \bigwedge\{ & \neg r^i \ \vee \ \neg f^{i} \ \mid \ \mathsf{node}(r) \longrightarrow_{del} \mathsf{node}(f) \in A_i\}
\end{array}$$

On pose $\Xi_i^{\mathsf{rule}}(F,R,A,M) = \Xi_i^{pre}(F,R,A,M) \wedge \Xi_i^{add}(F,R,A,M) \wedge \Xi_i^{del}(F,R,A,M)$.

Axiomes de changement. Les formules qui suivent expriment la nécessité d'appliquer une règle pour changer l'ensemble de faits actifs. En d'autre termes, un fait qui n'était pas actif ne peut le devenir que si une règle le produit, et un fait qui était actif ne peut disparaître que si une règle le supprime. Ce qui donne les formules suivantes pour chaque i :

$$\Xi_i^{+}(F,R,A,M) = \bigwedge_{f \in \mathsf{node}^{-1}(F_i)} \{\neg f^i \vee f^{i-1} \vee \bigvee\{r^i \mid \mathsf{node}(r) \longrightarrow_{add} \mathsf{node}(f) \in A_i\}\}$$

$$\Xi_i^{-}(F,R,A,M) = \bigwedge_{f \in \mathsf{node}^{-1}(F_i)} \{f^i \vee \neg f^{i-1} \vee \bigvee\{r^i \mid \mathsf{node}(r) \longrightarrow_{del} \mathsf{node}(f) \in A_i\}\}$$

On ajoute également la formule suivante, pour la première fois où le fait f apparaît dans le graphe (c'est-à-dire $\mathsf{node}(f) \in F_i \setminus F_{i-1}$ avec $F_{-1} = \emptyset$) :

$$\Xi_i^{\star}(F,R,A,M) = \bigwedge_{f_i \in F_i} \{\neg f^i \vee \bigvee\{r^i \mid r_i \longrightarrow_{add} f_i \in A_i\}\}$$

On pose $\Xi_i^{\mathsf{fact}}(F,R,A,M) = \Xi_i^{+}(F,R,A,M) \uplus \Xi_i^{-}(F,R,A,M) \uplus \Xi_i^{\star}(F,R,A,M)$.

Axiomes d'exclusion mutuelle. Enfin, pour chaque paire de nœuds en exclusion mutuelle, nous indiquons que les deux ne peuvent pas être actifs simultanément.

Ce qui donne :

$$\Xi_i^{mutex}(F,R,A,M) = \bigwedge\{\neg n^i \vee \neg m^i \mid \mathsf{node}(n) \overset{mutex}{\longleftrightarrow}{}_i^{\star} \mathsf{node}(m) \in M_i \text{ avec } \star \in \mathbb{N} \cup \{+;\infty\}\}$$

De cette manière, nous obtenons le morceau correspondant à l'étape i de la formule :

$$\Xi_i(F,R,A,M) = \Xi_i^{\mathsf{rule}}(F,R,A,M) \wedge \Xi_i^{\mathsf{fact}}(F,R,A,M) \wedge \Xi_i^{mutex}(F,R,A,M)$$

Solution. Pour garantir que l'état final contienne nos objectifs Θ, nous écrirons la formule suivante, à condition que $\Theta \subseteq F$, afin d'indiquer que les différents faits de Θ sont atteints à l'étape k :

$$\Xi_\Theta = \bigwedge \{f^k \mid f \in \Theta\}$$

Il n'est pas nécessaire d'énoncer cette formule pour tout $j \leqslant k$, puisque les règles nop permettent de ne rien faire jusqu'à la fin si le problème est résolu plus tôt.

La formule complète est

$$\Xi^k(\Pi, \Theta) = \Xi_\mathcal{I} \wedge \Xi_\Theta \wedge \Big(\bigwedge_{1 \leqslant i \leqslant k} \Xi_i(\mathsf{PG}_k(\Pi)) \Big)$$

4.5 Correction et complétude

Maintenant, nous devons montrer que ce codage est correct et complet par rapport à notre problème de départ, c'est-à-dire que les solutions du problème SAT correspondent à des plans du problème de planification. Il s'agit essentiellement de la même preuve que celle qui est présentée par Armando et Compagna dans [21], et qui est la première preuve de correction pour ce codage.

Cependant, l'un des avantages du graphe de planification est qu'il permet de mettre en parallèle de nombreuses règles, et en particulier d'aller bien au-delà du cas très simple des règles r avec $\mathsf{Del}(r) = \emptyset$. L'inconvénient est qu'une solution de la formule SAT peut donc correspondre à un chemin de longueur plus grande que k, et nous nous contenterons donc de démontrer que l'existence d'une solution du problème SAT correspond toujours à une solution du problème de planification.

Commençons par démontrer le résultat pour une seule étape. Pour l'énoncer clairement, étant donné un système de planification borné Π, un entier k et $(F, R, A, M) = \mathsf{PG}_k(\Pi)$, pour chaque ensemble $S \subseteq F_i$ on définit $\mathsf{vars}_i(S) = \{n^i \mid \mathsf{node}(n) \in S\}$, et, réciproquement, pour chaque valuation v on définit $\mathsf{facts}_i(v) = \{\mathsf{node}(f) \mid v(f^i) = 1\}$ et $\mathsf{rules}_i(v) = \{\mathsf{node}(r) \mid v(r^i) = 1\}$. Par extension, on dira qu'on a un chemin $S_i \xrightarrow{r} S_{i+1}$ avec $S_i, S_{i+1} \subseteq F$ si on a un chemin entre les états correspondants dans le système Π.

Lemme 4.4. *Soient Π un système de planification et i, k deux entiers tels que $1 \leqslant i \leqslant k$. Soient $\Xi = \Xi_i(\mathsf{PG}_k(\Pi))$ et $S_{i-1} \subseteq F_{i-1}$ un ensemble de nœuds. S'il existe un modèle v de la formule $\Xi \wedge (\bigwedge_{x \in vars SAT_{i-1}(S_{i-1})} x)$ tel que $\mathsf{facts}_{i-1}(v) = S_{i-1}$ alors il existe un pas $S_{i-1} \xrightarrow{p} \mathsf{facts}_i(v)$ où $\mathsf{rules}_i(v) = p$.*

Démonstration. Posons $(F, R, A, M) = \mathsf{PG}(\Pi)$. Dans cette preuve, nous dirons qu'un ensemble E de nœuds est *sans mutex* dans M_i si $n \xleftrightarrow{mutex}{}^{\star}_{p} m \notin M_i$ pour tout $n, m \in E$ et tout $\star \in \{+, \infty\} \cup \mathbb{N}$.

Considérons l'ensemble de nœuds $\mathsf{rules}_i(v)$. Comme il ne contient que des nœuds correspondants à des règles, appelons $E_i \subseteq R_i$ cet ensemble de règles. D'après la formule $\Xi_i^{mutex}(\mathsf{PG}_k(\Pi))$, E_i est sans mutex dans A_i.

D'après les étapes 8 et 11 du calcul de L_Π, on en déduit que pour $r, r' \in E_i$, on avait $\mathsf{Add}(r) \cap \mathsf{Del}(r') = \emptyset$, $\mathsf{Pre}(r) \cap \mathsf{Del}(r') = \emptyset$, et réciproquement.

Par ailleurs, par $\mathsf{facts}_{i-1}(v) = S_{i-1}$, seules les variables de $\mathsf{vars}(S_{i-1})$ s'évaluent à 1 par v à l'itération $i - 1$. Donc par $\Xi_i^{pre}(F, R, A, M)$ et l'étape 7 du calcul de L_Π^i, pour chaque $r \in E_i$, $\mathsf{Pre}(r) \subseteq S_{i-1}$.

Comme les règles de E_i ne suppriment pas mutuellement leurs préconditions, et comme toutes les préconditions sont vraies dans S_{i-1}, on peut montrer facilement par récurrence qu'elles s'appliquent dans n'importe quel ordre à partir de S_{i-1} et comme elles n'effacent pas mutuellement

leurs ajouts, toutes les séquences parviennent au même résultat final

$$S_i = (S_{i-1} \setminus (\cup_{r \in E_i} \mathsf{Del}(r))) \cup (\cup_{r \in E_i} \mathsf{Add}(r))$$

La formule $\Xi_i^{add}(F, R, A, M)$ garantit que $\cup_{r \in E_i} \mathsf{Add}(r) \subseteq \mathsf{facts}_i(v)$. La formule $\Xi_i^{+}(F, R, A, M)$ impose que $\mathsf{facts}_i(v) \subseteq S_{i-1} \cup (\cup_{r \in E_i} \mathsf{Add}(r))$ et $\Xi_i^{-}(F, R, A, M)$ donne $S_{i-1} \setminus (\cup_{r \in R_i} r) \subseteq \mathsf{facts}_i(v)$. Enfin, la formule $\Xi_i^{del}(F, R, A, M)$ fournit $\cup_{r \in E_i} \mathsf{Del}(r) \cap \mathsf{facts}_i(v) = \emptyset$, ce qui nous permet de conclure que

$$\mathsf{facts}_i(v) = (S_{i-1} \setminus (\cup_{r \in E_i} \mathsf{Del}(r))) \cup (\cup_{r \in E_i} \mathsf{Add}(r)) = S_i$$

Ce qui conclut la preuve du lemme. □

Nous pouvons maintenant démontrer le théorème principal de ce chapitre. Il nous autorise d'une part à ne pas écrire la formule SAT lorsque notre objectif Θ n'apparaît pas dans le graphe de planification, et d'autre part à traduire le problème de planification en formule SAT quand c'est nécessaire.

Théorème 4.1. *Soient $\langle \Pi, \Theta \rangle$ un problème de planification et k un entier. Soit $(F, R, A, M) = \mathsf{PG}_k(\Pi)$.*

- *Si $\Theta \not\subseteq F$, alors $\langle \Pi, \Theta \rangle$ n'admet pas de solution de longueur ℓ avec $\ell \leqslant k$.*
- *Si $\Theta \subseteq F$, et s'il existe un modèle de $\Xi = \Xi^k(\Pi, \Theta)$, alors $\langle \Pi, \Theta \rangle$ admet un plan.*
- *S'il existe un plan de longueur au plus k de Θ dans Π, alors il existe un modèle de Ξ.*

Démonstration. Il est clair que si $\Theta \not\subseteq F$, $\langle \Pi, \Theta \rangle$ n'admet pas de solution d'après la proposition 4.1. Supposons donc que $\Theta \subseteq F$. Dans ce cas, la formule Ξ est bien définie.

Montrons d'abord que s'il existe un modèle de Ξ, alors $\langle \Pi, \Theta \rangle$ admet un plan. Comme F_0 ne contient que $\mathcal{I}$, $\Xi_{\mathcal{I}}$ garantit que tout modèle v de Ξ vérifie $\mathsf{facts}_0(v) = F_0 = \mathcal{I}$. Ξ_Θ garantit que $\Theta \subseteq \mathsf{facts}_k(v)$. Le lemme 4.4 appliqué inductivement (k fois) nous permet de conclure directement qu'il existe un plan de Θ dans Π.

Considérons maintenant le plan $\mathcal{I} = S_0 \xrightarrow{c_1} S_1 \dots \xrightarrow{c_k} S_k$ avec $\Theta \subseteq S_k$ et démontrons qu'il existe un modèle v de Ξ. D'après la proposition 4.1, $\cup_{0 \leqslant i \leqslant k} S_i \subseteq F$ et $\cup_{1 \leqslant i \leqslant k} c_i \subseteq R$.

On pose v le modèle $v(f^i) = 1$ si $\mathsf{node}(f) \in S_i$; $v(r^i) = 1$ si $\mathsf{node}(r) \in c_i$ et $v(n) = 0$ sinon. Nous allons montrer que v est une solution. Procédons par récurrence sur $j \leqslant k$, et montrons que v est une solution de la formule pour toutes les variables jusqu'à l'étape j, c'est-à-dire que v est un modèle de $\Xi_j = \Xi_{\mathcal{I}} \wedge (\bigwedge_{1 \leqslant i \leqslant j} \Xi_i(F, R, A))$.

Si $j = 0$, v est une solution de $\Xi_{\mathcal{I}}$ par définition de v. Supposons que v soit une solution de Ξ_j. Il faut montrer que v est une solution de $\Xi_{j+1}(F, R, A)$. Comme v est une solution de Ξ_j, il n'y a pas de mutex à l'étape j entre les éléments de S_j d'après la formule $\Xi_j^{mutex}(F, R, A)$.

On a $S_{j+1} = (S_j \cup \mathsf{Add}(c_{j+1})) \setminus \mathsf{Del}(c_{j+1})$. Comme $\mathsf{rules}_{j+1}(v) = c_i$ et $\mathsf{facts}_{j+1}(v) = S_{j+1}$, v satisfait la formule $\Xi_{j+1}^{add}(F, R, A) \wedge \Xi_{j+1}^{add}(F, R, A)$. $\mathsf{Pre}(c_i) \subseteq S_j$ permet de satisfaire aussi la formule $\Xi_{j+1}^{pre}(F, R, A)$. Comme $(S_j \setminus \mathsf{Del}(c_{j+1})) \subseteq S_{j+1}$, v est un modèle de la formule $\Xi_{j+1}^{-}(F, R, A)$; et comme $S_{j+1} \setminus \mathsf{Add}(c_{j+1}) \subseteq S_j$, v est aussi un modèle de $\Xi_{j+1}^{+}(F, R, A)$ et $\Xi_{j+1}^{\star}(F, R, A)$.

Il reste à démontrer qu'il n'y a pas de mutex entre les nœuds représentant c_{j+1} ni entre ceux représentants S_{j+1}. Comme v était une solution de Ξ_j, il n'y avait pas de mutex entre les faits de S_j. Donc il n'y en a pas eu de propagées aux éléments de c_{j+1} par l'étape 9 du calcul de L_Π (ou alors elles ont déjà été interrompues à l'étape 9 du calcul de L_Π) . Comme c_{j+1} est soit un singleton, soit un ensemble de règles avec $\mathsf{Del}(c_{j+1}) = \emptyset$, aucune mutex n'a pas être créé entre les règles de c_{j+1} à l'étape 8 du calcul de L_Π. De plus, notons

$$c'_{j+1} = c_{j+1} \cup (\cup_{f \in S_{j+1} \cap S_j} \mathsf{nop}(f))$$

Comme $\mathsf{Pre}(\mathsf{nop}(f)) \cap \mathsf{Del}(c_{j+1}) = \emptyset$ pour $f \in S_{j+1} \cap S_j$, il n'y a pas de mutex entre les règles de c'_{j+1}. Il n'y a pas non plus de mutex dans les faits de S_{j+1} car elles n'ont pas pu être propagées à l'étape 10 du calcul de L_Π.

Ainsi, v satisfait Ξ^{mutex}_{j+1}. Ainsi, v est un modèle de $\Xi_{\mathcal{I}} \wedge (\bigwedge_{1 \leqslant i \leqslant j+1} \Xi_i(F, R, A))$, ce qui clôt la récurrence.

Comme $\mathsf{facts}_k(v) = S_k$ et $\Theta \subseteq S_k$, v est un modèle de Ξ_Θ. Donc v est un modèle de Ξ et le théorème est démontré. □

4.6 Conclusion

Dans ce chapitre, j'ai présenté les systèmes de planification et une adaptation de l'algorithme du graphe de planification, afin de réduire la taille qu'il occupe en mémoire. Cet algorithme permet de résoudre ces problèmes en s'appuyant sur la résolution de formules SAT. Le théorème 4.1 démontre que l'algorithme calcule tous les états accessibles et que le codage SAT permet de trouver toutes les attaques.

Nous utilisons le graphe de planification de la même manière qu'Armando et Compagna, c'est-à-dire à la fois comme un filtre (si le graphe ne contient pas les objectifs, ces objectifs ne sont pas accessibles) et comme un guide pour le codage SAT. Cependant, chez Armando et Compagna, le typage du protocole étudié suffit à garantir la finitude de l'espace de recherche et donc du problème de planification associé. La notion de problème borné leur est inutile, et à notre connaissance elle est nouvelle.

Une piste d'amélioration envisageable consisterait à adapter des méthodes de planification plus récentes proposées par Rintanen, Heljanko et Niemelä (voir [92]). En particulier, l'efficacité peut être améliorée en renonçant à construire une formule SAT minimale à travers plus d'appels au solveur SAT. Toutefois la traduction, présentée dans le chapitre suivant, de l'équivalence de protocoles vers les problèmes de planification utilise un système de planification théoriquement infini, et l'algorithme du graphe de planification sert également à en extraire un sous-domaine fini sur lequel la traduction est possible. Il n'est pas clair que les algorithmes de Rintanen, Heljanko et Niemelä puissent remplir ce rôle.

Dans le chapitre suivant, nous ferons le lien avec l'équivalence, que nous coderons comme un problème d'accessibilité dans le graphe de planification, grâce à l'hypothèse de déterminisme. En particulier, nous devrons expliciter l'oracle pour démontrer que nous arrivons à un problème borné. Dans le dernier chapitre, nous démontrerons l'efficacité de cette approche.

5 Équivalence de protocoles et planification

Les chapitres 2 et 3 ont permis de minimiser l'ensemble de traces à considérer. Les problèmes de planification ont été présentés au chapitre 4. Le présent chapitre représente donc la clef de voute de la démarche : il s'agit de traduire le problème de l'équivalence de trace sous forme de problème de planification. Ces travaux s'inspirent notamment de l'approche théorique de SATMC [21], qui présente une procédure analogue pour les propriétés d'accessibilité. Cependant, les travaux exposés ici atteignent un niveau de complexité plus élevé.

En effet, le cas de l'équivalence est bien plus délicat que celui des propriétés de trace. En particulier, nous devons considérer deux protocoles, mais les contraintes de typage ne portent que sur l'un d'entre eux. Il s'agit donc de contrôler indirectement l'autre protocole. De plus, les règles doivent modéliser le comportement des protocoles, mais doivent également permettre de rendre compte des échecs d'exécution qui peuvent procurer un témoin. Autrement dit, la difficulté consiste à formuler l'équivalence de protocoles en termes d'accessibilité dans un système de planification.

La section 5.1 expose les hypothèses qui seront nécessaires dans ce chapitre. En particulier, la signature est fixée pour permettre des optimisations précises. La section 5.2 présente alors la traduction pour l'attaquant passif, en commençant par démontrer une version optimisée de l'équivalence statique. Il s'agit ensuite de donner des règles de planification pour représenter le pouvoir de déduction de l'attaquant. Il devient alors possible de présenter la traduction pour l'attaquant actif dans la section 5.3. Le théorème principal de ce chapitre énonce alors que deux protocoles ne sont pas en inclusion de trace si et seulement si il existe un plan dans le système de planification correspondant. Une borne fine sur la longueur des plans est également énoncée. La section 5.4 aborde les aspects de l'algorithme du graphe de planification qui ont été laissés de côté dans le chapitre précédent. D'une part, nous définissons un oracle de planification, qui permet de démontrer que le système de planification est borné. D'autre part, nous discutons de la nécessité de la borne sur la longueur des plans pour obtenir la terminaison, en donnant des exemples pour lesquels la saturation ne suffit pas.

5.1 Précisions sur le modèle

Les résultats des chapitres 2 et 3 ont été introduits avec un certain degré de généralité. Des optimisations peuvent être obtenues au prix de quelques restrictions supplémentaires, qui excluent principalement les versions randomisées des chiffrements symétrique et asymétrique. De plus, dans le chapitre 3, nous avons montré qu'il suffisait de considérer les trois constantes $c_0^\star, c_1^\star$ et $c_+^\star$ en plus de celles qui apparaissent dans les protocoles. Dans ce chapitre, nous notons donc $\Sigma_0^\# = \Sigma_0^- \uplus \{c_0^\star; c_1^\star; c_+^\star\}$ et nous considérons souvent un ensemble fini $\Sigma_0 \subseteq \Sigma_0^\#$.

Soit $n \in \mathbb{N}$. Dans la suite, nous considérerons donc la signature fixée $\Sigma_n^{\mathsf{SatEq}} = \Sigma_c \uplus \Sigma_d$ avec $\Sigma_d = \Sigma_d' \uplus \{\mathsf{check}\}$ et

$$\Sigma_c = \{\mathsf{senc}, \mathsf{aenc}, \mathsf{hash}, \mathsf{pub}, \mathsf{sign}, \mathsf{vk}, \mathsf{ok}\} \cup \{\langle\rangle^k \mid 2 \leqslant k \leqslant n\}$$
$$\Sigma_d' = \{\mathsf{sdec}, \mathsf{adec}, \mathsf{getmsg}, \mathsf{proj}_1^2, \ldots, \mathsf{proj}_n^n\}$$

Les règles de réécriture, les sortes et les contours de ces symboles ont été définis dans la section 1.1.7.

Le symbole check est un destructeur particulier, car la règle associée est la seule règle de la forme $\ell_{\mathsf{des}} \to t_0$ avec t_0 un terme clos. Cette règle ne permet donc pas à l'attaquant de construire de nouveaux messages, et elle ne lui est utile que pour l'équivalence statique. Les autres règles, comme $\mathsf{sdec}(\mathsf{senc}(x,y),y) \to x$, servent à la fois pour l'équivalence statique et pour la déduction.

Il serait facile d'ajouter une primitive mac à cette signature (il s'agit simplement d'un symbole libre d'arité 2). Au contraire, les primitives rsenc et raenc ne permettraient pas d'effectuer les optimisations de la section 5.2.1, car ces optimisations supposent que l'attaquant peut reconstruire les chiffrements qu'il peut ouvrir.

La signature fixée permet de renforcer la notion de recette simple. Dans ce contexte, une $\Sigma_0^{\#}$-*recette destructrice* est un terme de $\mathcal{T}(\Sigma'_d, \mathcal{W} \cup \Sigma_0^{\#})$, et une $\Sigma_0^{\#}$-recette *fortement simple* est de la forme $C[R_1, \ldots, R_k]$ où C est un contexte construit sur Σ_c et chaque R_i est une recette destructrice. En particulier, le destructeur check n'apparaît dans aucune recette fortement simple. La section suivante montrera que les recettes fortement simples sont suffisantes pour déterminer si deux trames sont en équivalence statique.

La définition générale de la réécriture forcée a été donnée à la section 2.3.1. Les règles associées à notre théorie sont :

$$\mathsf{sdec}(\mathsf{senc}(x,y),z) \twoheadrightarrow x$$
$$\mathsf{adec}(\mathsf{aenc}(x,y),z) \twoheadrightarrow x$$
$$\mathsf{getmsg}(\mathsf{sign}(x,y)) \twoheadrightarrow x$$
$$\mathsf{check}(x,y) \twoheadrightarrow \mathsf{ok}$$
$$\mathsf{proj}_j^i(\langle x_1, \ldots, x_n \rangle^i) \twoheadrightarrow x_j \text{ pour } i,j \text{ tels que } 2 \leqslant j \leqslant i \leqslant n$$

Rappelons l'énoncé du lemme 2.2 du chapitre 2, qui montre que la forme normale forcée ne change pas la valeur d'un message.

Lemme 2.2. *Soient ϕ une Σ_0-trame et R une Σ_0-recette telles que $R\phi\!\downarrow$ est un Σ_0-message. Si R' vérifie $R \twoheadrightarrow R'$, alors R' est une Σ_0-recette, et $R'\phi\!\downarrow = R\phi\!\downarrow$.*

Nous avons également démontré qu'une recette en forme normale forcée est simple dès qu'elle se réduit comme un message. Nous sommes en mesure d'énoncer un résultat analogue.

Lemme 5.1. *Soit R une Σ_0-recette en forme normale pour $\twoheadrightarrow$. Si $R\phi\!\downarrow$ est un message pour une certaine $\Sigma_0^{\#}$-trame ϕ, alors R est une $\Sigma_0^{\#}$-recette fortement simple.*

Démonstration. Démontrons le résultat par récurrence structurelle sur R.

Initialisation. Traitons le cas $R \in \mathcal{W} \cup \Sigma_0$. Comme R est fortement simple, le résultat est vrai.

Hérédité. Traitons le cas $R = \mathsf{f}(R_1, \ldots, R_k)$ pour un certain $\mathsf{f} \in \Sigma_n^{\mathsf{SatEq}}$ et $R_1, \ldots, R_k$ sont des $\Sigma_0^{\#}$-recettes en forme normale pour $\twoheadrightarrow$.

— *Cas où $f \in \Sigma_c$.* Le terme $R\phi\!\downarrow = \mathsf{f}(R_1\phi\!\downarrow, \ldots, R_k\phi\!\downarrow)$ est un $\Sigma_0^{\#}$-message pour une certaine $\Sigma_0^{\#}$-trame ϕ, et donc $R_i\phi\!\downarrow$ est un $\Sigma_0^{\#}$-message pour chaque i. L'hypothèse de récurrence appliquée à chaque R_i permet de conclure.

— *Cas où $f = \mathsf{des} \in \Sigma'_d \uplus \{\mathsf{check}\}$.* Soient $\ell_{\mathsf{des}} \to r_{\mathsf{des}}$ la règle de réécriture associée à des, et $\ell'_{\mathsf{des}} \twoheadrightarrow r_{\mathsf{des}}$ sa règle de réécriture forcée associée. Le cas $\mathsf{des} = \mathsf{check}$ est impossible car R est en forme normale pour $\twoheadrightarrow$. Donc $\mathsf{f} \in \{\mathsf{proj}_i^j, \mathsf{sdec}, \mathsf{adec}, \mathsf{getmsg}\}$. Supposons que $\mathsf{f} = \mathsf{adec}$ (les autres cas se traitent de la même manière). Puisque $\mathsf{f} = \mathsf{adec}$, $R = \mathsf{f}(R_1, R_2)$. Comme $R\phi\!\downarrow$ est un $\Sigma_0^{\#}$-message, $R_1\phi\!\downarrow$ et $R_2\phi\!\downarrow$ sont des $\Sigma_0^{\#}$-messages. En appliquant l'hypothèse

de récurrence, R_1 et R_2 sont des recettes fortement simples. Comme $R_2\phi\downarrow$ est un message atomique, R_2 est une recette destructrice. De plus, supposons que $R_1 = \mathsf{g}(R'_1, \ldots, R'_n)$ pour un certain $\mathsf{g} \in \Sigma_c$. Comme $R\phi\downarrow$ est un $\Sigma_0^{\#}$-message, $\mathsf{g} = \mathsf{aenc}$, ce qui mène à une contradiction, car R est en forme normale pour $\twoheadrightarrow$. Donc, comme R_1 est fortement simple et comme sa racine n'est pas un constructeur, nécessairement R_1 est une $\Sigma_0^{\#}$-recette destructrice, donc R est fortement simple.

Comme nous avons traité tous les cas, le résultat est démontré. □

5.2 Attaquant passif

La première étape consiste à modéliser l'attaquant passif dans le langage de la planification. En limitant fortement le nombre de recettes de l'attaquant pour l'équivalence statique, nous permettons à la procédure de terminer, mais aussi d'atteindre une meilleure efficacité. Pour cette raison, nous simplifions d'abord l'inclusion statique en donnant une définition équivalente, en section 5.2.1. Ensuite, la section 5.2.2 expose les règles de planification qui représentent l'attaquant passif, c'est-à-dire à la fois les règles de déduction et les règles pour les tests d'inclusion statique. Cette section énonce également la proposition 5.1 qui affirme que la traduction de l'attaquant passif sous forme de règles est correcte et complète, et donne une borne sur la longueur du plan. Le raisonnement qui mène à cette proposition est présenté dans la section 5.2.3. Enfin, les preuves formelles sont données en section 5.2.4.

5.2.1 Équivalence statique

Dans cette section, nous transformons l'inclusion statique pour en donner une définition plus adaptée à notre modèle, puis nous montrons que cette notion coïncide avec la notion d'origine, et nous en tirons des conséquences sur la forme des recettes à retenir pour tester l'équivalence statique.

Définition 5.1. *Soient ϕ, ψ deux $\Sigma_0^{\#}$-trames telles que $dom(\phi) = dom(\psi)$. L'inclusion statique simplifiée $\phi \sqsubseteq_s^{\mathsf{simple}} \psi$ est définie par :*

1. *Pour toute $\Sigma_0^{\#}$-recette destructrice R telle que $R\phi\downarrow$ est un $\Sigma_0^{\#}$-message (respectivement un $\Sigma_0^{\#}$-message atomique), $R\psi\downarrow$ est un $\Sigma_0^{\#}$-message (respectivement un $\Sigma_0^{\#}$-message atomique).*
2. *Pour toute $\Sigma_0^{\#}$-recette fortement simple R et toute $\Sigma_0^{\#}$-recette destructrice R' telles que $R\phi\downarrow$, $R'\phi\downarrow$ sont des $\Sigma_0^{\#}$-messages et $R\phi\downarrow = R'\phi\downarrow$, l'égalité $R\psi\downarrow = R'\psi\downarrow$ est vérifiée.*
3. *Pour toutes recettes destructrices R, R', si $R\phi\downarrow = \mathsf{sign}(t, s)$ pour un certain $\Sigma_0^{\#}$-message t et un certain atome s, et $R'\phi\downarrow = \mathsf{vk}(s)$, alors $R\psi\downarrow = \mathsf{sign}(t', s')$ et $R'\psi\downarrow = \mathsf{vk}(s')$ pour un certain $\Sigma_0^{\#}$-message t' et un certain atome s'.*
4. *Pour toute $\Sigma_0^{\#}$-recette fortement simple R telle que $R\phi\downarrow = \mathsf{pub}(s)$ pour chaque $\Sigma_0^{\#}$-message atomique s, il existe un $\Sigma_0^{\#}$-message atomique s' tel que $R\psi\downarrow = \mathsf{pub}(s')$.*

De plus, les deux trames vérifient $\phi \sqsubseteq_s^{\mathsf{simple}^+} \psi$ si les tests de l'item 2 ne sont considérés que (i) si R est une $\Sigma_0^{\#}$-recette destructrice ; ou (ii) si $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\rangle^k \mid 2 \leqslant k \leqslant n\}$.

Ces inclusions coïncident avec l'inclusion originale. L'idée est d'utiliser les tests d'une inclusion pour simuler ceux de l'autre, ce qui nous donne le lemme suivant, qui sera démontré en section 5.2.4 :

Lemme 5.2. *Soient ϕ et ψ deux $\Sigma_0^{\#}$-trames avec $dom(\phi) = dom(\psi)$. On a :*

$$\phi \sqsubseteq_s \psi \Leftrightarrow \phi \sqsubseteq_s^{\mathsf{simple}} \psi \Leftrightarrow \phi \sqsubseteq_s^{\mathsf{simple}^+} \psi$$

En particulier, ce lemme nous autorise à considérer seulement un nombre fini de tests : les recettes destructrices qui se réduisent comme un message dans une trame donnée sont en nombre fini si l'on ne tient pas compte des constantes ; pour les tests d'égalité, la recette destructrice contraint fortement le chapeau constructeur de la recette correspondante.

Une manière d'utiliser au mieux le lemme 5.2 consiste à repérer un ensemble de sous-termes sur lesquels on va effectuer les tests d'égalité, et qui correspond à l'ensemble des termes qui ne peuvent pas être reconstruits à partir de leur sous-termes. Plus précisément, soit t un terme. L'ensemble $St_{\mathsf{opti}}(t)$ des sous-termes optimaux de t est défini récursivement par :

— $St_{\mathsf{opti}}(\langle t_1, \ldots, t_k \rangle^k) = St_{\mathsf{opti}}(t_1) \cup \cdots \cup St_{\mathsf{opti}}(t_n)$;
— $St_{\mathsf{opti}}(\mathsf{senc}(t_1, t_2)) = St_{\mathsf{opti}}(t_1)$;
— $St_{\mathsf{opti}}(\mathsf{aenc}(t_1, t_2)) = \{\mathsf{aenc}(t_1, t_2)\} \cup (St_{\mathsf{opti}}(t_1) \setminus t_1)$;
— $St_{\mathsf{opti}}(\mathsf{sign}(t_1, t_2)) = \{\mathsf{sign}(t_1, t_2)\} \cup St_{\mathsf{opti}}(t_1)$;
— $St_{\mathsf{opti}}(\mathsf{f}(t)) = \{\mathsf{f}(t)\}$ pour $\mathsf{f} \in \{\mathsf{hash}, \mathsf{pub}, \mathsf{vk}\}$
— $St_{\mathsf{opti}}(t) = \{t\}$ quand t est une constante ou un nom.

Grâce au lemme 5.2, il est possible de considérer seulement les recettes dont le résultat est un élément de $St_{\mathsf{opti}}(\phi) = \cup_{t \in img(\phi)} St_{\mathsf{opti}}(t)$.

Lemme 5.3. *Soient ϕ une $\Sigma_0^{\#}$-trame, R une recette fortement simple telle que $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\rangle^k \mid 2 \leqslant k \leqslant n\}$ et R' une recette destructrice telle que $\mathsf{root}(R') \neq \mathsf{adec}$. Supposons que $R\phi\!\downarrow$ et $R'\phi\!\downarrow$ soient deux $\Sigma_0^{\#}$-messages tels que $R\phi\!\downarrow = R'\phi\!\downarrow$. Ou bien C est le contexte vide, ou bien $R\phi\!\downarrow \in St_{\mathsf{opti}}(\phi) \cup \Sigma_0^{\#}$.*

Pour montrer ce résultat, il faut tout d'abord remarquer que tout terme $R\phi\!\downarrow$ est un sous-terme déductible de ϕ, et que l'ensemble des sous-termes potentiellement déductibles de ϕ peut se calculer *a priori*. Puis il faut examiner la recette R' pour en déduire que nécessairement $R'\phi\!\downarrow \in St_{\mathsf{opti}}(\phi)$. Voir la section 5.2.4 pour la preuve formelle.

5.2.2 Règles symboliques

Comme annoncé dans le chapitre précédent, la vérification de l'équivalence de protocoles va être codée sous forme de problème de planification. L'ensemble $\mathcal{F}_0$ des faits est constitué de formules qui représentent la connaissance de l'attaquant, de la forme $\mathsf{att}(u_P, u_Q)$ avec u_P et u_Q des messages ; et d'un symbole spécial bad. Intuitivement, les faits de la forme $\mathsf{att}(u_P, u_Q)$ expriment l'idée que l'attaquant a appris simultanément u_P du côté du protocole P et u_Q du côté du protocole Q. Le symbole bad signifie que l'on a atteint un état d'attaque.

Pour le moment, il s'agit d'énoncer les règles symboliques de l'attaquant passif. Ces règles permettent à l'attaquant de déduire de nouveaux messages à partir de ceux qu'il connaît. Commençons par l'ensemble $\mathsf{R}_{\mathsf{ana}}$ règles d'analyse, qui décomposent les messages existants :

$$\begin{aligned}
\mathsf{att}(\langle x_1, \ldots, x_k \rangle^k, \langle y_1, \ldots, y_k \rangle^k) &\to \mathsf{att}(x_i, y_i) \text{ avec } i \leqslant k \\
\mathsf{att}(\mathsf{senc}(x_1, x_2), \mathsf{senc}(y_1, y_2)), \mathsf{att}(y_1, y_2) &\to \mathsf{att}(x_1, y_1) \\
\mathsf{att}(\mathsf{aenc}(x_1, \mathsf{pub}(x_2)), \mathsf{aenc}(y_1, \mathsf{pub}(y_2))), \mathsf{att}(y_1, y_2) &\to \mathsf{att}(x_1, y_1) \\
\mathsf{att}(\mathsf{sign}(x_1, x_2), \mathsf{sign}(y_1, y_2)) &\to \mathsf{att}(x_1, y_1)
\end{aligned}$$

Ces règles correspondent à la capacité de l'attaquant de projeter, déchiffrer et récupérer les messages signés. Elles n'effacent rien ($\mathsf{Del} = \emptyset$) car l'attaquant n'oublie pas. Les règles de $\mathsf{R}_{\mathsf{ana}}$ sont symboliques (elles contiennent des variables). Elles permettent d'obtenir des règles instanciées à

travers la concrétisation. Étant donnée une règle $r \in \mathsf{R_{ana}}$, sa *concrétisation positive* $\mathsf{Concrete}^+(r)$ est l'ensemble des instances de r dont les termes sont des messages. Plus formellement :

$$\mathsf{Concrete}^+(r) = \{r\sigma \mid \sigma \text{ est une substitution telle que } r\sigma \text{ ne contient que des } \Sigma_0^{\#}\text{-messages.}\}$$

Lorsque la concrétisation est possible côté P mais pas côté Q, nous faisons appel à la *concrétisation négative* $\mathsf{Concrete}^-(r)$. Plus formellement, une séquence de faits $\mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k)$ *s'unifie à gauche* avec une autre séquence $\mathsf{att}(u'_1, v'_1), \ldots, \mathsf{att}(u'_k, v'_k)$ s'il existe une substitution σ telle que $u'_1\sigma = u_1, \ldots, u'_k\sigma = u_k$. L'unification à droite se définit similairement. Étant donnée une règle de l'attaquant abstraite $r = \mathsf{Pre} \to \mathsf{Add}$, $\mathsf{Concrete}^-(r)$ est l'ensemble contenant $f_1, \ldots, f_k \to \mathsf{bad}$ pour chaque séquence $f_1, \ldots, f_k \in \mathcal{F}_0$ qui s'unifie à gauche, mais pas à droite, avec Pre. L'ensemble $\mathsf{Concrete}(r)$ rassemble les concrétisations positives et négatives de r :

$$\mathsf{Concrete}(r) = \mathsf{Concrete}^+(r) \uplus \mathsf{Concrete}^-(r)$$

Illustrons la concrétisation sur notre exemple de référence.

Exemple 5.1. *Considérons les Σ_0^+-trames $\phi^\star$ et $\psi^\star$ définies à l'exemple 3.1. Ces trames sont en fait des Σ_0-trames avec $\Sigma_0 = \{a, b, c_0^\star\}$, et $\mathsf{Fact}_{\Sigma_0}(\phi, \psi)$ contient, en plus de $\mathsf{att}(c_0^\star, c_0^\star), \mathsf{att}(a, a)$ et $\mathsf{att}(b, b)$ qui correspondent aux constantes de Σ_0, les faits suivants :*

$$\begin{aligned}
f_1 &= \mathsf{att}(k_{as}, k)\\
f_2 &= \mathsf{att}(\langle m, a, b, t_A\rangle^4, \langle m, a, b, t_A\rangle^4)\\
f_3 &= \mathsf{att}(\langle m, a, b, c_0^\star, t_B\rangle^4, \langle m, a, b, c_0^\star, t_B\rangle^4)\\
f_4 &= \mathsf{att}(\langle m, \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}), \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{bs})\rangle^4,\\
&\qquad \langle m, \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}), \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{bs})\rangle^4)
\end{aligned}$$

Considérons donc le plan r_1, r_2 avec r_1 (resp. r_2) une concrétisation positive (resp. négative) de la règle de projection (resp. déchiffrement symétrique) :

$$\begin{aligned}
r_1 &= f_4 \to \mathsf{att}(\mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}), \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}))\\
r_2 &= \mathsf{att}(\mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as}), \mathsf{senc}(\langle n_a, k_{ab}\rangle^2, k_{as})), \mathsf{att}(k_{as}, k) \to \mathsf{bad}
\end{aligned}$$

$r_1.r_2$ est bien un plan de bad.

Grâce au lemme 5.3, nous pouvons nous contenter des règles suivantes pour les tests d'équivalence statique :

$$\begin{aligned}
\mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} &= \{\mathsf{att}(u, v) \to \mathsf{bad} \mid u \text{ est un atome mais } v \text{ n'en est pas un.}\}\\
\mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}} &= \{\mathsf{att}(\mathsf{pub}(u), v) \to \mathsf{bad} \mid v \text{ n'est pas de la forme } \mathsf{pub}(v')\}\\
\mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} &= \{\mathsf{att}(\mathsf{sign}(u_1, u_2), v_1), \mathsf{att}(\mathsf{vk}(u_2), v_2) \to \mathsf{bad} \mid \mathsf{check}(v_1, v_2)\downarrow \text{ n'est pas un } \Sigma_0^{\#}\text{-message. }\}\\
\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} &= \{\mathsf{att}(u_1, v_1), \mathsf{att}(u_1, v_2) \to \mathsf{bad} \mid v_1 \neq v_2\}\\
\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} &= \{\mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k), \mathsf{att}(C[u_1, \ldots, u_k], v) \to \mathsf{bad} \mid\\
&\qquad C \text{ est un contexte constructeur non vide,}\\
&\qquad C[u_1, \ldots, u_k] \in St_{\mathsf{opti}}(\phi) \cup \Sigma_0^{\#}, \text{ et } v \neq C[v_1, \ldots, v_k]\}
\end{aligned}$$

Les règles de l'ensemble $\mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}}$ modélisent la capacité de l'attaquant à détecter l'atomicité. Celles de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$ représentent la capacité de l'attaquant à savoir si un terme est une clé publique. La vérification des signatures est modélisée par $\mathsf{R}_{\mathsf{fail}}^{\mathsf{check}}$ et les tests d'égalité par $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1}$ et $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2}$. L'ensemble $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1}$ représente l'égalité entre termes obtenus directement par des recettes destructrices, tandis

que $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2}$ modélise le cas où l'ajout d'un contexte est nécessaire. L'algorithme de planification lui-même n'évalue pas les conditions sur les règles : c'est la fonction qui implémente l'oracle de planification (voir section 4.1.2) qui effectue ce calcul.

Soient ϕ et ψ deux $\Sigma_0^\#$-trames avec $dom(\phi) = dom(\psi)$, et construites en utilisant des constantes de l'ensemble $\Sigma_0 \subseteq \Sigma_0^\#$. L'ensemble des faits associés à ϕ et ψ est défini par :

$$\mathsf{Fact}_{\Sigma_0}(\phi, \psi) = \{\mathsf{att}(a, a) \mid a \in \Sigma_0\} \cup \{\mathsf{att}(\mathsf{w}\phi, \mathsf{w}\psi) \mid \mathsf{w} \in dom(\phi)\}$$

La *profondeur* $\mathsf{depth}(t)$ d'un terme t est la longueur maximale d'une position de t. La profondeur $\mathsf{depth}(\phi)$ d'une $\Sigma_0^\#$-trame ϕ est la profondeur du terme le plus profond de son image, c'est-à-dire $\mathsf{depth}(\phi) = \mathsf{max}_{t \in img(\phi)}\mathsf{depth}(t)$. La proposition suivante stipule que ces règles suffisent à exprimer le pouvoir de l'attaquant passif dans le langage de la planification, et donne une borne sur la longueur du plan nécessaire pour y parvenir.

Proposition 5.1. *Soit $\Sigma_0 \subset \Sigma_0^\#$ un ensemble fini de constantes. Soient ϕ et ψ deux Σ_0-trames avec $dom(\phi) = dom(\psi)$, et le système de planification $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi), \mathsf{R}\rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$$

Considérons le problème de planification $\Pi = \langle \Theta, \{\mathsf{bad}\}\rangle$. Nous avons $\phi \not\sqsubseteq_s \psi$ si, et seulement si, Π a une solution de longueur au plus $(N + 1) \times \mathsf{depth}(\phi) + 1$ où N est le nombre de noms n apparaissant dans ϕ en position de clé, c'est-à-dire tels qu'il existe un terme t avec $\mathsf{senc}(t, n) \in St(\phi)$ ou $\mathsf{aenc}(t, \mathsf{pub}(n)) \in St(\phi)$.

L'idée de la preuve est la suivante : chaque règle imite un destructeur ou un test, et il est donc facile de traduire les recettes sous forme de règles ou inversement. Quant à la borne, une fois que toutes les clés nécessaires ont été déduites, le plan minimal qui témoigne de la non-inclusion contient au plus $\mathsf{depth}(\phi)$ règles. Nous avons donc besoin au maximum de $\mathsf{depth}(\phi)$ règles pour dériver une telle clé déductible. Le raisonnement est exposé plus précisément dans la section suivante.

5.2.3 Correction et complétude

L'objet de cette section est d'exposer le raisonnement pour démontrer la proposition 5.1. Les preuves formelles sont laissées de côté et peuvent être trouvées dans la section 5.2.4. Pour commencer, nous montrons que, pour tout plan utilisant exclusivement $\mathsf{R}_{\mathsf{ana}}$, il existe une recette destructrice équivalente. La démonstration, qui est exposée formellement en section 5.2.4, est directe : chaque règle de $\mathsf{R}_{\mathsf{ana}}$ correspond à un destructeur précis.

Lemme 5.4. *Soient $\Sigma_0 \subseteq \Sigma_0^\#$ un ensemble fini et ϕ et ψ deux Σ_0-trames de même domaine. Soient*

$$\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi), \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})\rangle$$

et $\Pi = \langle \Theta, \{\mathsf{att}(u, v)\}\rangle$ pour certains Σ_0-messages u, v. Si Π admet une solution alors il existe une Σ_0-recette destructrice R telle que $R\phi{\downarrow} = u$ et $R\psi{\downarrow} = v$.

Ensuite, il s'agit de démontrer la réciproque du lemme 5.4, tout en donnant une borne sur la longueur du plan associé à une recette. Pour cela, pour toute $\Sigma_0^\#$-recette destructrice R, nous notons $|R|_{\mathsf{main}}$ la longueur de son *chemin principal*, c'est-à-dire de la plus longue séquence de 1 qui correspond à une position de R. Démontrons d'abord que cette longueur est bornée par la profondeur d'une trame ϕ dès que $R\phi{\downarrow}$ est un message.

Lemme 5.5. *Soient ϕ une $\Sigma_0^\#$-trame, et R une $\Sigma_0^\#$-recette destructrice telle que $R\phi{\downarrow}$ est un $\Sigma_0^\#$-message. Nous avons alors $|R|_{\mathsf{main}} \leqslant \mathsf{depth}(\phi)$.*

L'idée derrière ce lemme est que chaque destructeur de la recette supprime un constructeur du terme sur le chemin principal.

Nous démontrons alors que, si nous connaissons toutes les clés nécessaires pour déduire un ensemble de messages, il suffit d'un plan de longueur au plus $\mathsf{depth}(\phi)$ pour les déduire tous. Pour ce faire, notons $\mathsf{Key}(R)$ l'ensemble des *recettes de clé* d'une $\Sigma_0^{\#}$-recette R, c'est-à-dire l'ensemble des recettes qui apparaissent en position de clé dans R (comme second argument de sdec ou adec), à l'exception de celles qui sont de la forme $a \in \Sigma_0$. Soit $\mathcal{R}$ un ensemble de recettes. L'ensemble $\mathsf{Key}(\mathcal{R})$ est l'union des $\mathsf{Key}(R)$ pour les recettes $R \in \mathcal{R}$, et $|\mathcal{R}|$ le cardinal de $\mathcal{R}$.

Lemme 5.6. *Soient $\Sigma_0 \subseteq \Sigma_0^{\#}$ un ensemble fini, ϕ et ψ deux Σ_0-trames de même domaine, et $\mathcal{R}$ un ensemble de Σ_0-recettes destructrices telles que $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des Σ_0-messages pour tout $R \in \mathcal{R}$. Soient $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi,\psi) \cup \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathsf{Key}(\mathcal{R})\}, \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})\rangle$, et $\Pi = \langle \Theta, \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}\}\rangle$. Le problème Π admet une solution de longueur au plus $\mathsf{depth}(\phi)$.*

Une fois que toutes les clés sont déduites, seuls les messages sur le chemin principal de la recette restent à déduire. Ces déductions se font par un plan de longueur au plus $\mathsf{depth}(\phi)$ puisque nous avons vu que le chemin principal d'une recette était au plus $\mathsf{depth}(\phi)$. Les déductions de recettes différentes peuvent s'effectuer en parallèle puisqu'elles ne dépendent pas les unes des autres. Le lemme suivant récapitule les lemmes précédents et permet de commencer par déduire les clés, puis de déduire les autres termes déductible. Chaque clé coûte au plus $\mathsf{depth}(\phi)$ étapes. Ce lemme nous sera également utile en présence d'un attaquant actif, dans la section 5.3.

Lemme 5.7. *Soient $\Sigma_0 \subset \Sigma_0^{\#}$ un ensemble fini, ϕ et ψ deux Σ_0-trames de même domaine. Soit $\mathcal{R}$ un ensemble de Σ_0-recettes tel que $\mathsf{Key}(\mathcal{R}) \subseteq \mathcal{R}$, et $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des Σ_0-messages pour chaque $R \in \mathcal{R}$. Soient $\hat{\mathsf{F}} = \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}\}$, $\mathcal{R}_0 \subseteq \mathcal{R}$ et $\hat{\mathsf{F}}_0 = \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}_0\}$. Soient $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi,\psi) \cup \hat{\mathsf{F}}_0, \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})\rangle$ et $\Pi = \langle \Theta, \hat{\mathsf{F}}\rangle$. Le problème Π admet une solution de longueur au plus $|\mathcal{R} \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi)$.*

Il reste donc à en déduire que tout témoin de non-inclusion statique donne un plan de bad dans le langage de la planification. De plus, la longueur de ce plan est bornée en tenant compte du parallélisme, afin de donner une limite relativement faible au nombre d'itérations de l'algorithme du graphe de planification. Il faut $\mathsf{depth}(\phi)$ étapes pour déduire chaque clé, et encore $\mathsf{depth}(\phi)$ pour déduire un terme quelconque, ce qui explique la borne de $(k+1)\mathsf{depth}(\phi)$ étapes, plus une étape pour effectuer un test de non inclusion.

Lemme 5.8. *Soient $\Sigma_0 \subset \Sigma_0^{\#}$ un ensemble fini, ϕ et ψ deux Σ_0-trames de même domaine. Soit $\mathcal{R}_0$ un ensemble de Σ_0-recette destructrices tel que $\mathsf{Key}(\mathcal{R}_0) \subseteq \mathcal{R}_0$, et $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des Σ_0-messages pour chaque recette $R \in \mathcal{R}_0$. Soit $\hat{\mathsf{F}} = \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}_0\}$ et $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi,\psi) \cup \hat{\mathsf{F}}, \mathsf{R}\rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}^1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}^2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$$

Soit $\Pi = \langle \Theta, \{\mathsf{bad}\}\rangle$. Supposons que $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$. Soit T un test d'inclusion statique pour $\sqsubseteq_s^{\mathsf{simple}^+}$ vrai dans ϕ mais faux dans ψ. Alors Π admet une solution de longueur au plus $(k+1)\times\mathsf{depth}(\phi)+1$ où $k = |\mathsf{Key}(T) \setminus \mathcal{R}_0|$.

La preuve est laissée pour la section 5.2.4, mais elle découle d'une disjonction de cas sur le test T et de l'utilisation des lemmes précédents. À ce moment, nous avons tous les éléments pour déduire la proposition 5.1 que nous rappelons. Cette propriété garantit que nous n'avons pas changé la puissance de l'attaquant dans la traduction.

Proposition 5.1. *Soit $\Sigma_0 \subset \Sigma_0^\#$ un ensemble fini de constantes. Soient ϕ et ψ deux Σ_0-trames avec $dom(\phi) = dom(\psi)$, et le système de planification $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi), \mathsf{R} \rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$$

Considérons le problème de planification $\Pi = \langle \Theta, \{\mathsf{bad}\} \rangle$. Nous avons $\phi \not\sqsubseteq_s \psi$ si, et seulement si, Π a une solution de longueur au plus $(N+1) \times \mathsf{depth}(\phi) + 1$ où N est le nombre de noms n apparaissant dans ϕ en position de clé, c'est-à-dire tels qu'il existe un terme t avec $\mathsf{senc}(t, n) \in St(\phi)$ ou $\mathsf{aenc}(t, \mathsf{pub}(n)) \in St(\phi)$.

La preuve formelle est donnée en section 5.2.4. D'après le lemme 5.2, il suffit de montrer que $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$ si, et seulement si, Π a une solution. Dans le sens direct, nous considérons un test d'inclusion statique T avec $\mathsf{Key}(T)$ minimal. Le lemme 5.8 donne un plan de longueur $(|\mathsf{Key}(T)| + 1) \times \mathsf{depth}(\phi) + 1$. Il suffit alors de montrer que $|\mathsf{Key}(T)| \leqslant N$, ce qui provient du fait que les clés sont nécessairement des nonces qui apparaissent dans la trame (ou alors sont connues de l'attaquant dès le départ), et de la minimalité de T.

Réciproquement, considérons un plan de longueur minimale. En particulier, toutes les règles sauf éventuellement la dernière sont des règles de $\mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})$. Il s'agit alors d'appliquer le lemme 5.4 pour relever ce plan sous forme de recettes, puis de compléter le test d'équivalence statique suivant la dernière règle utilisée.

5.2.4 Démonstrations

Cette section contient les preuves détaillées des différents résultats énoncés précédemment. Nous commençons par démontrer le lemme 5.2 :

Lemme 5.2. *Soient ϕ et ψ deux $\Sigma_0^\#$-trames avec $dom(\phi) = dom(\psi)$. On a :*

$$\phi \sqsubseteq_s \psi \Leftrightarrow \phi \sqsubseteq_s^{\mathsf{simple}} \psi \Leftrightarrow \phi \sqsubseteq_s^{\mathsf{simple}^+} \psi$$

Démonstration. Il est facile de voir que $\phi \sqsubseteq_s \psi \Rightarrow \phi \sqsubseteq_s^{\mathsf{simple}} \psi \Rightarrow \phi \sqsubseteq_s^{\mathsf{simple}^+} \psi$. En particulier, il a été montré dans le lemme 2.16 que l'inclusion $\phi \sqsubseteq_s \psi$ permettait à l'attaquant de tester l'atomicité. Donc, nous considérerons seulement les deux autres implications.

Première implication : $\phi \sqsubseteq_s^{\mathsf{simple}} \psi \Rightarrow \phi \sqsubseteq_s \psi$. Considérons une définition alternative de l'inclusion statique, notée $\sqsubseteq'_s$, similaire à $\sqsubseteq_s^{\mathsf{simple}}$, mais en considérant des recettes arbitraires au lieu de recettes fortement simples ou destructrices.

Il est clair que $\phi \sqsubseteq'_s \psi \Rightarrow \phi \sqsubseteq_s \psi$, et il suffit donc de montrer que $\phi \sqsubseteq_s^{\mathsf{simple}} \psi \Rightarrow \phi \sqsubseteq'_s \psi$.

Étant donné un test arbitraire T vrai dans ϕ, montrons que T est aussi vrai dans ψ, en supposant que tous les tests plus petits que T ont déjà été transférés de ϕ à ψ.

Considérons la mesure suivante où $|R|$ est la taille de R, c'est-à-dire le nombre de symboles de fonctions de R.

1. T est une recette (message/message atomique/clé publique) : $\mu(T) = |R|$
2. T est une paire de recettes (égalité/signature) : $\mu(T) = |R| + |R'|$.

R est une recette telle que $R\phi\downarrow$ est un $\Sigma_0^\#$-message (respectivement un $\Sigma_0^\#$-message atomique).

— Cas où R *n'est pas* en forme normale pour $\twoheadrightarrow$. Considérons R' telle que $R \twoheadrightarrow R'$. Le terme $R'\phi\downarrow$ est un $\Sigma_0^\#$-message (lemme 2.2). Par hypothèse de récurrence $R'\psi\downarrow$ est également un $\Sigma_0^\#$-message. Il reste à montrer que $R\psi\downarrow$ est un $\Sigma_0^\#$-message. Nous avons $R = C[\mathsf{adec}(\mathsf{aenc}(R_1, R_2), R_3)]$ et $R' = C[R_1]$ (les autres cas sont similaires). Puisque $R\phi\downarrow$ est un $\Sigma_0^\#$-message, nécessairement $R_2\phi\downarrow = \mathsf{pub}(R_3)\phi\downarrow$. Par hypothèse de récurrence $R_2\psi\downarrow = \mathsf{pub}(R_3)\psi\downarrow$, ce qui permet de conclure.

— Cas où R est en forme normale pour $\twoheadrightarrow$. Dans ce cas, R est fortement simple (lemme 5.1), c'est-à-dire $R = C[R_1, \ldots, R_k]$, où C est un contexte construit sur Σ_c et chaque R_i est une $\Sigma_0^\#$-recette destructrice. Si C est vide, alors R est destructrice. L'hypothèse permet de conclure. Sinon $R = \mathsf{f}(R'_1, \ldots, R'_n)$. Par hypothèse de récurrence, le terme $R'_i\psi\!\downarrow$ est un $\Sigma_0^\#$-message ($1 \leqslant i \leqslant n$). Montrons que $C[R_1, \ldots, R_k]\psi\!\downarrow = \mathsf{f}(R'_1, \ldots, R'_n)\psi\!\downarrow$ est un $\Sigma_0^\#$-message. Par hypothèse de récurrence, les $\Sigma_0^\#$-messages aux positions de clé sont atomiques. Si $\mathsf{f} = \mathsf{aenc}$ (et donc $n = 2$), il faut garantir que $R'_2\psi\!\downarrow$ est de la forme $\mathsf{pub}(s)$, ce qui se déduit de la définition 5.1 (item 4).

R et R' sont des $\Sigma_0^\#$-recettes, $R\phi\!\downarrow$, $R'\phi\!\downarrow$ sont des $\Sigma_0^\#$-messages, et $R\phi\!\downarrow = R'\phi\!\downarrow$.

— Cas où R (resp. R') *n'est pas* en forme normale pour $\twoheadrightarrow$. Soit $R'' = R\!\downarrow$. Puisque $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des $\Sigma_0^\#$-messages, $R''\phi\!\downarrow = R\phi\!\downarrow$ et $R''\psi\!\downarrow = R\psi\!\downarrow$. Il en découle que $R''\phi\!\downarrow = R\phi\!\downarrow = R'\phi\!\downarrow$. Par hypothèse de récurrence appliquée au test $R'' = R'$, $R''\psi\!\downarrow = R'\psi\!\downarrow$, et donc $R\psi\!\downarrow = R'\psi\!\downarrow$.

— Cas où R et R' sont fortement simples, c'est-à-dire $R = C[R_1, \ldots, R_k]$ et $R' = C'[R'_1, \ldots, R'_\ell]$, où C, C' sont des contextes constructeurs et R_i ($1 \leqslant i \leqslant k$), ainsi que R'_j ($1 \leqslant j \leqslant \ell'$) sont des $\Sigma_0^\#$-recette destructrices. Si aucun des deux contextes C et C' n'est vide (c'est-à-dire, ni R ni R' ne sont destructrices) alors $\mathsf{root}(R) = \mathsf{root}(R')$, ce qui permet de conclure par hypothèse de récurrence. Sinon, l'hypothèse permet de conclure.

R et R' sont des $\Sigma_0^\#$-recettes, $R\phi\!\downarrow = \mathsf{sign}(t, s)$, et $R'\phi\!\downarrow = \mathsf{vk}(s)$ pour un certain terme t et un certain atome s.

— Cas où R (resp. R') *n'est pas* en forme normale forcée. $R\!\downarrow$ (resp. $R'\!\downarrow$) est une plus petite recette que R (resp. R'), c'est-à-dire $\mu(R\!\downarrow) < \mu(R)$. Par le lemme 2.2, $R\!\downarrow\phi\!\downarrow = R\phi\!\downarrow$ (resp. $R'\!\downarrow\phi\!\downarrow = R'\phi\!\downarrow$). Donc $R\!\downarrow, R'$ (resp. $R, R'\!\downarrow$) donne un plus petit test que R, R'. Par hypothèse de récurrence on obtient $R\!\downarrow\psi\!\downarrow = \mathsf{sign}(t', s')$ et $R'\psi\!\downarrow = \mathsf{vk}(s')$ pour un certain terme t et un certain atome s. Le cas où $R\phi\!\downarrow$ est un $\Sigma_0^\#$-message a déjà été considéré, donc supposons que $R\psi\!\downarrow$ est un $\Sigma_0^\#$-message. Alors $R\psi\!\downarrow = R\!\downarrow\psi\!\downarrow$ par le lemme 2.2, ce qui conclut ce cas.

— Cas où R et R' sont deux recettes fortement simples. Si ce sont deux $\Sigma_0^\#$-recette destructrices, l'hypothèse permet de conclure. Sinon, supposons $R = \mathsf{sign}(R_1, R_2)$. Dans ce cas, $\mathsf{vk}(R_2) = R'$ et ce test est plus petit que $R = R'$, et comme il est vrai dans ϕ, il se transfère à ψ par hypothèse de récurrence. Il en découle que $R = R'$ est vrai dans ψ. Supposons maintenant que $R' = \mathsf{vk}(R'_1)$. Comme $R'_1\phi\!\downarrow$ est un $\Sigma_0^\#$-message atomique, R'_1 est destructrice. R' est destructrice, donc $\mathsf{sign}(\mathsf{getmsg}(R), R')$ est fortement simple. R est destructrice et le test d'égalité $\mathsf{sign}(\mathsf{getmsg}(R), R'_1) = R$ est vrai dans ϕ. Par hypothèse, il se transfère à ψ, ce qui implique $R\psi\!\downarrow = \mathsf{sign}(t', s')$ avec $R'_1\psi\!\downarrow = \mathsf{vk}(s')$.

R est une recette telle que $R\phi\!\downarrow = \mathsf{pub}(s)$ pour un certain atome s.

— Cas où R *n'est pas* en forme normale forcée. $R\!\downarrow$ est une plus petite recette que R. D'après le lemme 2.2, $R\!\downarrow\phi\!\downarrow = R\phi\!\downarrow$. Donc par hypothèse de récurrence $R\!\downarrow\psi\!\downarrow = \mathsf{pub}(s)$ pour un certain atome s. Il a déjà été montré que, puisque $R\phi\!\downarrow$ est un $\Sigma_0^\#$-message, $R\psi\!\downarrow$ est aussi un $\Sigma_0^\#$-message. Donc d'après le lemme 2.2, $R\!\downarrow\phi\!\downarrow = R\phi\!\downarrow = \mathsf{pub}(s)$.

— Cas où R est une $\Sigma_0^\#$-recette fortement simple. Dans le cas où R est une $\Sigma_0^\#$-recette destructrice, l'hypothèse permet de conclure. Sinon $R = \mathsf{pub}(R_1)$ et $R_1\phi\!\downarrow$ est un atome, donc R_1 est destructrice. Le message $R_1\psi\!\downarrow$ est également un atome, d'après l'hypothèse de récurrence. Donc $R\psi\!\downarrow = \mathsf{pub}(s')$ pour un certain atome s'.

Seconde implication : $\phi \sqsubseteq_s^{\mathsf{simple}^+} \psi \;\Rightarrow \phi \sqsubseteq_s^{\mathsf{simple}} \psi$. Soit R une $\Sigma_0^\#$-recette fortement simple avec un contexte constructeur C et R' une $\Sigma_0^\#$-recette destructrice telle que $R\phi\!\downarrow$ et $R'\phi\!\downarrow$ sont deux

$\Sigma_0^{\#}$-messages. Supposons que $R\phi{\downarrow} = R'\phi{\downarrow}$. Il faut montrer que $R\psi{\downarrow} = R'\psi{\downarrow}$. Prouvons qu'un tel test se transfère de ϕ à ψ par récurrence sur $\#_{\mathsf{senc}}(R)$ (nombre d'occurences de senc et de $\langle \ldots \rangle^k$ à partir de la racine de C), et $\#_{\mathsf{adec}}(R')$ (nombre d'occurences de adec à partir de la racine de R') ordonnées lexicographiquement.

Initialisation : $(\#_{\mathsf{senc}}(R), \#_{\mathsf{adec}}(R')) = (0, 0)$. Dans ce cas, le test se transfère par hypothèse.

Hérédité : $\#_{\mathsf{senc}}(R) \geqslant 1$. Dans ce cas, considérons $R = \mathsf{senc}(R_1, R_2)$ et le test $R_1 = \mathsf{sdec}(R', R_2)$. Comme $(R_1 = \mathsf{sdec}(R', R_2))\phi$, l'hypothèse de récurrence donne $(R_1 = \mathsf{sdec}(R', R_2))\psi$. Donc $(R' = \mathsf{senc}(R_1, R_2))\psi$, c'est-à-dire $(R = R')\psi$. Le cas où $R = \langle R_1, R_2 \rangle$ est similaire.

Maintenant, considérons le cas où $\#_{\mathsf{senc}}(R) = 0$ mais $\#_{\mathsf{adec}}(R') \geqslant 1$. Dans ce cas, $R' = \mathsf{adec}(R'_1, R'_2)$. Considérons le test $\mathsf{aenc}(R, \mathsf{pub}(R'_2)) = R'_1$. Alors $\#_{\mathsf{senc}}(\mathsf{aenc}(R, \mathsf{pub}(R'_2))) = 0$, et $\#_{\mathsf{adec}}(R'_1) < \#_{\mathsf{adec}}(R')$. $(\mathsf{aenc}(R, \mathsf{pub}(R'_2)) = R'_1)\phi$ donc par hypohtèse de récurrence $(\mathsf{aenc}(R, \mathsf{pub}(R'_2)) = R'_1)\psi$. Il en découle que $(R = \mathsf{adec}(R'_1, R'_2))\psi$, c'est-à-dire $(R = R')\psi$. □

Pour la suite, nous aurons besoin de la notion de sous-terme en position déductible d'un terme t. Ainsi, l'ensemble $St_{\mathsf{ded}}(t)$ est défini récursivement par :

- $St_{\mathsf{ded}}(\langle t_1, \ldots, t_k \rangle^k) = \{\langle t_1, \ldots, t_k \rangle^k\} \cup St_{\mathsf{ded}}(t_1) \cup \cdots \cup St_{\mathsf{ded}}(t_k)$
- $St_{\mathsf{ded}}(\mathsf{f}(t_1, t_2)) = \{\mathsf{f}(t_1, t_2)\} \cup St_{\mathsf{ded}}(t_1)$ pour $\mathsf{f} \in \{\mathsf{senc}, \mathsf{aenc}, \mathsf{sign}\}$
- $St_{\mathsf{ded}}(\mathsf{f}(t)) = \{\mathsf{f}(t)\}$ pour $\mathsf{f} \in \{\mathsf{hash}, \mathsf{pub}, \mathsf{vk}\}$
- $St_{\mathsf{ded}}(t) = \{t\}$ quand t est une constante ou un nom.

De cette manière, nous pouvons démontrer le lemme 5.3.

Lemme 5.3. *Soient ϕ une $\Sigma_0^{\#}$-trame, R une recette fortement simple telle que $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\rangle^k \mid 2 \leqslant k \leqslant n\}$ et R' une recette destructrice telle que $\mathsf{root}(R') \neq \mathsf{adec}$. Supposons que $R\phi{\downarrow}$ et $R'\phi{\downarrow}$ soient deux $\Sigma_0^{\#}$-messages tels que $R\phi{\downarrow} = R'\phi{\downarrow}$. Ou bien C est le contexte vide, ou bien $R\phi{\downarrow} \in St_{\mathsf{opti}}(\phi) \cup \Sigma_0^{\#}$.*

Démonstration. Soient ϕ une $\Sigma_0^{\#}$-trame, R' une recette destructrice telle que $R'\phi{\downarrow}$ est un $\Sigma_0^{\#}$-message. Montrons que $R'\phi{\downarrow} \in St_{\mathsf{ded}}(\phi) \cup \Sigma_0^{\#}$ par récurrence structurelle sur R'.

Initialisation : $R' = \mathsf{w} \in dom(\phi)$ *ou* $R' = a \in \Sigma_0^{\#}$. Dans les deux cas, $R'\phi{\downarrow} \in St_{\mathsf{ded}}(\phi) \cup \Sigma_0^{\#}$.

Hérédité : $R' = \mathsf{g}(R'_1, \ldots, R'_k)$. Distinguons plusieurs cas suivant la valeur de g. Si $\mathsf{g} = \mathsf{sdec}$, alors $k = 2$, $R'_1\phi{\downarrow} = \mathsf{senc}(t_1, t_2)$, $R'_2\phi{\downarrow} = t_2$, et $R'\phi{\downarrow} = t_1$ pour certains termes t_1, t_2. D'après l'hypothèse de récurrence appliquée à R'_1, $R'_1\phi{\downarrow} \in St_{\mathsf{ded}}(\phi) \cup \Sigma_0^{\#}$, donc $R'\phi{\downarrow} \in St_{\mathsf{ded}}(\phi) \cup \Sigma_0^{\#}$. Les cas où $R' = \langle R'_1, R'_2 \rangle$, $R' = \mathsf{getmsg}(R'_1)$, et $R' = \mathsf{adec}(R'_1, R'_2)$ sont similaires.

Maintenant, montrons le lemme. Soient $R = C[R_1, \ldots, R_k]$ une recette fortement simple telle que $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\,\rangle_k \mid 2 \leqslant k \leqslant n\}$, et R' une recette destructrice telle que $\mathsf{root}(R') \neq \mathsf{adec}$. Grâce au résultat que nous venons de démontrer, nous avons $R'\phi{\downarrow} \in St_{\mathsf{ded}}(\phi) \cup \Sigma_0^{\#}$. De plus, soit *(i)* $R' = \mathsf{w} \in dom(\phi)$, soit *(ii)* $R' = a \in \Sigma_0^{\#}$ soit *(iii)* $R' = \mathsf{g}(R'_1, \ldots, R'_k)$ pour un certain $\mathsf{g} \in \Sigma'_d \setminus \{\mathsf{adec}\}$. Dans les cas *(i)* et *(ii)*, il est facile de conclure que C est vide ou $R\phi{\downarrow} = C[R_1\phi{\downarrow}, \ldots, R_k\phi{\downarrow}] \in St_{\mathsf{opti}}(\phi) \cup \Sigma_0^{\#}$. Dans le cas *(iii)*, supposons que $R' = \mathsf{sdec}(R'_1, R'_2)$ (et donc $\mathsf{g} = \mathsf{sdec}$). Il en découle que $R'_1\phi{\downarrow} \in St_{\mathsf{ded}}(\phi)$ et comme $R'_1\phi{\downarrow} = \mathsf{senc}(t_1, t_2)$, nous avons $R'_1\phi{\downarrow} \in St_{\mathsf{opti}}(\phi)$, et donc $R'\phi{\downarrow} \in St_{\mathsf{opti}}(\phi)$ puisque la racine de $R'\phi{\downarrow} = R\phi{\downarrow}$ n'est ni un symbole de $\{\langle\,\rangle_k \mid 2 \leqslant k \leqslant n\}$, ni senc. Le cas où $\mathsf{g} \in \{\langle\,\rangle_k \mid 2 \leqslant k \leqslant n\} \cup \{\mathsf{getmsg}\}$ peut être traité de manière semblable (par hypothèse, $\mathsf{g} \neq \mathsf{adec}$). □

Passons à la démonstration du lemme 5.4.

Lemme 5.4. *Soient $\Sigma_0 \subseteq \Sigma_0^\#$ un ensemble fini et ϕ et ψ deux Σ_0-trames de même domaine. Soient*

$$\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi), \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}}) \rangle$$

et $\Pi = \langle \Theta, \{\mathsf{att}(u, v)\} \rangle$ pour certains Σ_0-messages u, v. Si Π admet une solution alors il existe une Σ_0-recette destructrice R telle que $R\phi\downarrow = u$ et $R\psi\downarrow = v$.

Démonstration. Soit $\pi = r_1, \ldots, r_n$ un chemin linéaire de S_0 à S_n, et $\mathsf{att}(u, v) \in S_n$. Le résultat se montre par récurrence sur la longueur de π.

Initialisation. Le plan π est vide. Dans ce cas, le résultat découle directement de la définition de S_0.

Hérédité. La règle r_n est une instance de l'une des règles abstraites de $\mathsf{R}_{\mathsf{ana}}$, par exemple

$$\mathsf{att}(\mathsf{senc}(u_1, u_2), \mathsf{senc}(v_1, v_2)),\ \mathsf{att}(u_2, v_2) \to \mathsf{att}(u_1, v_1)$$

D'après l'hypothèse de récurrence, il existe :

— une Σ_0-recette destructrice R_1 telle que $R_1\phi\downarrow = \mathsf{senc}(u_1, u_2)$, et $R_1\psi\downarrow = \mathsf{senc}(v_1, v_2)$;

— une Σ_0-recette destructrice R_2 telle que $R_2\phi\downarrow = u_2$, et $R_2\psi\downarrow = v_2$.

Donc, la Σ_0-recette $R = \mathsf{sdec}(R_1, R_2)$ permet de conclure. Le raisonnement est semblable pour les autres règles. □

Démontrons également le lemme 5.5.

Lemme 5.5. *Soient ϕ une $\Sigma_0^\#$-trame, et R une $\Sigma_0^\#$-recette destructrice telle que $R\phi\downarrow$ est un $\Sigma_0^\#$-message. Nous avons alors $|R|_{\mathsf{main}} \leqslant \mathsf{depth}(\phi)$.*

Démonstration. Établissons d'abord que $|R|_{\mathsf{main}} + \mathsf{depth}(R\phi\downarrow) \leqslant \mathsf{depth}(\phi)$ pour chaque $\Sigma_0^\#$-recette R telle que $R\phi\downarrow$ est un $\Sigma_0^\#$-message par récurrence structurelle sur R.

Initialisation. $R = \mathsf{w}$ ou $R \in \Sigma_0$. Dans ce cas, nous avons :

$$|R|_{\mathsf{main}} + \mathsf{depth}(R\phi\downarrow) = 0 + \mathsf{depth}(R\phi\downarrow) \leqslant \mathsf{depth}(\phi).$$

Hérédité. $R = \mathsf{g}(R_1, \ldots, R_k)$ pour un certain $\mathsf{g} \in \Sigma_d$. Puisque $R\phi\downarrow$ est un Σ_0-message, le terme $R_1\phi\downarrow$ est un Σ_0-message. Donc, l'hypothèse de récurrence s'applique et $|R_1|_{\mathsf{main}} + \mathsf{depth}(R_1\phi\downarrow) \leqslant \mathsf{depth}(\phi)$. De plus, $\mathsf{depth}(R_1\phi\downarrow) \geqslant 1 + \mathsf{depth}(R\phi\downarrow)$, et $|R|_{\mathsf{main}} = |R_1|_{\mathsf{main}} + 1$. Donc :

$$|R|_{\mathsf{main}} + \mathsf{depth}(R\phi\downarrow) \leqslant |R_1|_{\mathsf{main}} + 1 + \mathsf{depth}(R_1\phi\downarrow) - 1 \leqslant \mathsf{depth}(\phi).$$

Nous avons $|R|_{\mathsf{main}} + \mathsf{depth}(R\phi\downarrow) \leqslant \mathsf{depth}(\phi)$. Puisque $\mathsf{depth}(R\phi\downarrow) \geqslant 0$, nous concluons que $|R|_{\mathsf{main}} \leqslant \mathsf{depth}(\phi)$. □

Il est maintenant possible de démontrer le lemme 5.6.

Lemme 5.6. *Soient $\Sigma_0 \subseteq \Sigma_0^\#$ un ensemble fini, ϕ et ψ deux Σ_0-trames de même domaine, et $\mathcal{R}$ un ensemble de Σ_0-recettes destructrices telles que $R\phi\downarrow$ et $R\psi\downarrow$ sont des Σ_0-messages pour tout $R \in \mathcal{R}$. Soient $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi) \cup \{\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \mid R \in \mathsf{Key}(\mathcal{R})\}, \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}}) \rangle$, et $\Pi = \langle \Theta, \{\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \mid R \in \mathcal{R}\} \rangle$. Le problème Π admet une solution de longueur au plus $\mathsf{depth}(\phi)$.*

Démonstration. Montrons l'existence d'une solution de longueur au plus $\ell = \mathsf{max}\{|R|_{\mathsf{main}} \mid R \in \mathcal{R}\}$, ce qui permettra de conclure, grâce au lemme 5.5 que $\ell \leqslant \mathsf{depth}(\phi)$.

Initialisation. $\ell = 0$. Dans ce cas, $R \in dom(\phi) \cup \Sigma_0$ pour chaque $R \in \mathcal{R}$, et le chemin vide de longueur 0 est une solution de Π.

Hérédité. $\ell > 0$. Soit $\mathcal{R}' = \{R \in \mathcal{R} \mid |R|_{\mathsf{main}} = \ell\} = \{R_1, \ldots, R_n\}$. Pour chaque $i \in \{1, \ldots, n\}$, nous avons :

— ou bien $R_i = \mathsf{des}(R_i^1, R_i^2)$ avec $\mathsf{des} \in \{\mathsf{adec}, \mathsf{sdec}\}$; et $R_i^1\phi\!\downarrow$ et $R_i^1\psi\!\downarrow$, ainsi que $R_i^2\psi\!\downarrow$ et $R_i^2\psi\!\downarrow$, sont des Σ_0-messages. De plus, $R_i^2 \in \mathsf{Key}(R_i)$ ou $R_i^2 \in \Sigma_0^{\#}$, donc, par hypothèse, le fait $\mathsf{att}(R_i^2\phi\!\downarrow, R_i^2\psi\!\downarrow)$ appartient à l'état initial du système de planification Θ.

— ou bien $R_i = \mathsf{des}(R_i^1)$ avec $\mathsf{des} \in \{\mathsf{proj}_\ell^k, \mathsf{getmsg}\}$; et $R_i^1\phi\!\downarrow$ et $R_i^1\psi\!\downarrow$ sont des Σ_0-messages.

De plus, $\mathsf{Key}(R_i^1) \subseteq \mathsf{Key}(R')$ pour chaque $i \in \{1, \ldots, n\}$. Donc, par hypothèse de récurrence, il existe un plan π_0 de longueur $\ell - 1$ de

$$\Pi_0 = \langle \Theta, \{\mathsf{att}(R_i^1\phi\!\downarrow, R_i^1\psi\!\downarrow) \mid 1 \leqslant i \leqslant n\} \cup \{\mathsf{att}(R'\phi\!\downarrow, R'\psi\!\downarrow) \mid R' \in \mathcal{R} \setminus \mathcal{R}'\}\rangle.$$

De plus, si $R_i = \mathsf{sdec}(R_i^1, R_i^2)$ (le cas où $R_i = \mathsf{adec}(R_i^1, R_i^2)$ se traite de manière semblable), comme $R_i\phi\!\downarrow$ et $R_i\psi\!\downarrow$ sont des Σ_0-messages, les égalités $R_i^1\phi\!\downarrow = \mathsf{senc}(R_i\phi\!\downarrow, R_i^2\phi\!\downarrow)$ et $R_i^1\psi\!\downarrow = \mathsf{senc}(R_i\psi\!\downarrow, R_i^2\psi\!\downarrow)$ sont vérifiées. Soit r_i la règle de planification :

$$\mathsf{att}(\mathsf{senc}(R_i\phi\!\downarrow, R_i^2\phi\!\downarrow), \mathsf{senc}(R_i\psi\!\downarrow, R_i^2\psi\!\downarrow)),\ \mathsf{att}(R_i^2\phi\!\downarrow, R_i^2\psi\!\downarrow) \to \mathsf{att}(R_i\phi\!\downarrow, R_i\psi\!\downarrow)$$

Nous avons :

— $\mathsf{Add}(r_i) = \{\mathsf{att}(R_i\phi\!\downarrow, R_i\psi\!\downarrow)\}$;

— $\mathsf{att}(\mathsf{senc}(R_i\phi\!\downarrow, R_i^2\phi\!\downarrow), \mathsf{senc}(R_i\psi\!\downarrow, R_i^2\psi\!\downarrow)) = \mathsf{att}(R_i^1\phi\!\downarrow, R_i^1\psi\!\downarrow)$, et

— $\mathsf{att}(R_i^2\phi\!\downarrow, R_i^2\psi\!\downarrow)$ appartient à l'état initial du système de planification Θ.

Donc chaque r_i $(1 \leqslant i \leqslant n)$ est applicable après π_0, et donc $\pi_0.\{r_1, \ldots, r_n\}$ est une solution de Π de longueur ℓ. □

Nous en déduisons le lemme 5.7.

Lemme 5.7. *Soient $\Sigma_0 \subset \Sigma_0^{\#}$ un ensemble fini, ϕ et ψ deux Σ_0-trames de même domaine. Soit $\mathcal{R}$ un ensemble de Σ_0-recettes tel que $\mathsf{Key}(\mathcal{R}) \subseteq \mathcal{R}$, et $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des Σ_0-messages pour chaque $R \in \mathcal{R}$. Soient $\hat{\mathsf{F}} = \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}\}$, $\mathcal{R}_0 \subseteq \mathcal{R}$ et $\hat{\mathsf{F}}_0 = \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}_0\}$. Soient $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi) \cup \hat{\mathsf{F}}_0, \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})\rangle$ et $\Pi = \langle \Theta, \hat{\mathsf{F}}\rangle$. Le problème Π admet une solution de longueur au plus $|\mathcal{R} \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi)$.*

Démonstration. Considérons l'ordre partiel $R < R'$ sur les Σ_0-recettes destructrices telles que $R < R'$ si, et seulement si, $R \in \mathsf{Key}(R')$. Montrons le résultat par récurrence sur $|\mathcal{R} \setminus \mathcal{R}_0|$, c'est-à-dire le nombre d'éléments de $\mathcal{R} \setminus \mathcal{R}_0$.

Initialisation. $|\mathcal{R} \setminus \mathcal{R}_0| = 0$. Dans ce cas, le plan vide de longueur 0 est une solution de $\Pi = \langle \Theta, \hat{\mathsf{F}}\rangle$.

Hérédité. $|\mathcal{R} \setminus \mathcal{R}_0| > 0$. Dans ce cas, considérons R un élément minimal de $\mathcal{R} \setminus \mathcal{R}_0$ par rapport à $<$. Les termes $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des Σ_0-messages. Comme $\mathsf{Key}(R) \subseteq \mathcal{R}$, et comme R est minimale, nous avons $\mathsf{Key}(R) \subseteq \mathcal{R}_0$. Donc le lemme 5.6 s'applique à $\{R\}$, et il existe une solution π de longueur au plus $\mathsf{depth}(\phi)$ de $\langle \Theta, \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow)\}\rangle$.

Soit $\Theta' = \langle \mathsf{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi) \cup \hat{\mathsf{F}}_0 \cup \mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow), \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})\rangle$. Par hypothèse de récurrence, il existe une solution π' de $\langle \Theta', \hat{\mathsf{F}}\rangle$ de longueur au plus $|\mathcal{R} \setminus (\mathcal{R}_0 \cup \{R\})| \times \mathsf{depth}(\phi) = |\mathcal{R} \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi) - \mathsf{depth}(\phi)$. Il en découle que $\pi.\pi'$ est une solution de Π de longueur au plus $|\mathcal{R} \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi)$. □

Nous pouvons maintenant démontrer le lemme 5.8.

Lemme 5.8. *Soient $\Sigma_0 \subset \Sigma_0^{\#}$ un ensemble fini, ϕ et ψ deux Σ_0-trames de même domaine. Soit $\mathcal{R}_0$ un ensemble de Σ_0-recette destructrices tel que $\mathsf{Key}(\mathcal{R}_0) \subseteq \mathcal{R}_0$, et $R\phi\!\downarrow$ et $R\psi\!\downarrow$ sont des Σ_0-messages pour chaque recette $R \in \mathcal{R}_0$. Soit $\hat{\mathsf{F}} = \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathcal{R}_0\}$ et $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi) \cup \hat{\mathsf{F}}, \mathsf{R}\rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}^1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}^2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$$

Soit $\Pi = \langle \Theta, \{\mathsf{bad}\}\rangle$. Supposons que $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$. Soit T un test d'inclusion statique pour $\sqsubseteq_s^{\mathsf{simple}^+}$ vrai dans ϕ mais faux dans ψ. Alors Π admet une solution de longueur au plus $(k+1)\times\mathsf{depth}(\phi)+1$ où $k = |\mathsf{Key}(T) \setminus \mathcal{R}_0|$.

Démonstration. Puisque $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$, considérons les quatre cas séparément.

1. Il existe une Σ_0-recette destructrice R telle que $R\phi\!\downarrow$ est un Σ_0-message mais $R\psi\!\downarrow$ n'en est pas un. Soit R' la plus petite sous-recette de R telle que $R'\phi\!\downarrow$ est un Σ_0-message mais $R'\psi\!\downarrow$ n'en est pas un. Puisque ϕ et ψ sont des trames, et ne contiennent donc que des messages, nous avons $R' = \mathsf{g}(R'_1, R'_2)$ avec $\mathsf{g} \in \{\mathsf{sdec}, \mathsf{adec}\}$, ou $R' = \mathsf{g}(R'_1)$ avec $\mathsf{g} \in \{\mathsf{getmsg}\} \cup \{\mathsf{proj}_j^k \mid 2 \leqslant k \leqslant n, 1 \leqslant j \leqslant k\}$. Supposons sans perte de généralité que $R' = \mathsf{sdec}(R'_1, R'_2)$, et par minimalité du test, $R'_1\psi\!\downarrow$ et $R'_2\psi\!\downarrow$ sont des messages.
   Appliquons le lemme 5.7 avec $\mathcal{R}' = \mathsf{Key}(\{R'_1, R'_2\}) \cup \{R'_2\}$ et $\mathcal{R}_0$. Il existe un plan de longueur au plus $|\mathsf{Key}(\{R'_1, R'_2\}) \cup \{R'_2\} \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi)$ qui mène à

   $$\{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \mid R \in \mathsf{Key}(\{R'_1, R'_2\}) \cup \{R'_2\}\}.$$

   Alors, d'après le lemme 5.6, $\langle \Theta, \{\mathsf{att}(R'_1\phi\!\downarrow, R'_1\psi\!\downarrow), \mathsf{att}(R'_2\phi\!\downarrow, R'_2\psi\!\downarrow)\}\rangle$ a une solution de longueur au plus

   $$|(\mathsf{Key}(\{R'_1, R'_2\}) \cup \{R'_2\}) \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi) + \mathsf{depth}(\phi).$$

   Nous considérons la règle r de la forme :

   $$\mathsf{att}(R'_1\phi\!\downarrow, R'_1\psi\!\downarrow), \mathsf{att}(R'_2\phi\!\downarrow, R'_2\psi\!\downarrow) \to \mathsf{bad}$$

   qui est une instance d'une règle de $\mathsf{Concrete}^-(\mathsf{R}_{\mathsf{ana}})$ puisque $R'_1\phi\!\downarrow = \mathsf{senc}(u_1, u_2)$ pour certains termes u_1, u_2, et $\mathsf{senc}(u_1, u_2)$, $R'_1\psi\!\downarrow$, $R'_2\psi\!\downarrow$ sont des Σ_0-messages, tandis que $\mathsf{sdec}(R'_1\psi\!\downarrow, R'_2\psi\!\downarrow)\!\downarrow$ n'est pas un Σ_0-message. La règle R s'applique et mène à bad. Puisque $\mathsf{Key}(\{R'_1, R'_2\}\cup\{R'_2\}) \subseteq \mathsf{Key}(R') \subseteq \mathsf{Key}(R) = \mathsf{Key}(T)$, le problème Π a une solution de longueur au plus

   $$(|\mathsf{Key}(T) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi) + 1.$$

   Maintenant, supposons que $R\phi\!\downarrow$ est un Σ_0-message atomique, et $R\psi\!\downarrow$ est un Σ_0-message mais n'est pas atomique. Dans ce cas, avec un raisonnement similaire, nous obtenons que le problème $\langle \Theta, \{\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow)\}\rangle$ a une solution de longueur au plus

   $$|\mathsf{Key}(T) \setminus \mathcal{R}_0| \times \mathsf{depth}(\phi) + \mathsf{depth}(\phi).$$

   Considérons alors la règle r de la forme :

   $$\mathsf{att}(R\phi\!\downarrow, R\psi\!\downarrow) \to \mathsf{bad}$$

   qui est une instance de la règle $\mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}}$, et qui mène à une solution de Π. Donc, le problème Π a une solution de longueur au plus

   $$(|\mathsf{Key}(T) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi) + 1.$$

2. Le test minimal est une Σ_0-recette destructrice R telle que $R\phi\downarrow = \mathsf{pub}(s)$ pour un certain atome s, tandis que $R\psi\downarrow \neq \mathsf{pub}(s')$ pour un certain atome s'. Premièrement, grâce au point 1, nous pouvons supposer que $R\psi\downarrow$ est un message. D'après le lemme 5.7 avec $\mathcal{R} = \mathsf{Key}(R)$ et $\mathcal{R}_0$, puis en appliquant le lemme 5.6 sur $\{R\}$, nous obtenons une solution de $\langle \Theta, \{\mathsf{att}(R\phi\downarrow, R\psi\downarrow)\}\rangle$ de longueur au plus $(|\mathsf{Key}(R)| + 1) \times \mathsf{depth}(\phi)$. Considérons ensuite la règle r de la forme :

$$\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \to \mathsf{bad}$$

qui est une instance d'une règle de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$, et qui mène à une solution du problème Π de longueur au plus $(|\mathsf{Key}(T) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi) + 1$.

3. Il existe des Σ_0-recette destructrices R et R' telles que $R\phi\downarrow = \mathsf{sign}(t, s)$, $R'\phi\downarrow = \mathsf{vk}(s)$ pour un certain Σ_0-message t, et un certain atome s, tandis que $R\psi\downarrow \neq \mathsf{sign}(t', s')$ ou $R\psi\downarrow \neq \mathsf{vk}(s')$ pour chaque Σ_0-message t' et chaque atome s'. D'après le premier point, nous pouvons supposer que $R\psi\downarrow$, et $R'\psi\downarrow$ sont des Σ_0-messages. En appliquant le lemme 5.7 avec $\mathcal{R} = \mathsf{Key}(\{R, R'\})$ et $\mathcal{R}_0$, et ensuite le lemme 5.6 sur $\{R, R'\}$, nous obtenons une solution du problème de planification $\langle \Theta, \{\mathsf{att}(R\phi\downarrow, R\psi\downarrow), \mathsf{att}(R'\phi\downarrow, R'\psi\downarrow)\}\rangle$ de longueur au plus $(|\mathsf{Key}(\{R, R'\}) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi)$. Ensuite, considérons la règle r de la forme :

$$\mathsf{att}(R\phi\downarrow, R\psi\downarrow), \mathsf{att}(R'\phi\downarrow, R'\psi\downarrow) \to \mathsf{bad}$$

qui est une instance d'une règle de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{check}}$, et mène à une solution du problème Π de longueur au plus $(|\mathsf{Key}(T) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi) + 1$.

4. Il existe une Σ_0-recette simple $R = C[R_1, \ldots, R_k]$ et une Σ_0-recette destructrice R' telle que $R\phi\downarrow = R'\phi\downarrow$ sont des Σ_0-messages tandis que $R\psi\downarrow \neq R'\psi\downarrow$. De plus, nous savons que $\mathsf{root}(R) \notin \{\mathsf{senc}\} \cup \{\langle\,\rangle_k \mid 2 \leqslant k \leqslant n\}$, et aussi que $\mathsf{root}(R') \neq \mathsf{adec}$. D'abord, grâce au premier point, nous pouvons supposer que $R_1\psi\downarrow, \ldots, R_k\psi\downarrow$, ainsi que $R\psi\downarrow$, sont des Σ_0-messages. En appliquant le lemme 5.7 sur $\mathsf{Key}(\{R_1, \ldots, R_i, R'\})$, puis le lemme 5.6 sur $\{R_1, \ldots, R_i, R'\}$, nous obtenons une solution de

$$\langle \Theta, \{\mathsf{att}(R_1\phi\downarrow, R_1\psi\downarrow), \ldots, \mathsf{att}(R_i\phi\downarrow, R_i\psi\downarrow), \mathsf{att}(R'\phi\downarrow, R'\psi\downarrow)\}\rangle$$

de longueur au plus $(|\mathsf{Key}(\{R_1, \ldots, R_i, R'\}) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi)$. Ensuite, considérons la règle r de la forme :

$$\mathsf{att}(R_1\phi\downarrow, R_1\psi\downarrow), \ldots, \mathsf{att}(R_i\phi\downarrow, R_i\psi\downarrow), \mathsf{att}(R'\phi\downarrow, R'\psi\downarrow) \to \mathsf{bad}$$

qui est une instance d'une règle de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1}$ ou de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2}$. En effet, grâce au lemme 5.3, nous savons que :

— ou bien C est le contexte vide, et donc r est une instance de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1}$ puisque $R'\phi\downarrow = R\phi\downarrow$, et $R'\psi\downarrow \neq R\psi\downarrow$;

— ou bien $R\phi\downarrow \in St_{\mathsf{opti}}(\phi) \cup \Sigma_0^{\#}$, r est une instance de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2}$ puisque :

$$R'\phi\downarrow = R\phi\downarrow = C[R_1\phi\downarrow, \ldots, R_i\phi\downarrow]$$
$$R'\psi\downarrow \neq R\psi\downarrow = C[R_1\psi\downarrow, \ldots, R_i\psi\downarrow]$$

Il est donc possible d'en conclure que le problème Π a une solution de longueur au plus

$$(|\mathsf{Key}(T) \setminus \mathcal{R}_0| + 1) \times \mathsf{depth}(\phi) + 1$$

Tous les cas ayant été traités, le résultat est démontré. □

De cette manière, nous démontrons la propriété 5.1.

Proposition 5.1. *Soit $\Sigma_0 \subset \Sigma_0^\#$ un ensemble fini de constantes. Soient ϕ et ψ deux Σ_0-trames avec $dom(\phi) = dom(\psi)$, et le système de planification $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0}(\phi, \psi), \mathsf{R} \rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$$

Considérons le problème de planification $\Pi = \langle \Theta, \{\mathsf{bad}\} \rangle$. Nous avons $\phi \not\sqsubseteq_s \psi$ si, et seulement si, Π a une solution de longueur au plus $(N+1) \times \mathsf{depth}(\phi) + 1$ où N est le nombre de noms n apparaissant dans ϕ en position de clé, c'est-à-dire tels qu'il existe un terme t avec $\mathsf{senc}(t, n) \in St(\phi)$ ou $\mathsf{aenc}(t, \mathsf{pub}(n)) \in St(\phi)$.

Démonstration. D'après le lemme 5.2, il suffit de montrer que $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$ si, et seulement si, Π a une solution. Nous montrons les deux directions séparément.

($\Rightarrow$) Supposons $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$. Considérons un test d'inclusion statique T pour $\sqsubseteq_s^{\mathsf{simple}^+}$, avec $\mathsf{Key}(T)$ minimal, et parmi ceux qui satisfont cette condition, avec une taille minimale (en nombre de symboles de fonction). Le lemme 5.8 s'applique avec $\mathcal{R}_0 = \emptyset$, donc le problème Π a une solution de longueur au plus $(|\mathsf{Key}(T)|+1) \times \mathsf{depth}(\phi)+1$. Pour conclure, nous devons montrer que $|\mathsf{Key}(T)| \leqslant N$. En fait, $\{R\phi\downarrow \mid R \in \mathsf{Key}(T)\} \setminus \Sigma_0^\# \leqslant N$ puisque toutes ces recettes apparaissent en position de clé (c'est-à-dire en second argument de $\mathsf{sdec}/\mathsf{adec}$) d'une recette qui donne un message dans ϕ. Maintenant, au cas où il y aurait deux recettes $K, K' \in \mathsf{Key}(T) \cup \Sigma_0$ telles que $K\phi\downarrow = K'\phi\downarrow$, alors par minimalité du test (remarquons que la taille du test décroît), nous savons que $K\psi\downarrow = K'\psi\downarrow$, et donc un remplacement de K par K' nous donnerait un plus petit test pour notre mesure. Nous en concluons donc que le problème Π a une solution de longueur au plus $(N+1) \times \mathsf{depth}(\phi) + 1$.

($\Leftarrow$) Supposons qu'il existe un plan de bad. Considérons un tel plan linéaire $r_1, \ldots, r_n$ de longueur minimale. Puisque ce plan est minimal, les règles $r_1, \ldots, r_{n-1}$ sont des règles de $\mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}})$, et donc le lemme 5.4 permet de conclure qu'il existe une Σ_0-recette destructrice R telle que $R\phi\downarrow = u$ et $R\psi\downarrow = v$ pour chaque $\mathsf{att}(u, v) \in S_{n-1}$ (où S_{n-1} est l'état qui résulte de l'application de $r_1, \ldots, r_{n-1}$). Alors, afin de déduire bad, une règle de $\mathsf{Concrete}^-(\mathsf{R}_{\mathsf{ana}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$ a été appliquée. Il est facile d'en déduire un témoin de $\phi \not\sqsubseteq_s \psi$. Par exemple, dans le premier cas, la définition de $\mathsf{Concrete}^-(\mathsf{R}_{\mathsf{ana}})$, implique que l'on a $r_n = \mathsf{att}(\mathsf{senc}(u_1, u_2), v), \mathsf{att}(u_2, v') \to \mathsf{bad}$ où v n'est de la forme $\mathsf{senc}(v_0, v')$ pour aucun v_0. De plus, nous avons des $\Sigma_0^\#$-recettes destructrices R_1 (et R_2) qui nous permettent d'accéder à ces faits. Dans ce cas, nous concluons grâce à la recette $\mathsf{sdec}(R_1, R_2)$. Les autres cas sont similaires. □

5.3 Attaquant actif

L'objectif de cette section est d'établir la traduction pour l'attaquant actif, c'est-à-dire pour l'équivalence de trace. Pour ce faire, nous considérons deux protocoles simples $\mathcal{P}$ et $\mathcal{Q}$. L'ensemble des constantes apparaissant dans $\mathcal{P}$ est noté $\Sigma_\mathcal{P}$ et l'ensemble des constantes apparaissant dans $\mathcal{Q}$ est noté $\Sigma_\mathcal{Q}$. Pour simplifier, nous supposons que les variables de $\mathcal{P}$ et $\mathcal{Q}$ sont disjointes (c'est-à-dire $fv(\mathcal{P}) \cap fv(\mathcal{Q}) = \emptyset$). De plus, nous notons $\Sigma_0^\star = (\Sigma_\mathcal{P} \cup \Sigma_\mathcal{Q}) \uplus \{c_0^\star; c_1^\star; c_+^\star\}$. Nous supposons également que $\mathcal{P}$ est conforme à un système de types structuré $(\mathcal{T}_0, \delta)$. Rappelons que d'après le théorème 3.1 démontré au chapitre 3, il existe un témoin d'attaque par rapport à Σ_0^- si, et seulement si, il existe un témoin quasiment typé par rapport à $\Sigma_0^\star$.

Dans un premier temps, nous présentons le calcul des règles de protocole dans la section 5.3.1. Elles viennent compléter les règles de déduction présentées dans la section précédente. Or, ces règles de déduction ne contenaient que des règles d'analyse. Il serait trop coûteux d'ajouter des règles de synthèse : la technique de l'aplatissement, exposée en section 5.3.2, remplace avantageusement

ces règles. Comme toutes les règles de protocole sont abstraites, il est nécessaire, comme dans le cas passif, de définir leur concrétisation. C'est ce que nous faisons en section 5.3.3. Le théorème de correction et de complétude de la traduction y est également énoncé, ainsi qu'une borne sur la longueur des plans. La section 5.3.4 expose le raisonnement nécessaire à la démonstration du théorème principal. Toutes les preuves se trouvent en section 5.3.5 ou dans l'annexe A.

5.3.1 Règles symboliques

Dans cette section, nous définissons les règles pour le comportement du protocole.

En supplément des faits de la forme $\mathsf{att}(u, v)$ introduits dans la section 5.2, nous considérons également :

— des faits de la forme $\mathsf{Phase}(i)$ avec $i \in \mathbb{N}$ qui représentent la phase ;

— des faits de la forme $\mathsf{St}(P, Q) = \mathsf{state}^{c}_{P,Q}(id_P, id_Q)$ où P, Q sont deux processus séquentiels utilisant un seul et même canal c (voir section 1.4), et id_P (respectivement id_Q) est la substitution identité sur le domain $fv(P)$ (respectivement $fv(Q)$).

Ainsi, dans cette section, nous considérons un ensemble infini de faits $\mathcal{F}_0$ constitué de tous les faits de cette forme, plus le symbole spécial bad. Pour modéliser la phase, nous normalisons les processus pour la règle de réécriture $i{:}j{:}P \to j{:}P$, ce qui permet d'imiter la règle de sémantique PHASE. Pour des processus normalisés, la transformation $\mathsf{Rule}(P; Q)$ des processus séquentiels vers les règles est définie récursivement par $\mathsf{Rule}(P, Q) = \emptyset$ quand $P = i{:}0$, et sinon par :

1. Émission : $P = i{:}\mathsf{out}(c, u).P'$.
   — $\{\mathsf{St}(P, Q), \mathsf{Phase}(i) \to \mathsf{att}(u, v), \mathsf{St}(P', Q'); \mathsf{St}(P, Q)\} \cup \mathsf{Rule}(i{:}P', i{:}Q')$ si $Q = i{:}\mathsf{out}(c, v).Q'$
   — $\{\mathsf{St}(P, Q), \mathsf{Phase}(i) \to \mathsf{bad}, \mathsf{att}(u, c_0^\star)\}$ sinon.
2. Réception : $P = i{:}\mathsf{in}(c, u).P'$.
   — $\{\mathsf{St}(P, Q), \mathsf{att}(u, v), \mathsf{Phase}(i) \to St(P', Q'); St(P, Q)\} \cup \mathsf{Rule}(i{:}P', i{:}Q')$ si $Q = i{:}\mathsf{in}(c, v).Q'$
   — $\{\mathsf{St}(P, Q), \mathsf{att}(u, x), \mathsf{Phase}(i) \to \mathsf{bad}\}$ sinon (avec x une variable fraîche).

Ces règles ont pour but d'imiter l'exécution du processus séquentiel P par une exécution similaire dans Q. Le déterminisme par action permet d'affirmer qu'il existe au plus une manière d'imiter P dans Q à chaque étape, puisque les actions silencieuses ont été traitées en amont. Ainsi, dans tous les cas où Q ne peut pas imiter P, une règle mène à bad. Dans le cas de l'émission, il est important d'ajouter un fait $\mathsf{att}(u, c_0^\star)$. En effet, les émissions ne sont effectuées que lorsque le terme à émettre est un message. La règle d'émission $\mathsf{St}(P, Q), \mathsf{Phase}(i) \to \mathsf{bad}, \mathsf{att}(u, c_0^\star)$ ne pourra être concrétisée que si le terme u est instancié par un message. Le calcul des règles s'étend naturellement aux protocoles simples. En effet, quitte à ajouter des processus nuls, nous pouvons supposer que $\mathcal{P} = \{P_1, \ldots, P_n\}$ et $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$ où, pour chaque i, P_i et Q_i sont des processus séquentiels utilisant un même canal c_i. Nous posons alors :

$$\mathsf{Rule}(\mathcal{P}, \mathcal{Q}) = \mathsf{Rule}(P_1, Q_1) \cup \cdots \cup \mathsf{Rule}(P_n, Q_n)$$

Pour traduire une exécution complète, nous avons également besoin de règles pour changer de phase :

$$\mathsf{R}^{\mathsf{phase}} = \{\mathsf{Phase}(i) \to \mathsf{Phase}(i + 1); \mathsf{Phase}(i) \mid i \in \mathbb{N}\}$$

Exemple 5.2. *Rappelons le processus P_A défini à l'exemple 1.12 :*

$$\begin{aligned} P_A \quad = \quad & \mathsf{out}(c_A, \langle m, a, b, \mathsf{senc}(\langle n_a, m, a, b\rangle, k_{as})\rangle). \\ & \mathsf{in}(c_A, \langle m, \mathsf{senc}(\langle n_a, x\rangle, k_{as})\rangle) \end{aligned}$$

Notons u et v les termes $u = \langle m, a, b, \mathsf{senc}(\langle n_a, m, a, b\rangle, k_{as})\rangle$, $v = \langle m, \mathsf{senc}(\langle n_a, x\rangle, k_{as})\rangle$ et P' le processus $P' = \mathsf{in}(c_A, \langle m, \mathsf{senc}(\langle n_a, x\rangle^2, k_{as})\rangle)$. Nous définissons également $\hat{x} \in \mathcal{X}$, $\hat{v} = v\{x \triangleright \hat{x}\}$, $Q' = P'\{x \triangleright \hat{x}\}$ et $Q_A = P_A\{x \triangleright \hat{x}\}$. Nous avons $\mathsf{Rule}(P_A, Q_A) = \{r_1; r_2\}$ avec :

$$r_1 = \mathsf{St}(P_A, Q_A), \mathsf{Phase}(0) \to \mathsf{att}(u, u), \mathsf{St}(P', Q'); \mathsf{St}(P_A, Q_A)$$
$$r_2 = \mathsf{St}(P', Q'), \mathsf{att}(v, \hat{v}), \mathsf{Phase}(0) \to St(0, 0); St(P', Q')$$

où $\mathsf{St}(P_A, Q_A) = \mathsf{state}^{c_A}_{P_A,Q_A}(id_{P_A}, id_{Q_A})$, $\mathsf{St}(P', Q') = \mathsf{state}^{c_A}_{P',Q'}(id_{P'}, id_{Q'})$ *et* $St(0, 0) = \mathsf{state}^{c_A}_{0,0}(\emptyset, \emptyset)$.

Nous donnons également un exemple simple de cas où l'on obtient bad :

Exemple 5.3. *Nous considérons deux processus* $\mathsf{in}(c, x)$ *et* $\mathsf{out}(c, a)$ *avec* $x \in \mathcal{X}$ *et* a *une constante, la transformation donne*

$$\mathsf{Rule}(\mathsf{in}(c, x), \mathsf{out}(c, a)) = \mathsf{state}^{c}_{\mathsf{in}(c,x),\mathsf{out}(c,a)}(\{x \triangleright x\}, \emptyset), \mathsf{att}(x, c_0^\star) \to \mathsf{bad}$$

En effet, dès que l'attaquant effectue une réception sur le canal c, il est capable de faire la différence entre $\mathsf{in}(c, x)$ *et* $\mathsf{out}(c, a)$.

5.3.2 Aplatissement

Dans la section 5.2, nous avons seulement énoncé des règles d'analyse pour l'attaquant. Nous avons démontré que ces règles sont suffisantes pour l'équivalence statique, mais elles ne suffisent pas dans le cas de l'attaquant actif : si le protocole attend un message de la forme $\langle u, v\rangle$, l'attaquant doit parfois construire ce message, par exemple à partir de u et v. Cependant, il serait impossible de permettre à l'attaquant d'effectuer des synthèses arbitraires, puisque les possibilités sont infinies. Une solution possible consiste à utiliser le typage pour autoriser seulement un nombre fini de synthèses. Cependant, la solution plus efficace que nous retenons consiste à *aplatir* les règles de protocole, c'est-à-dire que le protocole acceptera u, v aussi bien que $\langle u, v\rangle$. Il s'agira ensuite de démontrer que cette manière de faire est correcte et complète.

Définition 5.2. *Étant donné un terme $u \in \mathcal{T}_0(\Sigma_c, \Sigma_0 \uplus \mathcal{N} \uplus \mathcal{X})$, nous disons que u est* décomposable *quand :*

- *ou bien $u \in \mathcal{X}$ et $\delta(u)$ n'est pas un type initial ;*
- *ou bien $u \notin \Sigma_0 \uplus \mathcal{N} \uplus \mathcal{X}$.*

Intuitivement, une variable d'un type non initial est décomposable, puisqu'elle peut être instanciée par un terme composé, qui peut avoir été obtenu par une synthèse. Soient $\mathsf{f} \in \Sigma_c$ et $\mathsf{att}(u, v)$ avec u décomposable, tels que $\delta(u) = \mathsf{f}(\tau_1, \ldots, \tau_k)$. Par convention, nous écrivons $mgu(u, u') = \bot$ quand u et u' ne sont pas unifiables. Nous définissons $\mathsf{split}(\mathsf{att}(u, v))$ par :

$$\mathsf{split}(\mathsf{att}(u, v)) = (\mathsf{f}; \{\mathsf{att}(x_1, y_1), \ldots, \mathsf{att}(x_k, y_k)\}; \sigma_{\mathcal{P}}; \sigma_{\mathcal{Q}})$$

avec :

- $x_1, \ldots, x_k$ des variables fraîches de type $\tau_1, \ldots, \tau_k$ et $\sigma_{\mathcal{P}} = mgu(u, \mathsf{f}(x_1, \ldots, x_k))$;
- $y_1, \ldots, y_k$ sont des variables fraîches, $\sigma_{\mathcal{Q}} = mgu(v, \mathsf{f}(y_1, \ldots, y_k))$.

Remarquons que $\sigma_{\mathcal{P}} \neq \bot$ et est nécessairement une substitution quasiment typée.

Soit r une règle abstraite de la forme $\mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$ avec $f = \mathsf{att}(u, v) \in \mathsf{Pre}$ tel que u est décomposable et $\mathsf{split}(f) = (\mathsf{f}; S; \sigma_{\mathcal{P}}; \sigma_{\mathcal{Q}})$. La décomposition de r par rapport à f, notée $\mathsf{decom}(r, f)$, est définie comme suit :

1. $((\mathsf{Pre} \setminus f) \cup S \to \mathsf{bad})\sigma_{\mathcal{P}}$ dans le cas où $\sigma_{\mathcal{Q}} = \bot$;
2. $((\mathsf{Pre} \setminus f) \cup S \to \mathsf{Add}; \mathsf{Del})(\sigma_{\mathcal{P}} \uplus \sigma_{\mathcal{Q}})$ sinon.

Ensuite, la décomposition est appliquée récursivement sur chaque règle.

$$\mathsf{Flat}(r) = \{r\} \cup \mathsf{Flat}(\{\mathsf{decom}(r, f) \mid f = \mathsf{att}(u, v) \in \mathsf{Pre}(r) \text{ avec } u \text{ décomposable.}\})$$

Exemple 5.4. *Revenons aux règles r_1, r_2 de l'exemple 5.2.* $\mathsf{Flat}(r_1) = \{r_1\}$ *car r_1 est une règle d'émission. Les décomposition possibles de $v = \langle m, \mathsf{senc}(\langle n_a, x\rangle, k_{as})\rangle^2$ sont*

$$\{v; m, \mathsf{senc}(\langle n_a, x\rangle, k_{as}); m, \langle n_a, x\rangle, k_{as}; m, n_a, x, k_{as}\}$$

Donc $\mathsf{Flat}(r_2) = \{r_2, r_2^1, r_2^2, r_2^3\}$ *avec :*

$r_2^1 = \mathsf{St}(P', P'), \mathsf{att}(m, m), \mathsf{att}(\mathsf{senc}(\langle n_a, x\rangle, k_{as}), \mathsf{senc}(\langle n_a, x\rangle, k_{as})), \mathsf{Phase}(0) \to St(0, 0); St(P', P')$
$r_2^2 = \mathsf{St}(P', P'), \mathsf{att}(m, m), \mathsf{att}(\langle n_a, x\rangle, \langle n_a, x\rangle), \mathsf{att}(k_{as}, k_{as}), \mathsf{Phase}(0) \to St(0, 0); St(P', P')$
$r_2^3 = \mathsf{St}(P', P'), \mathsf{att}(m, m), \mathsf{att}(n_a, n_a), \mathsf{att}(x, x), \mathsf{att}(k_{as}, k_{as}), \mathsf{Phase}(0) \to St(0, 0); St(P', P')$

Donnons également un exemple dans lequel l'aplatissement donne bad.

Exemple 5.5. *Considérons la règle*

$$r = \mathsf{state}^c_{P,Q}(id_P, id_Q), \mathsf{att}(x, a) \to \mathsf{state}^c_{P',Q'}(id_{P'}, id_{Q'})$$

où P, Q, P', Q' sont des processus séquentiels, x est une variable, a une constante et c un canal public. Soit $(\mathcal{T}_0, \delta_0)$ un système de type structuré tel que $\delta_0(a) = \tau_a \in \mathcal{T}_0$ et $\delta_0(x) = \mathsf{senc}(\tau_a, \tau_a)$. Nous avons $\mathsf{Flat}(r) = \{r; r'\}$ *avec*

$$r' = \mathsf{state}^c_{P,Q}(id_P, id_Q), \mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2) \to \mathsf{bad}$$

où x_1, x_2, y_1 et y_2 sont des variables, et $\delta(x_1) = \delta(x_2) = \tau_a$ (il n'y a pas de contraintes sur les types des variables de droite). En effet, quand l'attaquant sait assigner des valeurs de type τ_a aux variables x_1, x_2, il est capable de construire un message de type $\mathsf{senc}(\tau_a, \tau_a)$. Le message qui correspond à droite est nécessairement de la forme $\mathsf{senc}(\_, \_)$, donc il ne s'unifie pas avec a.

5.3.3 Concrétisation

Étant donnée une règle r, nous notons $vars_{\mathsf{left}}(r)$ les variables qui apparaissent à gauche d'un prédicat de r, c'est-à-dire :

— $vars_{\mathsf{left}}(\mathsf{att}(u, v)) = vars(u)$
— $vars_{\mathsf{left}}(\mathsf{state}^c_{P,Q}(\sigma_P, \sigma_Q)) = vars(img(\sigma_P))$

Étant donnée une substitution σ telle que le terme $x\sigma$ est clos pour toute variable $x \in vars(r)$, l'application de σ sur un état est l'état obtenu en composant les substitutions, c'est-à-dire

$$\mathsf{state}^c_{P,Q}(\sigma_P, \sigma_Q)\sigma = \mathsf{state}^c_{P,Q}(\sigma \circ \sigma_P, \sigma \circ \sigma_Q)$$

Les concrétisations positives d'une règle de protocole r donnée consistent simplement en toutes les instanciations qui sont quasiment typées par rapport au côté gauche, c'est-à-dire :

$$\mathsf{Concrete}^+(r) = \{r\sigma \mid \sigma \text{ substitution telle que } r\sigma \text{ ne contient que des } \Sigma_0^\star\text{-messages} \\ \text{et } \delta(x\sigma) \preccurlyeq \delta(x) \text{ pour tout } x \in vars_{\mathsf{left}}(r)\}$$

De même que pour le cas de l'attaquant passif, il faut aussi disposer de règles pour les cas où l'inclusion statique n'est pas vérifiée. Nous considérons donc la concrétisation négative des règles du protocole, qui correspond aux situations où une étape peut être exécutée par le protocole $\mathcal{P}$ mais pas par le protocole $\mathcal{Q}$.

Plus formellement, étant donnée une règle $r = \mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$, l'ensemble $\mathsf{Concrete}^-(r)$ contient les règles de la forme $f_1, \ldots, f_k \to \mathsf{bad}$ pour chaque séquence de faits $f_1, \ldots, f_k$ qui s'unifient à gauche avec Pre (avec la substitution σ_L) et où u est un $\Sigma_0^\star$-message pour chaque $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_L$, si elle vérifie au moins l'une des conditions suivantes :

— $f_1, \ldots, f_k$ ne s'unifie pas à droite avec Pre

— $f_1, \ldots, f_k$ s'unifie à droite avec Pre par la substitution σ_R, mais v n'est pas un $\Sigma_0^\star$-message pour un certain $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_R$.

Exemple 5.6. *Considérons à nouveau la règle r de l'exemple 5.5 :*

$$r = \mathsf{state}^c_{P,Q}(id_P, id_Q), \mathsf{att}(x, a) \to \mathsf{state}^c_{P',Q'}(id_{P'}, id_{Q'})$$

Supposons que l'ensemble de faits considérés soit $F = \{\mathsf{state}^c_{P,Q}(\sigma_P, \sigma_Q), \mathsf{att}(b, b)\}$ avec b une constante et σ_P, σ_Q des substitutions. Soit un système de type structuré $(\mathcal{T}_1, \delta_1)$ tel que $\delta_1(x) = \delta_1(a) = \delta_1(b) = \tau_a \in \mathcal{T}_1$, et r'' la règle suivante :

$$r'' = \mathsf{state}^c_{P,Q}(\sigma_P, \sigma_Q), \mathsf{att}(b, b) \to \mathsf{bad}$$

Nous avons $r'' \in \mathsf{Concrete}^-(r)$ car $\mathsf{att}(b, b)$ s'unifie à gauche, mais pas à droite, avec $\mathsf{att}(x, a)$. De plus, r'' est applicable dans F.

Avant de continuer, il est nécessaire de préciser la définition de l'ensemble $\mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}}$. En effet, nous avons défini $\mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}}$ comme l'ensemble des règles $\mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k), \mathsf{att}(C[u_1, \ldots, u_k], v) \to \mathsf{bad}$ avec C un contexte constructeur non vide tel que $C[u_1, \ldots, u_k] \in St_{\mathsf{opti}}(\phi) \cup \Sigma_0^\#$ et $v \neq C[v_1, \ldots, v_k]$. En particulier, les contextes considérés dépendent de ϕ. Cependant, comme la trace est quasiment typée par rapport à $\mathcal{P}$, nous pouvons choisir plutôt les contextes tels que $C[\tau_1, \ldots, \tau_n] \in St_{\mathsf{opti}}(\delta_{\mathcal{P}}(\mathcal{P}))$ où $\tau_1, \ldots, \tau_n$ sont des types quelconques. Ce sont ces contextes que nous considérons désormais. Énonçons le théorème principal de ce chapitre, qui affirme que la traduction que nous avons définie est correcte et complète. De plus, ce théorème donne une borne relativement fine sur la longueur du plan. Il est facile de concevoir des exemples pour lesquels elle est atteinte.

Théorème 5.1. *Soit $\mathcal{P}$ un protocole simple conforme à un système de types structuré $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$, et $\mathcal{Q}$ un autre protocole simple. Nous considérons l'ensemble R de règles :*

$$\mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))) \cup \mathsf{R}^{\mathsf{phase}} \cup \mathsf{R}^{\mathsf{test}_1}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{atom}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{check}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{pub}}_{\mathsf{fail}}$$

Soit $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q}), \mathsf{R} \rangle$ et $\Pi = \langle \Theta, \{\mathsf{bad}\} \rangle$. Nous avons $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ si, et seulement si, Π a une solution de longueur

$$1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + \mathsf{nb}_{\mathsf{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_{\mathcal{P}}(\mathcal{P})) \times [1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + N]$$

où N est le nombre de noms apparaissant dans $\mathcal{P}$ qui ont le type d'une clé, c'est-à-dire tels que $\delta_{\mathcal{P}}(n)$ apparaît en position de clé dans $\delta_{\mathcal{P}}(\mathcal{P})$.

5.3.4 Correction et complétude

Le rôle de cette section est de démontrer la correction de la traduction du problème d'équivalence sous forme de planification, c'est-à-dire le théorème 5.1.

Aplatissement. Une première étape consiste à expliciter le lien entre une règle r et ses aplatissements dans $\mathsf{Flat}(r)$. Les preuves formelles, qui sont particulièrement techniques, ainsi que des lemmes intermédiaires, se trouvent dans l'annexe A. Le lemme suivant énonce que les instanciations d'une règle induisent des instanciations équivalentes sur ses aplatissements.

Lemme 5.9. *Soit $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ une règle. Soit σ une substitution telle que $r\sigma$ est close et $\delta(x\sigma) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r)$. Soit C un contexte constructeur tel que $u\sigma = C[u_1, \ldots, u_n]$ et $v\sigma = C[v_1, \ldots, v_n]$. Il existe $r' \in \mathsf{Flat}(r)$:*

$$r' = \mathsf{Pre}', \mathsf{att}(u'_1, v'_1), \ldots, \mathsf{att}(u'_n, v'_n) \to \mathsf{Add}'; \mathsf{Del}'$$

et σ' une substitution telle que :

1. *$r'\sigma'$ est une règle close.*
2. *$\delta(x\sigma') \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r')$;*
3. *$(\mathsf{Pre}', \mathsf{Add}', \mathsf{Del}')\sigma' = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})\sigma$;*
4. *$\mathsf{att}(u, v)\sigma = \mathsf{att}(C[u'_1, \ldots, u'_n], C[v'_1, \ldots, v'_n])\sigma'$.*

La démonstration de ce lemme repose sur l'idée simple que chaque décomposition donne un élément du contexte C.

Pour la suite, il est nécessaire de distinguer l'origine de chaque bad. Ainsi, nous distinguerons entre bad-proto, lorsque le fait bad provient directement d'une règle de protocole, bad-flat lorsqu'il provient de l'aplatissement, et bad-concrete quand il découle de la concrétisation. Nous continuerons d'écrire bad pour désigner un cas quelconque parmi ces trois possibilités chaque fois que la distinction ne sera pas importante.

Le lemme qui suit est essentiellement la réciproque du lemme 5.9. Il établit que l'on peut retrouver la règle d'origine à partir d'un règle aplatie, d'un contexte et d'une substitution.

Lemme 5.10. *Soit r une règle de protocole, et $r' \in \mathsf{Flat}(r)$ telle que*

$$r' = \mathsf{state}, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_n, v_n) \to \mathsf{Add}; \mathsf{Del}$$

— *Ou bien $\mathsf{bad\text{-}flat} \notin \mathsf{Add}$, et alors il existe un contexte constructeur C et une substitution τ tels que*

$$r\tau = \mathsf{Pre}, \mathsf{att}(C[u_1, \ldots, u_n], C[v_1, \ldots, v_n]) \to \mathsf{Add}; \mathsf{Del}$$

— *Ou bien $\mathsf{Add} = \mathsf{bad\text{-}flat}$, $\mathsf{Del} = \emptyset$ et il existe une substitution τ, un terme v et deux ensembles Add_0 et Del_0 tels que $r\tau = \mathsf{state}, \mathsf{att}(C[u_1, \ldots, u_n], v) \to \mathsf{Add}_0; \mathsf{Del}_0$ mais v ne s'unifie pas avec C.*

Chaque règle de $\mathsf{Flat}(r)$ provient d'une décomposition qui a aplati un symbole de fonction. Pour reconstruire le contexte, il suffit de remonter la chaîne des décompositions qui ont mené de la règle de départ à la règle aplatie.

Le dernier lemme permet de montrer qu'il existe une règle qui donne bad quand l'aplatissement est possible côté P mais ne correspond pas à la forme du motif attendu côté Q. Pour l'énoncer, il est nécessaire d'introduire la relation $=^{\mathsf{left}}$ sur les faits. Formellement, pour deux faits f_1 et f_2, nous écrivons $f_1 = f_2$ si l'une des deux conditions suivantes est satisfaite :

— $f_1 = \mathsf{att}(u, v)$, $f_2 = \mathsf{att}(u', v')$ et $u = u'$.

— $f_1 = \mathsf{state}^{c}_{P,Q}(\sigma_P, \sigma_Q)$, $f_2 = \mathsf{state}^{c'}_{P',Q'}(\sigma'_P, \sigma'_Q)$ et $c = c'$, $P = P'$ et $\sigma_P = \sigma'_P$.

Cette définition s'étend naturellement à Pre.

Lemme 5.11. *Soient $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ une règle, σ une substitution telle que pour tout $x \in vars_{\mathsf{left}}(r)$, $x\sigma$ est clos et $\delta(x\sigma) \preccurlyeq \delta(x)$, et C un contexte linéaire construit sur Σ_c. Supposons que $u\sigma = C[u_1, \ldots, u_n]$ et v et C ne sont pas unifiables.*

Alors il existe $r' \in \mathsf{Flat}(r)$ tel que $r' = \mathsf{Pre}', \mathsf{att}(u'_1, v_1), \ldots, \mathsf{att}(u'_n, v_n) \to \mathsf{bad}$ et σ' tel que $u_i = u'_i\sigma'$ et $\mathsf{Pre}'\sigma' =^{\mathsf{left}} \mathsf{Pre}\sigma$.

Traduction complète. Nous démontrons maintenant la correction et la complétude de la traduction complète. Les preuves formelles se trouvent en section 5.3.5. Soit $\mathcal{K}_\mathcal{P} = (\mathcal{P}; \phi; \sigma_\mathcal{P}; i_P)$ et $\mathcal{K}_\mathcal{Q} = (\mathcal{Q}; \psi; \sigma_\mathcal{Q}; i_Q)$ deux configurations avec $i_P = i_Q$ et $dom(\phi) = dom(\psi)$. L'ensemble de faits associés à $\mathcal{K}_\mathcal{P}$ et $\mathcal{K}_\mathcal{Q}$ relativement à un ensemble de constantes Σ_0 est défini par :

$$\begin{aligned}\mathsf{Fact}_{\Sigma_0}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q}) = \{\mathsf{Phase}(i)\} \cup \mathsf{Fact}_{\Sigma_0}(\phi, \psi) \cup \\ \{\mathsf{state}^{c}_{P,Q}(\sigma_P, \sigma_Q) \mid P \in \mathcal{P}, Q \in \mathcal{Q} \text{ sont séquentiels sur le canal } c, \\ \sigma_P = \sigma_\mathcal{P}|_{fv(P)} \text{ et } \sigma_Q = \sigma_\mathcal{Q}|_{fv(Q)}\}\end{aligned}$$

Définition 5.3. *Soient deux ensembles S et S' de faits tels que $S = \mathsf{Fact}_{\Sigma_0}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q})$ avec $\mathcal{K}_\mathcal{P} = (\mathcal{P}; \phi; \sigma_\mathcal{P}; i)$ et $\mathcal{K}_\mathcal{Q} = (\mathcal{Q}; \psi; \sigma_\mathcal{Q}; i)$ avec $dom(\phi) = dom(\psi)$, nous écrivons $\mathsf{Fact}_{\Sigma_0}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q}) \uparrow S'$ si :*

— $\mathsf{Fact}_{\Sigma_0}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q})$ *et S' coïncident sur les faits* state *et* Phase *;*

— *pour chaque* $\mathsf{att}(u, v) \in \mathsf{Fact}_{\Sigma_0}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q})$*, nous avons* $\mathsf{att}(u, v) \in S'$ *;*

— *pour chaque* $\mathsf{att}(u, v) \in S'$*, il existe une Σ_0-recette destructrice R telle que $R\phi\downarrow = u$ et $R\psi\downarrow = u$.*

Ci-dessous, $\mathsf{nb}_{\mathsf{in}}(\mathsf{tr})$ (resp. $\mathsf{nb}_{\mathsf{out}}(\mathsf{tr})$) est le nombre de réceptions (resp. émissions) qui apparaissent dans tr tandis que $\mathsf{maxphase}(\mathsf{tr})$ est l'entier maximal qui apparaît dans une instruction de phase dans tr. Le résultat suivant permet d'établir un lien entre une trace qui passe dans $\mathcal{P}$ et $\mathcal{Q}$ et un chemin dans le système de planification. La borne est énoncée sur la trace et la trame plutôt que sur le protocole de départ, pour conserver le résultat le plus précis possible. Le calcul en fonction du protocole sera effectué ultérieurement. $\mathsf{Key}(\mathsf{tr})$ est l'ensemble des recettes de clés dans tr, c'est-à-dire :

$$\mathsf{Key}(\mathsf{tr}) = \{R \mid \exists \mathsf{in}(c, R') \in \mathsf{tr}, R \text{ est en position de clé dans } R' \text{ et } R\phi\downarrow \in \mathcal{N}\}$$

Lemme 5.12. *Soit $\mathcal{P}$ un protocole simple conforme à un système de type structuré $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$, et $\mathcal{Q}$ un autre protocole simple. Soit $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q}), \mathsf{R} \rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}^{+}(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))) \cup \mathsf{R}^{\mathsf{phase}}$$

Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ pour un certain ϕ tel que :

— tr *ne contient que des recettes fortement simples.*

— (tr, ϕ) *est quasiment typée par rapport à $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$.*

— $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$ *pour un certain ψ.*

Alors, il existe un chemin depuis $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P},\mathcal{Q})$ *vers un certain* S *tel que* $\mathsf{Fact}(\mathcal{K}'_\mathcal{P},\mathcal{K}'_\mathcal{Q}) \uparrow S$ *de longueur au plus :*

$$\mathsf{nb_{in}}(\mathsf{tr}) + \mathsf{nb_{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb_{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr})|) \times \mathsf{depth}(\phi)$$

où $\mathcal{K}'_\mathcal{P}$ *(resp.* $\mathcal{K}'_\mathcal{Q}$*) est la configuration obtenue après l'exécution de la trace* tr *dans* $\mathcal{P}$ *(resp.* $\mathcal{Q}$*), et* $\{\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \mid R \in \mathsf{Key}(\mathsf{tr})\} \subseteq S$.

Réciproquement, soit π *un chemin depuis* $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P},\mathcal{Q})$ *vers* S *tel que* $\mathsf{bad} \notin S$*. Alors il existe une trace* tr*, et des* $\Sigma_0^\star$*-trames* ϕ *et* ψ*, telles que :*

- tr *ne contient que des recettes fortement simples ;*
- $(\mathsf{tr},\phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ *est quasiment typée pour* $(\mathcal{T}_\mathcal{P},\delta_\mathcal{P})$ *;*
- $(\mathsf{tr},\psi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$ *;*
- $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'_\mathcal{P},\mathcal{K}'_\mathcal{Q}) \uparrow S$ *où* $\mathcal{K}'_\mathcal{P}$ *(resp.* $\mathcal{K}'_\mathcal{Q}$*) est la configuration obtenue après l'exécution de la trace* tr *dans* $\mathcal{P}$ *(resp.* $\mathcal{Q}$*).*

La preuve est assez naturelle et demande seulement de traiter chaque cas avec la règle de planification adaptée. La borne est calculée comme suit : $\mathsf{nb_{in}}(\mathsf{tr}) + \mathsf{nb_{out}}(\mathsf{tr})$ représentent le nombre d'étapes du protocole. $\mathsf{maxphase}(\mathsf{tr})$ est le nombre d'étapes utilisées pour atteindre la phase maximale. Ensuite chaque déduction de clé prend $\mathsf{depth}(\phi)$ étapes, ce qui fait un total de $|\mathsf{Key}(\mathsf{tr})| \times \mathsf{depth}(\phi)$ pour déduire les clés, puis $\mathsf{nb_{in}}(\mathsf{tr}) \times \mathsf{depth}(\phi)$ étapes pour déduire à chaque réception le message de l'attaquant. Ce résultat permet de démontrer le théorème principal.

Théorème 5.1. *Soit* $\mathcal{P}$ *un protocole simple conforme à un système de types structuré* $(\mathcal{T}_\mathcal{P},\delta_\mathcal{P})$*, et* $\mathcal{Q}$ *un autre protocole simple. Nous considérons l'ensemble* R *de règles :*

$$\mathsf{Concrete}(\mathsf{R_{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P},\mathcal{Q}))) \cup \mathsf{R}^{\mathsf{phase}} \cup \mathsf{R}^{\mathsf{test}_1}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{atom}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{check}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{pub}}_{\mathsf{fail}}$$

Soit $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0{}^\star}(\mathcal{P},\mathcal{Q}), \mathsf{R}\rangle$ *et* $\Pi = \langle \Theta, \{\mathsf{bad}\}\rangle$*. Nous avons* $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ *si, et seulement si,* Π *a une solution de longueur*

$$1 + \mathsf{nb_{in}}(\mathcal{P}) + \mathsf{nb_{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_\mathcal{P}(\mathcal{P})) \times [1 + \mathsf{nb_{in}}(\mathcal{P}) + N]$$

où N *est le nombre de noms apparaissant dans* $\mathcal{P}$ *qui ont le type d'une clé, c'est-à-dire tels que* $\delta_\mathcal{P}(n)$ *apparaît en position de clé dans* $\delta_\mathcal{P}(\mathcal{P})$*.*

La démonstration de ce théorème découle de l'application du lemme 5.12 suivie d'une disjonction de cas, selon l'origine de la non-inclusion : si le protocole $\mathcal{Q}$ ne peut pas suivre $\mathcal{P}$, la règle de planification utilisée est une règle de protocole ; si c'est un problème d'équivalence statique, il faut encore traiter chaque cas (atome, message, clé publique, vérification de signature, égalité) séparément. Pour obtenir la borne, il suffit de reformuler celle du lemme 5.12 pour l'exprimer directement sur $\mathcal{P}$ après avoir ajouté $\mathsf{depth}(\phi) + 1$ étapes pour la déduction et le test d'équivalence statique. $\mathsf{depth}(\phi)$ est inférieure à $\mathsf{depth}(\delta_\mathcal{P}(\mathcal{P}))$, la profondeur des types de $\mathcal{P}$, car le système de types $(\mathcal{T}_\mathcal{P},\delta_\mathcal{P})$ est structuré. $\mathsf{nb_{in}}(\mathsf{tr})$ et $\mathsf{nb_{out}}(\mathsf{tr})$ sont bornés directement par $\mathsf{nb_{in}}(\mathcal{P})$ et $\mathsf{nb_{out}}(\mathcal{P})$. Enfin, l'ensemble $\mathsf{Key}(\mathsf{tr})$ ne contient que des recettes de clés inconnues de l'attaquant. Nous pouvons supposer que dans la trace tr une clé donnée n'est déduite que par une seule recette, nous avons $|\mathsf{Key}(\mathsf{tr})| \leqslant N$.

5.3.5 Démonstrations

Commençons par montrer le lemme 5.12.

Lemme 5.12. *Soit $\mathcal{P}$ un protocole simple conforme à un système de type structuré $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$, et $\mathcal{Q}$ un autre protocole simple. Soit $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q}), \mathsf{R} \rangle$ où*

$$\mathsf{R} = \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))) \cup \mathsf{R}^{\mathsf{phase}}$$

Soit $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ pour un certain ϕ tel que :

- *tr ne contient que des recettes fortement simples.*
- *(tr, ϕ) est quasiment typée par rapport à $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$.*
- *$(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$ pour un certain ψ.*

Alors, il existe un chemin depuis $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q})$ vers un certain S tel que $\mathsf{Fact}(\mathcal{K}'_\mathcal{P}, \mathcal{K}'_\mathcal{Q}) \uparrow S$ de longueur au plus :

$$\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr})|) \times \mathsf{depth}(\phi)$$

où $\mathcal{K}'_\mathcal{P}$ (resp. $\mathcal{K}'_\mathcal{Q}$) est la configuration obtenue après l'exécution de la trace tr dans $\mathcal{P}$ (resp. $\mathcal{Q}$), et $\{\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \mid R \in \mathsf{Key}(\mathsf{tr})\} \subseteq S$.

Réciproquement, soit π un chemin depuis $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q})$ vers S tel que $\mathsf{bad} \notin S$. Alors il existe une trace tr, et des $\Sigma_0^\star$-trames ϕ et ψ, telles que :

- *tr ne contient que des recettes fortement simples ;*
- *$(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ est quasiment typée pour $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$;*
- *$(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$;*
- *$\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'_\mathcal{P}, \mathcal{K}'_\mathcal{Q}) \uparrow S$ où $\mathcal{K}'_\mathcal{P}$ (resp. $\mathcal{K}'_\mathcal{Q}$) est la configuration obtenue après l'exécution de la trace tr dans $\mathcal{P}$ (resp. $\mathcal{Q}$).*

Démonstration. Montrons les deux implications séparément.
($\Rightarrow$) La preuve se fait par récurrence sur l'exécution $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}} \mathcal{K}'_\mathcal{P}$.

Initialisation. Le chemin vide suffit à établir le résultat.

Hérédité. Nous avons $\mathsf{tr} = \mathsf{tr}'.\alpha$. Donc, considérons $\mathcal{K}''_\mathcal{P}$ et $\mathcal{K}''_\mathcal{Q}$ tels que $\mathcal{K}_\mathcal{P} \xrightarrow{\mathsf{tr}'} \mathcal{K}''_\mathcal{P} \xrightarrow{\alpha} \mathcal{K}'_\mathcal{P}$, et $\mathcal{K}_\mathcal{Q} \xrightarrow{\mathsf{tr}'} \mathcal{K}''_\mathcal{Q} \xrightarrow{\alpha} \mathcal{K}'_\mathcal{Q}$. Soit $\mathcal{K}''_\mathcal{P} = (\mathcal{P}''; \phi''; \sigma''_\mathcal{P}; i'')$, $\mathcal{K}'_\mathcal{P} = (\mathcal{P}'; \phi'; \sigma'_\mathcal{P}; i')$, $\mathcal{K}''_\mathcal{Q} = (\mathcal{Q}''; \psi''; \sigma''_\mathcal{Q}; i'')$, et $\mathcal{K}'_\mathcal{Q} = (\mathcal{Q}'; \psi'; \sigma'_\mathcal{Q}; i')$.

Appliquons l'hypothèse de récurrence à tr' : nous obtenons un chemin π_0 de longueur au plus

$$\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}') + \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}') + \mathsf{maxphase}(\mathsf{tr}') + (\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}') + |\mathsf{Key}(\mathsf{tr}')|) \times \mathsf{depth}(\phi'')$$

à partir de $\mathsf{Fact}_{\Sigma_0{}^\star}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q})$ vers un certain S''. De plus, $\mathsf{Fact}_{\Sigma_0{}^\star}(\mathcal{K}''_\mathcal{P}, \mathcal{K}''_\mathcal{Q}) \uparrow S''$, et $\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \in S''$ pour chaque $R \in \mathsf{Key}(\mathsf{tr}')$. Nous distinguons plusieurs cas suivant la valeur de α.

Cas $\alpha = \mathsf{phase}\ i'$. Nous avons $i' > i''$, $\mathcal{P}' = \mathcal{P}''$, $\phi' = \phi''$ et $\sigma'_\mathcal{P} = \sigma''_\mathcal{P}$. De même, $\mathcal{Q}' = \mathcal{Q}''$, $\psi' = \psi''$ et $\sigma'_\mathcal{Q} = \sigma''_\mathcal{Q}$. Comme $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}''_\mathcal{P}, \mathcal{K}''_\mathcal{Q}) \uparrow S''$, $\mathsf{Phase}(i'') \in S''$. Soit $r_i = \mathsf{Phase}(i) \to \mathsf{Phase}(i+1); \mathsf{Phase}(i)$ avec $i \in \mathbb{N}$. Le chemin $\pi = \pi_0.r_{i''}.\ldots.r_{i'-1}$ permet de passer de $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}_\mathcal{P}, \mathcal{K}_\mathcal{Q})$ à un certain S', avec $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'_\mathcal{P}, \mathcal{K}'_\mathcal{Q}) \uparrow S'$ et $\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \in S'$ pour chaque $R \in \mathsf{Key}(\mathsf{tr})$. Puisque $\phi' = \phi''$, et $\mathsf{tr} = \mathsf{tr}'.\mathsf{phase}\ i'$, la longueur de π est au plus

$$\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr})|) \times \mathsf{depth}(\phi').$$

Cas $\alpha = \mathsf{out}(c, \mathsf{w})$. Il existe P'_c, et Q'_c, ainsi que u et v tels que

— $\mathcal{P}'' = \{i'':\mathsf{out}(c,u).P'_c\} \uplus \mathcal{P}_0$, et $\mathcal{P}' = \{i'':P'_c\} \uplus \mathcal{P}_0$;

— $\mathcal{Q}'' = \{i'':\mathsf{out}(c,v).Q'_c\} \uplus \mathcal{Q}_0$, et $\mathcal{Q}' = \{i'':Q'_c\} \uplus \mathcal{Q}_0$.

De plus, $\phi' = \phi'' \uplus \{\mathsf{w} \triangleright u\sigma''_{\mathcal{P}}\}$, et $\psi' = \psi'' \uplus \{\mathsf{w} \triangleright v\sigma''_{\mathcal{Q}}\}$; et $\sigma'_{\mathcal{Q}} = \sigma''_{\mathcal{P}}$, ainsi que $\sigma'_{\mathcal{Q}} = \sigma''_{\mathcal{Q}}$, et $i' = i''$. De plus, $u\sigma''_{\mathcal{P}}$ et $v\sigma''_{\mathcal{Q}}$ sont des $\Sigma_0^\star$-messages. Soit r la règle de protocole correspondant à cette étape. Nous avons $r \in \mathsf{Rule}(\mathcal{P},\mathcal{Q})$ et cette règle est de la forme :

$$\mathsf{Phase}(i''), \mathsf{St}(P,Q) \to \mathsf{att}(u,v), \mathsf{St}(P',Q'); \mathsf{St}(P,Q)$$

Maintenant, considérons l'instance concrète qui correspond à l'exécution, c'est-à-dire obtenue après application de $\sigma'_{\mathcal{P}} \uplus \sigma'_{\mathcal{Q}}$. Elle permet de conclure en une seule étape, et puisque $\mathsf{nb}_{\mathsf{out}}(\mathsf{tr}) = \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}')+1$, $\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) = \mathsf{nb}_{\mathsf{in}}(\mathsf{tr}')$, $\mathsf{Key}(\mathsf{tr}) = \mathsf{Key}(\mathsf{tr}')$, et $\mathsf{depth}(\phi') \geqslant \mathsf{depth}(\phi'')$, nous obtenons un plan de longueur au plus

$$\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr})|) \times \mathsf{depth}(\phi).$$

Cas $\alpha = \mathsf{in}(c,R)$. Nous savons que R est une $\Sigma_0^\star$-recette fortement simple ($R = C[R_1,\ldots,R_k]$ avec R_i une recette destructrice pour chaque i), et qu'il existe P'_c, et Q'_c, ainsi que u et v tels que :

— $\mathcal{P}'' = \{i'':\mathsf{in}(c,u).P'_c\} \uplus \mathcal{P}_0$, et $\mathcal{P}' = \{i'':P'_c\} \uplus \mathcal{P}_0$;

— $\mathcal{Q}'' = \{i'':\mathsf{in}(c,v).Q'_c\} \uplus \mathcal{Q}_0$, et $\mathcal{Q}' = \{i'':Q'_c\} \uplus \mathcal{Q}_0$.

De plus, $\phi' = \phi''$, $\psi' = \psi''$; $\sigma'_{\mathcal{P}} = \sigma''_{\mathcal{P}} \uplus mgu(R\phi''\!\downarrow, u\sigma''_{\mathcal{P}})$, et $\sigma'_{\mathcal{Q}} = \sigma''_{\mathcal{Q}} \uplus mgu(R\psi''\!\downarrow, v\sigma''_{\mathcal{Q}})$, et $i' = i''$. De plus, $u\sigma'_{\mathcal{P}}$ et $v\sigma'_{\mathcal{Q}}$ sont des $\Sigma_0^\star$-messages. En appliquant le lemme 5.7 pour $\mathcal{R} = \mathsf{Key}(\mathsf{tr})$ et $\mathcal{R}_0 = \mathsf{Key}(\mathsf{tr}')$, nous obtenons un plan π_1 vers S'' tel que $\{\mathsf{att}(R'\phi\!\downarrow, R'\psi\!\downarrow) \mid R' \in \mathsf{Key}(\mathsf{tr}')\} \subseteq S''$, et la longueur de π_1 est au plus $|\mathsf{Key}(\{R_1,\ldots,R_k\}) \setminus \mathsf{Key}(\mathsf{tr}')| \times \mathsf{depth}(\phi'')$.

Soit r la règle de protocole correspondant à cette étape. Nous avons $r \in \mathsf{Rule}(\mathcal{P},\mathcal{Q})$ et cette règle est de la forme :

$$\mathsf{Phase}(i''), \mathsf{St}(P,Q), \mathsf{att}(u,v) \to \mathsf{St}(P',Q'); \mathsf{St}(P,Q).$$

Nous avons $R\phi'\!\downarrow = u\sigma'_{\mathcal{P}}$ et $R\psi'\!\downarrow = v\sigma'_{\mathcal{Q}}$ avec $R = C[R_1,\ldots,R_k]$. Donc $u\sigma'_{\mathcal{P}} = C[R_1\phi'\!\downarrow,\ldots,R_k\phi'\!\downarrow]$ et $v\sigma'_{\mathcal{Q}} = C[R_1\psi'\!\downarrow,\ldots,R_k\psi'\!\downarrow]$. D'après le lemme 5.9, il existe une règle $r' \in \mathsf{Flat}(r)$:

$$r' = \mathsf{Pre}', \mathsf{att}(u'_1,v'_1),\ldots,\mathsf{att}(u'_k,v'_k) \to \mathsf{Add}'; \mathsf{Del}'$$

et σ' une substitution telle que :

1. pour tout $x \in vars(r)$, $x\sigma$ est clos.
2. $\delta_{\mathcal{P}}(x\sigma') \preccurlyeq \delta_{\mathcal{P}}(x)$ pour chaque $x \in vars_{\mathsf{left}}(r')$;
3. $(\mathsf{Pre}', \mathsf{Add}', \mathsf{Del}')\sigma' = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})(\sigma'_{\mathcal{P}} \uplus \sigma'_{\mathcal{Q}})$;
4. $\mathsf{att}(u\sigma'_{\mathcal{P}}, v\sigma'_{\mathcal{Q}}) = \mathsf{att}(C[u'_1,\ldots,u'_k], C[v'_1,\ldots,v'_k])\sigma'$.

De plus, $\mathsf{att}(R_i\phi'\!\downarrow, R_i\psi'\!\downarrow) = \mathsf{att}(u'_i\sigma', v'_i\sigma')$ pour chaque $1 \leqslant i \leqslant k$. D'après le lemme 5.6 il existe un plan π_2 (applicable après π_1) de longueur au plus $\mathsf{depth}(\phi')$ et qui ne contient que des règles de l'attaquant, tel que $\mathsf{att}(R_i\phi'\!\downarrow, R_i\psi'\!\downarrow)$ appartient à l'état final après π_2. Nous considérons $r'\sigma' \in \mathsf{Concrete}^+(r')$, et nous obtenons le résultat attendu en considérant le plan $\pi_0.\pi_1.\pi_2.r''$ de longueur au plus

$$\begin{array}{rl} & \mathsf{nb}_{\mathsf{in}}(\mathsf{tr}') + \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}') + \mathsf{maxphase}(\mathsf{tr}') + (\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}') + |\mathsf{Key}(\mathsf{tr}')|) \times \mathsf{depth}(\phi'') \\ & +|\mathsf{Key}(\{R_1,\ldots,R_k\}) \setminus \mathsf{Key}(\mathsf{tr}')| \times \mathsf{depth}(\phi'') + \mathsf{depth}(\phi') + 1 \\ \leqslant & \mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + \mathsf{nb}_{\mathsf{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb}_{\mathsf{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr})|) \times \mathsf{depth}(\phi') \end{array}$$

($\Leftarrow$) Nous montrons ce résultat par récurrence sur la longueur du chemin.
Initialisation. Évident.

Hérédité. Considérons un chemin $r_1, \ldots, r_n$. Par hypothèse de récurrence, le résultat est vrai pour $r_1, \ldots, r_{n-1}$ ce qui mène à l'état S_{n-1}, et donc à l'existence d'une trace tr' telle que :

- tr' ne contient que des recettes fortement simples ;
- $(\mathsf{tr}', \phi'') \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ et est quasiment typée par rapport à $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$;
- $(\mathsf{tr}', \psi'') \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$ pour une certaine $\Sigma_0^\star$-trame ψ'' ;
- $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'''_\mathcal{P}, \mathcal{K}''_\mathcal{Q}) \uparrow S$ où $\mathcal{K}''_\mathcal{P}$ (resp. $\mathcal{K}''_\mathcal{Q}$) est la configuration obtenue après l'exécution de tr' dans $\mathcal{P}$ (resp. $\mathcal{Q}$).

Nous distinguons différents cas suivant la règle r_n. Si r_n est une instance de $\mathsf{Concrete}^+(\mathsf{R}_{\mathsf{Ana}})$, nous considérons à nouveau $\mathsf{tr} = \mathsf{tr}'$. Le cas où r_n est une règle avec $\mathsf{bad} \in \mathsf{Add}(r_n)$ est impossible puisque $\mathsf{bad} \notin S_n$. Le cas où r_n est $\mathsf{Phase}(i) \to \mathsf{Phase}(i+1); \mathsf{Phase}(i)$ peut être imité dans l'exécution par l'instruction $\mathsf{phase}\ i+1$. Maintenant, il reste à traiter le cas où r_n est une instance d'une règle de $\mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))$. Soit r_f la règle aplatie, et r la règle de protocole.

Si la règle r correspond au cas d'une émission sur le canal c, alors r_n est une instance de r puisque l'aplatissement ne produit pas de nouvelle règle. Dans ce cas, nous pouvons imiter cette étape en considérant $\mathsf{tr}'.\mathsf{out}(c, \mathsf{w})$.

Si r est une règle correspondant à une émission, alors r_n est une instance d'une règle $r_f \in \mathsf{Flat}(r)$. La règle r_f est de la forme :

$$\mathsf{Phase}(i), \mathsf{St}^c_{P,Q}(\theta_P, \theta_Q), \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k) \to \mathsf{St}^c_{P',Q'}(\theta_{P'}, \theta_{Q'})$$

et r_n est une instance de $r_f(\sigma''_\mathcal{P} \cup \sigma''_\mathcal{Q})$ où $\sigma''_\mathcal{P}$ (resp. $\sigma''_\mathcal{Q}$) est la substitution obtenue après l'exécution de tr'. Soit τ_P, et τ_Q des substitutions telles que $r_n = (r_f(\sigma''_\mathcal{P} \cup \sigma''_\mathcal{Q}))(\tau_P \cup \tau_Q)$ et $img(\tau_P)$ et $img(\tau_Q)$ ne contiennent que des termes clos.

Il existe des $\Sigma_0^\star$-recettes destructrices $R_1, \ldots, R_k$ telles que $R_i\phi''{\downarrow} = u_i\sigma''_\mathcal{P}\tau_P$ et $R_i\psi''{\downarrow} = v_i\sigma''_\mathcal{Q}\tau_Q$ d'après le lemme 5.4. Le lemme 5.10 s'applique à la règle r_n écrite sous la forme :

$$r_n = \mathsf{Pre}, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_n, v_n) \to \mathsf{Add}; \mathsf{Del}$$

Nous sommes dans le cas où $\mathsf{bad\text{-}flat} \notin \mathsf{Add}$, donc il existe un contexte constructeur C et une substitution τ telle que $r_f(\sigma''_\mathcal{P} \cup \sigma''_\mathcal{Q})\tau = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ où $u = C[u_1, \ldots, u_n]$ et $v = C[v_1, \ldots, v_n]$. Donc, considérons la trace $\mathsf{tr}' \cdot \mathsf{in}(c, C[R_1, \ldots, R_k])$. Cette étape peut être exécutée du côté $\mathcal{P}$ ainsi que du côté $\mathcal{Q}$. D'où le résultat. □

Il est maintenant possible de démontrer le théorème 5.1.

Théorème 5.1. *Soit $\mathcal{P}$ un protocole simple conforme à un système de types structuré $(\mathcal{T}_\mathcal{P}, \delta_\mathcal{P})$, et $\mathcal{Q}$ un autre protocole simple. Nous considérons l'ensemble* R *de règles :*

$$\mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))) \cup \mathsf{R}^{\mathsf{phase}} \cup \mathsf{R}^{\mathsf{test}_1}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{atom}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{check}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{pub}}_{\mathsf{fail}}$$

Soit $\Theta = \langle \mathcal{F}_0, \mathsf{Fact}_{\Sigma_0{}^\star}(\mathcal{P}, \mathcal{Q}), \mathsf{R} \rangle$ et $\Pi = \langle \Theta, \{\mathsf{bad}\} \rangle$. Nous avons $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$ si, et seulement si, Π a une solution de longueur

$$1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + \mathsf{nb}_{\mathsf{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_\mathcal{P}(\mathcal{P})) \times [1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + N]$$

où N est le nombre de noms apparaissant dans $\mathcal{P}$ qui ont le type d'une clé, c'est-à-dire tels que $\delta_\mathcal{P}(n)$ apparaît en position de clé dans $\delta_\mathcal{P}(\mathcal{P})$.

Démonstration. Montrons les deux implications séparément.

($\Rightarrow$) Dans le cas où $\mathcal{P} \not\sqsubseteq_t \mathcal{Q}$, nous savons d'après le théorème 3.1 qu'il existe un témoin de ce fait tel que $(\mathsf{tr}, \phi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ est quasiment typé. De plus, nous savons que tr ne contient que des recettes fortement simples. Nous considérons un tel témoin tr de longueur minimale, et pour lequel $\mathsf{Key}(\mathsf{tr})$ est minimal. Par minimalité de tr, il n'existe pas d'action $\mathsf{phase}\ i$ dans tr pour $i > \mathsf{maxphase}(\mathcal{P})$. Deux cas peuvent être distingués suivant que $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$ pour un certain ψ ou non.

Cas $(\mathsf{tr}, \psi) \in \mathsf{trace}(\mathcal{Q})$ pour un certain ψ. Le lemme 5.12 permet de conclure qu'il existe un chemin π_0 de $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q})$ vers S de longueur au plus

$$\mathsf{nb_{in}}(\mathsf{tr}) + \mathsf{nb_{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb_{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr})|) \times \mathsf{depth}(\phi)$$

tel que :

— $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'_\mathcal{P}, \mathcal{K}'_\mathcal{Q}) \uparrow S$ où $\mathcal{K}'_\mathcal{P}$ (resp. $\mathcal{K}'_\mathcal{Q}$) est la configuration obtenue après l'exécution de tr dans $\mathcal{P}$ (resp. $\mathcal{Q}$) ;

— $\{\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \mid R \in \mathsf{Key}(\mathsf{tr})\} \subseteq S$.

Dans le cas où $(\mathsf{tr}, \psi) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$, nous savons que $\phi \not\sqsubseteq_s^{\mathsf{simple}^+} \psi$. Nous considérons un test d'inclusion statique T pour $\sqsubseteq_s^{\mathsf{simple}^+}$ qui est minimal pour la mesure suivante (en ordre lexicographique) :

— $\mathsf{Key}(T) \setminus \mathsf{Key}(\mathsf{tr})$;

— sa taille, c'est-à-dire le nombre de symboles de fonction qui apparaissent dans T.

Le lemme 5.8 s'applique, avec $\mathcal{R}_0 = \mathsf{Key}(\mathsf{tr})$. Nous en déduisons l'existence d'un plan π_1 de bad de longueur au plus

$$(|\mathsf{Key}(T) \setminus \mathsf{Key}(\mathsf{tr})| + 1) \times \mathsf{depth}(\phi) + 1$$

De cette manière, nous obtenons un plan $\pi_0.\pi_1$ de bad de longueur au plus

$$\mathsf{nb_{in}}(\mathsf{tr}) + \mathsf{nb_{out}}(\mathsf{tr}) + \mathsf{maxphase}(\mathsf{tr}) + (\mathsf{nb_{in}}(\mathsf{tr}) + |\mathsf{Key}(\mathsf{tr}) \cup \mathsf{Key}(T)| + 1) \times \mathsf{depth}(\phi) + 1$$

Puisque $\mathsf{nb_{in}}(\mathsf{tr}) \leqslant \mathsf{nb_{in}}(\mathcal{P})$, $\mathsf{nb_{out}}(\mathsf{tr}) \leqslant \mathsf{nb_{out}}(\mathcal{P})$, $\mathsf{maxphase}(\mathsf{tr}) \leqslant \mathsf{maxphase}(\mathcal{P})$ (par minimalité de tr), $\mathsf{depth}(\phi) \leqslant \mathsf{depth}(\delta_\mathcal{P}(\mathcal{P}))$, il suffit de montrer que $|\mathsf{Key}(\mathsf{tr}) \cup \mathsf{Key}(T)| \leqslant N$ pour conclure. D'abord, nous avons $\{R\phi\downarrow \mid R \in \mathsf{Key}(\mathsf{tr}) \cup \mathsf{Key}(T)\} \setminus \Sigma_0^\# \leqslant N$ puisque toutes ces recettes apparaissent en position de clé d'une recette qui mène à un message de ϕ. Maintenant, dans le cas où il existe deux $\Sigma_0^\star$-recettes $K, K' \in \mathsf{Key}(\mathsf{tr}) \cup \mathsf{Key}(T) \cup \Sigma_0^\star$ telles que $K\phi\downarrow = K'\phi\downarrow$, alors par minimalité du témoin tr, nous avons $K\psi\downarrow = K'\psi\downarrow$. En remplaçant K ou K' par la recette qui est déductible le plus tôt, ou par la plus petite si elles sont déductibles au même moment, nous obtenons un test plus petit d'après notre mesure. Donc la longueur est plus petite que

$$\mathsf{nb_{in}}(\mathcal{P}) + \mathsf{nb_{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + [\mathsf{nb_{in}}(\mathcal{P}) + N + 1] \times \mathsf{depth}(\delta_\mathcal{P}(\mathcal{P})) + 1$$

Cas où tr ne passe pas dans $\mathcal{Q}$. Soit $\mathsf{tr} = \mathsf{tr}^{-1}.\alpha$. Nous avons $(\mathsf{tr}^{-1}, \psi^{-1}) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$ mais la dernière action α ne peut pas être exécutée. De plus, $(\mathsf{tr}^{-1}, \phi^{-1}) \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$. Le lemme 5.12 nous permet de conclure qu'il existe un chemin π_0 de $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{P}, \mathcal{Q})$ à S de longueur au plus

$$\mathsf{nb_{in}}(\mathsf{tr}^{-1}) + \mathsf{nb_{out}}(\mathsf{tr}^{-1}) + \mathsf{maxphase}(\mathsf{tr}^{-1}) + (\mathsf{nb_{in}}(\mathsf{tr}^{-1}) + |\mathsf{Key}(\mathsf{tr}^{-1})|) \times \mathsf{depth}(\phi^{-1})$$

tel que :

— $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'_\mathcal{P}, \mathcal{K}'_\mathcal{Q}) \uparrow S$ où $\mathcal{K}'_\mathcal{P} = (\mathcal{P}'; \phi'; \sigma'_\mathcal{P}; i')$ (resp. $\mathcal{K}'_\mathcal{Q} = (\mathcal{Q}'; \psi'; \sigma'_\mathcal{Q}; i')$) est la configuration obtenue après l'exécution de tr^{-1} dans $\mathcal{P}$ (resp. $\mathcal{Q}$) ;

— $\{\mathsf{att}(R\phi\downarrow, R\psi\downarrow) \mid R \in \mathsf{Key}(\mathsf{tr}^{-1})\} \subseteq S$.

Nous considérons trois cas suivant l'action α.

Cas : $\alpha = \mathsf{phase}\ i$. Ce cas est impossible puisque cette action peut être exécutée par $\mathcal{Q}$ à la seule condition que sa phase est inférieure à $i-1$, et c'est le cas puisque $\mathcal{P}$ et $\mathcal{Q}$ sont synchronisés et que l'action α s'exécute dans $\mathcal{P}$.

Cas : $\alpha = \mathsf{out}(c, \mathsf{w})$. Si une telle action ne peut pas être exécutée, alors cette action n'est pas disponible dans le processus ou aurait mené à l'émission d'un terme qui ne serait pas un message. Dans le premier cas, il existe une règle de protocole r qui peut être instanciée pour imiter cette étape. Dans le second cas, il faut considérer une instance de $\mathsf{Concrete}^-(r)$. Remarquons que pour une telle règle, $\mathsf{Flat}(r) = \{r\}$. Dans ce cas, nous utilisons seulement une seule règle, donc le plan est de longueur au plus

$$\mathsf{nb_{in}}(\mathsf{tr}^{-1}) + \mathsf{nb_{out}}(\mathsf{tr}^{-1}) + \mathsf{maxphase}(\mathsf{tr}^{-1}) + (\mathsf{nb_{in}}(\mathsf{tr}^{-1}) + |\mathsf{Key}(\mathsf{tr}^{-1})|) \times \mathsf{depth}(\phi^{-1}) + 1$$

Comme avant, c'est inférieur à

$$\mathsf{nb_{in}}(\mathcal{P}) + \mathsf{nb_{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_{\mathcal{P}}(\mathcal{P})) \times [1 + \mathsf{nb_{in}}(\mathcal{P}) + N] + 1$$

Cas : $\alpha = \mathsf{in}(c, R)$ avec $R = C[R_1, \ldots, R_k]$ une recette fortement simple. Considérons $\mathsf{in}(c, u)$ l'action correspondante dans $\mathcal{K}'_p$. Nous avons $R\phi'\downarrow = (u\sigma'_{\mathcal{P}})\tau_{\mathcal{P}}$ pour une certaine substitution $\tau_{\mathcal{P}}$. Si une telle action ne peut pas être exécutée, alors cette action n'est pas syntaxiquement disponible dans le processus ou alors le terme du côté Q ne correspond pas. Soit r la règle de protocole abstraite correspondante à cette étape.

D'après le lemme 5.7 appliqué pour $\mathcal{R} = \mathsf{Key}(\{R_1, \ldots, R_k\})$ et $\mathcal{R}_0 = \mathsf{Key}(\mathsf{tr}^{-1})$, il existe un plan π_1 de longueur au plus $|\mathsf{Key}(\{R_1, \ldots, R_k\}) \setminus \mathsf{Key}(\mathsf{tr}^{-1})| \times \mathsf{depth}(\phi)$ de $\{\mathsf{att}(R''\phi\downarrow, R''\psi\downarrow) \mid R'' \in \mathsf{Key}(\{R_1, \ldots, R_k\})\}$. Le lemme 5.6 s'applique donc à $\mathcal{R} = \{R_1, \ldots, R_k\}$: il existe un plan π_2 de longueur au plus $\mathsf{depth}(\phi^{-1}) = \mathsf{depth}(\phi)$ de

$$\{\mathsf{att}(R_1\phi\downarrow, R_1\psi\downarrow), \ldots, \mathsf{att}(R_k\phi\downarrow, R_k\psi\downarrow)\}.$$

Donc, de même qu'auparavant, $\pi_0.\pi_1.\pi_2$ est un plan de longueur au plus

$$\mathsf{nb_{in}}(\mathcal{P}) + \mathsf{nb_{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_{\mathcal{P}}(\mathcal{P})) \times [1 + \mathsf{nb_{in}}(\mathcal{P}) + N]$$

Si la réception n'est pas syntaxiquement disponible dans le processus, alors r est de la forme $\mathsf{Phase}(i), St(P, Q), \mathsf{att}(u, y) \to \mathsf{bad\text{-}proto}$ où y est une variable fraîche et i un entier. $St(P, Q)$ s'unifie avec le fait correspondant de S avec la substitution $\sigma'_{\mathcal{P}} \uplus \sigma'_{\mathcal{Q}} \uplus \tau_{\mathcal{P}}$ telle que

— $St(P, Q)(\sigma'_{\mathcal{P}} \uplus \sigma'_{\mathcal{Q}} \uplus \tau_{\mathcal{P}}) \in S$.

— $R\phi'\downarrow = u(\sigma'_{\mathcal{P}} \uplus \tau_{\mathcal{P}})$

— $\delta_{\mathcal{P}}(x(\sigma'_{\mathcal{P}} \uplus \tau_{\mathcal{P}})) \preccurlyeq \delta_{\mathcal{P}}(x)$ pour chaque $x \in vars_{\mathsf{left}}(r)$.

Nous pouvons appliquer le lemme 5.9 à r et $\sigma'_{\mathcal{P}} \uplus \sigma'_{\mathcal{Q}} \uplus \tau_{\mathcal{P}} \uplus \{y \mapsto R\psi'\downarrow\}$ (remarquons que $R\psi'\downarrow$ est un $\Sigma_0^\star$-message par minimalité du témoin tr). Nous obtenons une règle $r_f \in \mathsf{Flat}(r)$:

$$r_f = \mathsf{Pre}', \mathsf{att}(u'_1, v'_1), \ldots, \mathsf{att}(u'_k, v'_k) \to \mathsf{Add}'; \mathsf{Del}'$$

et une substitution σ' telle que :

— $x\sigma'$ est close pour toute variable $x \in vars(r)$.

— $\delta_{\mathcal{P}}(x\sigma') \preccurlyeq \delta_{\mathcal{P}}(x)$ pour toute variable $x \in vars_{\mathsf{left}}(r')$

— $(\mathsf{Pre}', \mathsf{Add}', \mathsf{Del}')\sigma' = (St(P,Q), \mathsf{bad\text{-}proto}, \emptyset)(\sigma'_{\mathcal{P}} \uplus \sigma'_{\mathcal{Q}} \uplus \tau_{\mathcal{P}} \uplus \{y \mapsto R\psi'\downarrow\})$

— $u(\sigma'_{\mathcal{P}} \uplus \tau_{\mathcal{P}}) = C[u'_1, \ldots, u'_k]$ et $R\psi'\downarrow = C[v'_1, \ldots, v'_k]$.

Comme les $\mathsf{att}(R_i\phi'\downarrow, R_i\psi'\downarrow)$ s'unifient avec $\mathsf{Pre}(r_f)$, il existe une règle r' dans $\mathsf{Concrete}^+(r_f)$ dont les préconditions sont exactement les $\mathsf{att}(R_i\phi'\downarrow, R_i\psi'\downarrow)$, ce qui conclut le cas où la réception n'est pas syntaxiquement disponible dans le processus.

Les cas où il existe une instruction $\mathsf{in}(c, v)$ côté Q peut être montré d'une manière semblable. Nous nous appuyons sur le lemme 5.9 où v s'unifie avec $C[z_1, \ldots, z_k]$, et nous avons un plan qui atteint bad en applicant simplement une règle concrète dans $\mathsf{Concrete}^-(\mathsf{Flat}(r))$. Sinon, le lemme 5.11 s'applique, et la règle de planification menant à bad est dans $\mathsf{Concrete}^+(\mathsf{Flat}(r))$ ou dans $\mathsf{Concrete}^-(\mathsf{Flat}(r))$.

Dans tous les cas, nous obtenons un plan de bad de longueur au plus

$$\mathsf{nb_{in}}(\mathcal{P}) + \mathsf{nb_{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_{\mathcal{P}}(\mathcal{P})) \times [1 + \mathsf{nb_{in}}(\mathcal{P}) + N] + 1$$

($\Leftarrow$) Nous montrons ce résultat par induction sur la longueur du chemin menant à bad. Nous considérons un plan linéaire de longueur minimale. Nous avons $\pi = r_1, \ldots, r_n$, et nous notons S_i l'état obtenu après le chemin $r_1, \ldots, r_i$. Le lemme 5.12 s'applique à $r_1, \ldots, r_{n-1}$. Nous obtenons la trace tr', et les $\Sigma_0^\star$-trames ϕ' et ψ' telles que :

— tr' contient seulement des recettes simples ;

— $(\mathsf{tr}', \phi') \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{P})$ et est quasiment typé pour $(\mathcal{T}_{\mathcal{P}}, \delta_{\mathcal{P}})$;

— $(\mathsf{tr}', \psi') \in \mathsf{trace}_{\Sigma_0^\star}(\mathcal{Q})$;

— $\mathsf{Fact}_{\Sigma_0^\star}(\mathcal{K}'_{\mathcal{P}}, \mathcal{K}'_{\mathcal{Q}}) \uparrow S_{n-1}$ où $\mathcal{K}'_{\mathcal{P}} = (\mathcal{P}'; \phi'; \sigma'_{\mathcal{P}}; i')$ (resp. $\mathcal{K}'_{\mathcal{Q}} = (\mathcal{Q}'; \psi'; \sigma'_{\mathcal{Q}}; i')$) est la configuration obtenue en exécutant tr dans $\mathcal{P}$ (resp. $\mathcal{Q}$).

Remarquons que nous avons donc $\mathsf{Phase}(i) \in S_{n-1}$. La règle r_n est ou bien dans $\mathsf{Concrete}^-(\mathsf{R_{Ana}}) \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{pub}}$; ou bien dans $\mathsf{Concrete}(\mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))$. Notamment, cette règle r_n ne peut pas être une règle de phase puisqu'une telle règle ne permettrait pas d'atteindre bad. Dans le cas où $r_n \in \mathsf{Concrete}^-(\mathsf{R_{Ana}}) \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathcal{R}_{\mathsf{fail}}^{\mathsf{pub}}$, nous concluons grâce à la proposition 5.1, et nous obtenons donc que les $\Sigma_0^\star$-trames ϕ' et ψ' obtenues après l'exécution de tr' ne sont pas en inclusion statique.

Le cas où $r_n \in \mathsf{Concrete}(\mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))$ ne se produit que quand $r_n \in \mathsf{Concrete}(\mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})))$. Il existe donc une règle de protocole r_f telle que $r_n \in \mathsf{Concrete}(\mathsf{Flat}(r_f))$.

Cas où r_f provient d'une émission. Dans ce cas, cette règle n'a pas été aplatie. Donc : $r_n \in \mathsf{Concrete}(r_f)$. Nous avons :

$$r_f = \mathsf{Phase}(i'), St(P,Q) \to \mathsf{Add}; \mathsf{Del}$$
$$r_n = \mathsf{Phase}(i'), f_0 \to \mathsf{Add}_n; \mathsf{Del}_n$$

pour certains ensembles de faits $\mathsf{Add}, \mathsf{Del}, \mathsf{Add}_n, \mathsf{Del}_n$ et $f_0 \in S_{n-1}$. Comme $r_n \in \mathsf{Concrete}(r_f)$ est applicable, $St(P,Q)$ s'unifie avec $f_0 \in S_{n-1}$ par hypothèse de récurrence : appelons σ une substitution telle que $St(P,Q)\sigma = f_0$. $(\mathsf{Add}, \mathsf{Del})$ s'unifie à gauche avec $(\mathsf{Add}_n, \mathsf{Del}_n)$ par σ. Si $(\mathsf{Add}, \mathsf{Del})$ s'unifie aussi à droite, alors $r_n \in \mathsf{Concrete}^+(r_f)$. Donc $\mathsf{Del} = \emptyset$ et $\mathsf{Add} = \mathsf{att}(u, c_0^\star), \mathsf{bad\text{-}proto}$. u doit être instanciée par un message comme r_n existe et est applicable. Donc il existe, dans $\mathcal{P}'$, un processus P sur un certain canal c qui commence par une émission mais le processus dans $\mathcal{Q}'$ sur le canal c ne commence pas par une émission. De plus, il est possible d'émettre un message, donc $\mathsf{tr.out}(c, \mathsf{w})$ est un témoin de non-inclusion.

Si $(\mathsf{Add}, \mathsf{Del})$ ne s'unifient pas à droite, alors $r_n \in \mathsf{Concrete}^-(r_f)$. Supposons :

$$r_f = \mathsf{Phase}(i'), St(P,Q) \to \mathsf{att}(u, c_0^\star), \mathsf{bad\text{-}proto}$$

Alors par définition de $\mathsf{Concrete}^-$, $u\sigma$ est un $\Sigma_0^\star$-message, mais $c_0^\star$ n'en est pas un, ce qui est impossible. Nous obtenons $r_f = \mathsf{Phase}(i'), St(P,Q) \to \mathsf{att}(u,v), St(P',Q'); St(P,Q)$. Comme $St(P,Q)\sigma \in S_{n-1}$, $u\sigma$ est un $\Sigma_0^\star$-message mais $v\sigma$ n'en est pas un. Donc l'émission est possible du côté $\mathcal{P}$, mais pas du côté $\mathcal{Q}$. De cette manière, nous concluons le cas de l'émission.

Cas où r_f provient d'une réception. Nous écrivons :

$$\begin{aligned} r_f &= \mathsf{Phase}(i'), St(P,Q), \mathsf{att}(u,v) \to \mathsf{Add}; \mathsf{Del} \\ r_n &= \mathsf{Phase}(i'), f_0, \mathsf{att}(t_1, t_1'), \ldots, \mathsf{att}(t_k, t_k') \to \mathsf{Add}_n; \mathsf{Del}_n \end{aligned}$$

où f_0 s'unifie à gauche avec $St(P,Q)$ et les t_i, t_i' sont des $\Sigma_0^\star$-messages. Nous avons $\mathsf{bad} \in \mathsf{Add}_n$. Comme r_n est applicable, il existe des recettes $R_1, \ldots, R_k$ telles que $R_i\phi'{\downarrow} = t_i$ et $R_i\psi'{\downarrow} = t_i'$ pour chaque i d'après le lemme 5.4.

De plus, la forme de la règle indique qu'il existe un processus $P = \mathsf{in}(c,u).P' \in \mathcal{P}'$. Q est le processus séquentiel sur c dans $\mathcal{Q}'$. Nous distinguons trois cas suivant l'origine de ce bad :

Premier cas : $\mathsf{bad} = \mathsf{bad\text{-}proto}$. Nécessairement $r_f = \mathsf{Phase}(i'), St(P,Q), \mathsf{att}(u,x) \to \mathsf{bad\text{-}proto}$ où x est une variable fraîche. Nous avons $r_n \in \mathsf{Concrete}^+(\mathsf{Flat}(r_f))$ ($\mathsf{Concrete}^-$ donnerait $\mathsf{bad\text{-}concrete}$) donc il existe une règle $r' = \mathsf{Phase}(i'), f_0', \mathsf{att}(u_1,v_1), \ldots, \mathsf{att}(u_k,v_k)) \to \mathsf{Add}'; \mathsf{Del}' \in \mathsf{Flat}(r_f)$ telle que $r_n \in \mathsf{Concrete}^+(r')$. En particulier, il existe une substitution σ_0 telle que $r_n = r'\sigma_0$ et $f_0'\sigma_0 = f_0$, $u_i\sigma_0 = t_i$ et $v_i\sigma_0 = t_i'$ pour chaque i. Donc $\mathsf{bad\text{-}flat} \notin \mathsf{Add}'$. Le lemme 5.10 s'applique, et il existe un contexte constructeur C et une substitution τ telle que $r_f\tau = \mathsf{Phase}(i'), f_0', \mathsf{att}(u,v) \to \mathsf{Add}; \mathsf{Del}$ et $u = C[u_1, \ldots, u_k]$ et $v = C[v_1, \ldots, v_k]$.

Les $\mathsf{att}(t_i, t_i')$ sont dans S_{n-1}. Donc d'après le lemme 5.4, il existe des recettes destructrices $R_1, \ldots, R_k$ telles que $R_i\phi'{\downarrow} = t_i$ et $R_i\psi'{\downarrow} = t_i'$.

Donc $\mathsf{tr.in}(c, C[R_1, \ldots, R_k])$ est une trace de $\mathcal{P}$ mais ce n'est pas une trace de $\mathcal{Q}$ puisqu'il n'y a pas de réception dans $\mathcal{Q}'$.

Second cas : $\mathsf{bad} = \mathsf{bad\text{-}flat}$. Nous avons $r_n \in \mathsf{Concrete}^+(\mathsf{Flat}(r_f))$ ($\mathsf{Concrete}^-$ donnerait $\mathsf{bad\text{-}concrete}$) Donc il existe une règle

$$r' = \mathsf{Phase}(i'), f_0', \mathsf{att}(u_1,v_1), \ldots, \mathsf{att}(u_k,v_k) \to \mathsf{bad\text{-}flat}; \emptyset \in \mathsf{Flat}(r_f)$$

et une substitution σ telles que $r_n = r'\sigma$. Donc en appliquant le lemme 5.10, il existe un contexte constructeur C, une substitution τ, un terme v' et deux ensembles Add_0 et Del_0 tels que

$$r_f\tau = \mathsf{Phase}(i'), f_0', \mathsf{att}(C[u_1, \ldots, u_n], v') \to \mathsf{Add}_0; \mathsf{Del}_0$$

mais v' ne s'unifie pas avec C.

Donc $\mathsf{tr.in}(c, C[R_1, \ldots, R_k])$ est une trace de $\mathcal{P}$, mais ce n'est pas une trace de $\mathcal{Q}$ (soit il n'y avait pas de réception dans Q, soit il y en avait une dont le motif ne s'unifiait pas à $C[R_1, \ldots, R_k]\psi'{\downarrow}$). Nous obtenons donc un témoin de non-inclusion.

Troisième cas : $\mathsf{bad} = \mathsf{bad\text{-}concrete}$. Nous avons $r_n \in \mathsf{Concrete}^-(\mathsf{Flat}(r_f))$ ($\mathsf{Concrete}^+$ ne donnerait pas $\mathsf{bad\text{-}concrete}$). Donc il existe une règle

$$r' = \mathsf{Phase}(i'), f_0', \mathsf{att}(u_1,v_1), \ldots, \mathsf{att}(u_k,v_k) \to \mathsf{Add}'; \mathsf{Del}' \in \mathsf{Flat}(r_f)$$

pour certains $\mathsf{Add}', \mathsf{Del}'$ et une substitution σ telle que $r_n =^{\mathsf{left}} r'\sigma$, mais r_n et r' ne s'unifient pas à droite.

Dans tous les cas du lemme 5.10, nous obtenons un contexte constructeur C, une substitution τ et deux ensembles $\mathsf{Add}_0, \mathsf{Del}_0$ tels que $r_f\tau = \mathsf{Phase}(i'), St(P,Q), \mathsf{att}(u', v') \rightarrow \mathsf{Add}_0; \mathsf{Del}_0$ où $u = C[u_1, \ldots, u_k]$.

Donc $\mathsf{tr.in}(c, C[R_1, \ldots, R_k])$ est une trace valide de $\mathcal{P}$ (nous avons $C[R_1, \ldots, R_k]\phi'\downarrow = u' = u\tau$).

De plus, $C[R_1\psi'\downarrow, \ldots, R_k\psi'\downarrow]$ ne s'unifie pas avec v, (comme $r_n \in \mathsf{Concrete}^-(r')$). Donc

$$\mathsf{tr.in}(c, C[R_1, \ldots, R_k])$$

n'est pas une trace de $\mathcal{Q}$, ce qui nous donne un témoin de non-inclusion et conclut donc la preuve. □

5.4 Algorithme

Dans le chapitre 4, nous avions laissé deux aspects de côté. D'une part, l'algorithme du graphe de planification ne fonctionne que si le problème est *borné*, c'est-à-dire s'il existe un oracle de planification (voir section 4.1.2). Maintenant que la traduction a été définie complètement, la section 5.4.1 présente l'oracle. La difficulté principale consiste à abstraire les règles d'équivalence statique. D'autre part, l'algorithme du graphe de planification complet (voir section 4.3.2) dispose de deux conditions de terminaison : il s'arrête soit par saturation (étape (E)), soit lorsqu'il a atteint une borne fixée au départ (étape (F)). Dans la section 5.4.2, nous discutons l'utilité de cette borne en donnant deux exemples pour lesquels elle est nécessaire.

5.4.1 Oracle de planification

Le théorème 5.1 montre qu'il existe un plan de longueur bornée si, et seulement si, les protocoles ne sont pas en équivalence, mais ce n'est pas suffisant pour montrer que nous pouvons obtenir ce plan en temps fini. En particulier, comme les traces ne sont (quasiment) typées que par rapport à $\mathcal{P}$, des termes de taille arbitraire peuvent apparaître côté $\mathcal{Q}$, ce qui signifie que l'ensemble des règles concrètes à parcourir est infini. Puisque la longueur des traces est bornée, il serait en fait possible de borner les termes qui peuvent apparaître côté $\mathcal{Q}$, mais ce serait au prix d'une explosition combinatoire du nombre de règles. Pour maintenir un niveau d'efficacité acceptable, nous avons préféré disposer d'un oracle de planification, comme défini à la section 4.1.2. De cette manière, à partir d'un ensemble fini de faits, nous obtenons toujours un ensemble fini de règles applicables.

Pour les règles symboliques, l'oracle s'obtient assez naturellement. Il suffit de considérer l'ensemble des faits, et de regarder s'ils s'unifient à gauche avec les règles symboliques, ce qui donne un nombre fini de concrétisations possibles, positives ou négatives. De plus, pour unifier le formalisme, nous notons $\mathsf{Concrete}(r) = \mathsf{Concrete}^+(r) = \{r\}$ et $\mathsf{Concrete}^-(r) = \emptyset$ pour chaque $r \in \mathsf{R}^{\mathsf{phase}}$. La seule difficulté restante pour établir cet oracle consiste donc à définir des règles symboliques pour l'équivalence statique (les ensembles $\mathsf{R}^{\mathsf{test}_1}_{\mathsf{fail}}$, $\mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}}$, $\mathsf{R}^{\mathsf{atom}}_{\mathsf{fail}}$,$\mathsf{R}^{\mathsf{check}}_{\mathsf{fail}}$,$\mathsf{R}^{\mathsf{pub}}_{\mathsf{fail}}$ contiennent chacun une infinité de règles concrètes), sachant que les règles d'analyse sont déjà représentées par $\mathsf{R}_{\mathsf{ana}}$. Pour les obtenir, il suffit de considérer les règles suivantes :

$$\begin{aligned}
r_1 &= \mathsf{att}(x,y), \mathsf{att}(x,y) \rightarrow \emptyset \\
r_2(C) &= \mathsf{att}(C[x_1, \ldots, x_n], C[y_1, \ldots, y_n]), \mathsf{att}(x_1, y_1), \ldots, \mathsf{att}(x_n, y_n) \rightarrow \emptyset \text{ pour } C \text{ un contexte.} \\
r_{\mathsf{pub}} &= \mathsf{att}(\mathsf{pub}(x), \mathsf{pub}(y)) \rightarrow \emptyset \\
r_{\mathsf{check}} &= \mathsf{att}(\mathsf{sign}(x_1, x_2), \mathsf{sign}(y_1, y_2)), \mathsf{att}(\mathsf{vk}(x_2), \mathsf{vk}(y_2)) \rightarrow \emptyset
\end{aligned}$$

L'ensemble de ces règles est noté $\mathsf{R}^{\mathsf{static}}$.

La concrétisation est ensuite définie exactement comme pour les autres règles : $\mathsf{Concrete}^+(r)$ est l'ensemble des $r\sigma$ tels que $r\sigma$ ne contient que des $\Sigma_0^\star$-messages et $\delta(x\sigma) \preccurlyeq \delta(x)$ pour tout $x \in vars_{\mathsf{left}}(r)$, avec σ une substitution. Remarquons que, pour chacune des règles symboliques ci-dessus, l'ensemble $\mathsf{Concrete}^+(r)$ ne contient que des règles avec $\mathsf{Add} = \mathsf{Del} = \emptyset$. Ces règles concrètes ne modifient donc absolument pas l'ensemble des solutions du problème de planification.

Si $r = \mathsf{Pre} \to \mathsf{Add}; \mathsf{Del}$ alors $\mathsf{Concrete}^-(r)$ contient les règles de la forme $f_1, \ldots, f_k \to \mathsf{bad}$ pour chaque séquence $f_1, \ldots, f_k$ de faits qui s'unifient à gauche avec Pre (par la substitution σ_L) et où u est un $\Sigma_0^\star$-message pour chaque $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_L$, si elle vérifie au moins l'une des conditions suivantes :

— $f_1, \ldots, f_k$ ne s'unifie pas à droite avec Pre

— $f_1, \ldots, f_k$ s'unifie à droite avec Pre par la substitution σ_R, mais v n'est pas un $\Sigma_0^\star$-message pour un certain $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_R$.

L'ensemble $\mathsf{Concrete}^-(r_1)$ nous donne $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1}$. Si $Cont$ est l'ensemble des contextes tels qu'il existe $\tau_1, \ldots, \tau_n$ avec $C[\tau_1, \ldots, \tau_n] \in St_{\mathsf{opti}}(\delta_{\mathcal{P}}(\mathcal{P}))$, nous obtenons $\mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} = \cup_{C \in Cont} \mathsf{Concrete}^-(r_2(C))$. Nous avons également $\mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}} = \mathsf{Concrete}^-(r_{\mathsf{pub}})$ et $\mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} = \mathsf{Concrete}^-(r_{\mathsf{check}})$. Il ne reste donc que $\mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}}$, dont les règles sont ajoutés directement par l'oracle de planification.

Soit F un ensemble fini de faits, et R un ensemble de règles (symboliques). L'oracle $\mathcal{O}(F, \mathsf{R})$ se calcule de la façon suivante.

1. $Res := \emptyset$.
2. Pour chaque fait $\mathsf{att}(u, v) \in F$, si u est un atome et si v n'est pas un atome, $Res := Res \cup \{\mathsf{att}(u, v) \to \mathsf{bad}\}$.
3. Pour chaque règle $r \in \mathsf{R}$, regarder s'il existe une séquence de faits $f_1, \ldots, f_k$ qui s'unifie à gauche avec r, et respecte le typage à gauche de r.
4. Si les faits s'unifient également à droite (et donnent des messages), il existe une substitution σ telle que $r\sigma$ est une règle concrète. $Res := Res \cup \{r\sigma\}$.
5. Si les faits ne s'unifient pas à droite, ou s'il s'unifient avec la substitution σ_R mais que v n'est pas un $\Sigma_0^\star$-message pour un certain $\mathsf{att}(u, v) \in \mathsf{Add}\sigma_R$, alors $Res := Res \cup \{f_1, \ldots, f_n \to \mathsf{bad}\}$.
6. Une fois que toutes les règles ont été parcourues, éliminer les règles inutiles de Res puis renvoyer Res.

L'étape 2 garantit que nous obtenons toutes les règles applicables de $\mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}}$. Les concrétisations positives des règles de R sont obtenues à l'étape 4 et les concrétisations négatives à l'étape 5. Il est clair que ce calcul s'effectue en temps fini, puisque le nombre de règles symboliques est fini et l'ensemble de faits F est fini. Plus formellement :

Lemme 5.13. *L'oracle $\mathcal{O}(\cdot, \mathsf{R})$ vérifie la propriété de l'oracle de planification (propriété 4.1).*

Démonstration. Il suffit de démontrer le fait suivant :

Fait. Soient F un ensemble fini de faits, et $\mathsf{R} = \mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})) \cup \mathsf{R}^{\mathsf{phase}} \cup \mathsf{R}^{\mathsf{static}}$. Posons $\mathsf{R}_0 = \mathcal{O}(F, \mathsf{R})$. Alors R_0 est fini et $\mathsf{R}_0 = \mathsf{R}_1 \setminus \mathsf{R}_\emptyset$ où R_1 est l'ensemble des règles applicables dans F de

$$\mathsf{R}_{\mathsf{Concrete}} = \mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})) \cup \mathsf{R}^{\mathsf{phase}}) \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$$

et $\mathsf{R}_\emptyset$ est un ensemble des règles inutiles, c'est-à-dire de la forme $\mathsf{Pre} \to \emptyset$.

R_0 est fini car :

— À l'étape 2, au plus $|F|^2$ faits sont considérés, où $|F|$ est le cardinal de F.

— À l'étape 3, au plus $|K|$ règles symboliques sont considérées. Pour chaque règle r, la recherche d'une séquence de faits de taille k (le nombre de préconditions de r) suppose d'explorer au plus k fois F qui est fini.

— Dans les étapes ultérieures, une seule règle et un seul ensemble de préconditions sont traités à la fois.

Soit R_1 l'ensemble des règles de $\mathsf{R}_{\mathsf{Concrete}}$ applicables dans F. Commençons par montrer que $\mathsf{R}_1 \setminus \mathsf{R}_\emptyset \subseteq \mathsf{R}_0$: soit $r \in \mathsf{R}_1$, nous allons prouver que ou bien $r \in \mathsf{R}_0$ ou bien r est inutile.

Premier cas : $r \in \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}}$. Comme r est applicable dans F, nous avons $\mathsf{Pre}(r) \subseteq F$. Donc il existe un fait $\mathsf{att}(u, v) \in F$ tels que u est un atome mais v n'est pas un atome. Donc à l'étape 2, une règle $\mathsf{att}(u, v) \to \mathsf{bad}$ est ajoutée, et nous avons $r = \mathsf{att}(u, v) \to \mathsf{bad}$, donc $r \in \mathsf{R}_0$, sauf si r a été éliminée à la dernière étape (et alors r est inutile).

Deuxième cas : $r \in \mathsf{Concrete}^+(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})) \cup \mathsf{R}^{\mathsf{phase}})$. Alors il existe une règle $r_0 \in \mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})) \cup \mathsf{R}^{\mathsf{phase}}$ et des faits $f_1, \ldots, f_k$ tels que $f_1, \ldots, f_k$ s'unifient avec $\mathsf{Pre}(r_0)$ et $\mathsf{Pre}(r) = \{f_1, \ldots, f_k\}$. De plus, comme r est applicable dans F, $f_1, \ldots, f_k \in F$. Comme $r_0 \in \mathsf{R}$, ces faits sont détectés à l'étape 3 et r est obtenue à l'étape 4, donc $r \in \mathsf{R}_0$, sauf si r a été éliminée à la dernière étape (et alors r est inutile).

Troisième cas : $r \in \mathsf{Concrete}^-(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})))$. Alors il existe une règle $r_0 \in \mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}))$ et des faits $f_1, \ldots, f_k$ tels que $f_1, \ldots, f_k$ s'unifient à gauche, mais pas à droite, avec $\mathsf{Pre}(r_0)$ et $\mathsf{Pre}(r) = \{f_1, \ldots, f_k\}$. De plus, comme r est applicable dans F, $f_1, \ldots, f_k \in F$. Comme $r_0 \in \mathsf{R}$, ces faits sont détectés à l'étape 3 et r est obtenue à l'étape 5, donc $r \in \mathsf{R}_0$, sauf si r a été éliminée à la dernière étape (et alors r est inutile).

Quatrième cas : $r \in \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_1} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{check}} \cup \mathsf{R}_{\mathsf{fail}}^{\mathsf{pub}}$. Nous traitons uniquement le cas $r \in \mathsf{R}_{\mathsf{fail}}^{\mathsf{test}_2}$ car les autres sont similaires. La règle r est de la forme $\{\mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k), \mathsf{att}(C[u_1, \ldots, u_k], v) \to \mathsf{bad}$ où $C \in \mathit{Cont}$ et $v \neq C[v_1, \ldots, v_k]\}$. Les faits $\mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_k, v_k), \mathsf{att}(C[u_1, \ldots, u_k], v)$ s'unifient à gauche avec $r_2(C)$. Ils sont donc détectés à l'étape 3, et nous obtenons la règle r à l'étape 3, donc $r \in \mathsf{R}_0$, sauf si r a été éliminée à la dernière étape (et alors r est inutile).

Réciproquement, il reste à montrer que $\mathsf{R}_0 \subseteq \mathsf{R}_1 \setminus \mathsf{R}_\emptyset$ où $\mathsf{R}_\emptyset$ est un ensemble de règles de la forme $\mathsf{Pre} \to \emptyset$. D'abord, R_0 ne contient que des règles applicables dans F, puisqu'aux étapes 2 et 3, les préconditions des règles sont obtenues à partir de F. Ensuite, comme R_1 est l'ensemble des règles applicables dans F de $\mathsf{R}_{\mathsf{Concrete}}$, et comme R_0 est un ensemble de règles applicables dans F, nous avons $\mathsf{R}_0 \cap \mathsf{R}_{\mathsf{Concrete}} \subseteq \mathsf{R}_1$. Si nous montrons que $\mathsf{R}_0 \subseteq \mathsf{R}_{\mathsf{Concrete}} \setminus \mathsf{R}_\emptyset$ où $\mathsf{R}_\emptyset$ est un ensemble de règles de la forme $\mathsf{Pre} \to \emptyset$, nous aurons $\mathsf{R}_0 \subseteq \mathsf{R}_1 \setminus \mathsf{R}_\emptyset$. Nous cherchons donc à montrer que $\mathsf{R}_0 \subseteq \mathsf{R}_{\mathsf{Concrete}} \setminus \mathsf{R}_\emptyset$. Soit $r \in \mathsf{R}_0$: trois cas se présentent.

1. r a été obtenue à l'étape 2. Dans ce cas, $r = \mathsf{att}(u, v) \to \mathsf{bad}$ où u est un atome, mais v n'est pas un atome. Donc $r \in \mathsf{R}_{\mathsf{fail}}^{\mathsf{atom}}$ et ainsi $r \in \mathsf{R}_{\mathsf{Concrete}}$. Si r était inutile, elle aurait été éliminée à la dernière étape, donc $r \in \mathsf{R}_{\mathsf{Concrete}} \setminus \mathsf{R}_\emptyset$.
2. r a été obtenue à l'étape 4. Il existe une règle $r_0 \in \mathsf{R}$ et des faits $f_1, \ldots, f_k \in F$ tels que $\mathsf{Pre}(r) = \{f_1, \ldots, f_k\}$ et $f_1, \ldots, f_k$ s'unifient avec $\mathsf{Pre}(r_0)$ à gauche (étape 3) et à droite (étape 4). Donc $r \in \mathsf{Concrete}^+(r_0)$. Si r était inutile, elle aurait été éliminée à la dernière étape, donc $r \in \mathsf{Concrete}^+(r_0) \setminus \mathsf{R}_\emptyset$.
3. r a été obtenue à l'étape 5. Il existe une règle $r_0 \in \mathsf{R}$ et des faits $f_1, \ldots, f_k \in F$ tels que $\mathsf{Pre}(r) = \{f_1, \ldots, f_k\}$ et $f_1, \ldots, f_k$ s'unifient avec $\mathsf{Pre}(r_0)$ à gauche (étape 3) mais pas à droite (étape 4). Donc $r \in \mathsf{Concrete}^-(r_0)$. Si r était inutile, elle aurait été éliminée à la dernière étape, donc $r \in \mathsf{Concrete}^-(r_0) \setminus \mathsf{R}_\emptyset$.

Dans le premier cas, nous avons déjà $r \in \mathsf{R}_{\mathsf{Concrete}}$. Dans le second et le troisième cas, nous avons montré que $r \in \mathsf{Concrete}(\mathsf{R}) \setminus \mathsf{R}_\emptyset$. Il reste donc à prouver que $\mathsf{Concrete}(\mathsf{R}) \setminus \mathsf{R}_\emptyset$ ne contient que des règles de $\mathsf{R}_{\mathsf{Concrete}}$. Nous avons $\mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q})) \cup \mathsf{R}^{\mathsf{phase}}) \subseteq \mathsf{R}_{\mathsf{Concrete}}$. L'ensemble $\mathsf{Concrete}^+(\mathsf{R}^{\mathsf{static}})$ ne contient que des règles de la forme $\mathsf{Pre} \to \emptyset$, puisque les règles de $\mathsf{R}^{\mathsf{static}}$ sont de cette forme. Soit $r \in \mathsf{Concrete}^-(\mathsf{R}^{\mathsf{static}})$. Montrons que $r \in \mathsf{R}^{\mathsf{test}_1}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{atom}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{check}}_{\mathsf{fail}} \cup \mathsf{R}^{\mathsf{pub}}_{\mathsf{fail}}$. Nous traitons uniquement le cas $r \in \mathsf{Concrete}^-(r_2(C))$ pour $C \in Cont$, les autres étant similaires. Si $r \in \mathsf{Concrete}^-(r_2(C))$, alors $\mathsf{Pre}(r)$ est un ensemble de faits $f_1, \ldots, f_k$ tel que $f_1, \ldots, f_k$ s'unifie à gauche avec $\mathsf{Pre}(r_2(C))$, mais pas à droite. L'unification à gauche donne $f_i = \mathsf{att}(u_i, v_i)$ pour chaque $i < k$, et $f_k = \mathsf{att}(C[u_1, \ldots, u_{k-1}], v_k)$; la non-unification à droite donne $v_k \neq C[v_1, \ldots, v_{k-1}]$. Donc $r \in \mathsf{R}^{\mathsf{test}_2}_{\mathsf{fail}} \subseteq \mathsf{R}_{\mathsf{Concrete}}$.

Ainsi, nous avons montré que $\mathsf{Concrete}(\mathsf{R}_{\mathsf{ana}} \cup \mathsf{Flat}(\mathsf{Rule}(\mathcal{P}, \mathcal{Q}) \cup \mathsf{R}^{\mathsf{phase}}) \cup \mathsf{Concrete}^-(\mathsf{R}^{\mathsf{static}}) \subseteq \mathsf{R}_{\mathsf{Concrete}}$. Comme toutes les règles de R_0 sont applicables dans F, nous avons $\mathsf{R}_0 \subseteq \mathsf{R}_1 \setminus \mathsf{R}_\emptyset$ où l'ensemble $\mathsf{R}_\emptyset$ ne contient que des règles inutiles. □

5.4.2 Discussion

La définition de l'oracle de planification à la section 5.3.3 et le lemme 4.1 qui démontre que le calcul de chaque itération du graphe de planification s'arrête, permettent de garantir la terminaison de l'algorithme de la section 4.3.2 pour notre traduction. Le théorème 5.1 garantit que si nous calculons le graphe jusqu'à la borne

$$1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + \mathsf{nb}_{\mathsf{out}}(\mathcal{P}) + \mathsf{maxphase}(\mathcal{P}) + \mathsf{depth}(\delta_{\mathcal{P}}(\mathcal{P})) \times [1 + \mathsf{nb}_{\mathsf{in}}(\mathcal{P}) + N]$$

nous sommes assurés de ne pas manquer d'attaques, grâce à la proposition 4.1. De plus, l'algorithme du graphe de planification possède une condition d'arrêt par saturation, à l'étape (E) de l'algorithme complet (voir section 4.3.2). Dans la plupart des cas rencontrés en pratique, cette condition est suffisante, et la borne est donc superflue. Cependant, dans les deux exemples qui suivent, cette borne est nécessaire, c'est-à-dire que la saturation ne suffit pas à garantir l'arrêt.

Premier exemple. Étant donné un canal c, considérons les processus $P(c)$ et $Q(c)$ définis comme suit, avec a une constante publique et x une variable.

$$P(c) := \mathsf{in}(c, \langle x, a\rangle).\mathsf{out}(c, \langle x, a\rangle)$$
$$Q(c) := \mathsf{in}(c, \langle x, a\rangle).\mathsf{out}(c, \langle\langle x, x\rangle, a\rangle)$$

Nous considérons $\mathcal{K}_{\mathcal{P}} = \{P(c_1); P(c_2)\}$ et $\mathcal{K}_{\mathcal{Q}} = \{Q(c_1); Q(c_2)\}$ pour certains canaux publics c_1, c_2. Avec l'état initial $\mathsf{att}(b, b)$, où b est une constante publique appartenant à la connaissance initiale de l'attaquant, les faits suivants vont être ajoutés successivement en calculant le graphe de planification :

— $\mathsf{att}(\langle a, b\rangle, \langle b, a\rangle)$

— $\mathsf{att}(\langle b, a\rangle, \langle\langle b, b\rangle, a\rangle)$

— $\mathsf{att}(\langle b, a\rangle, \langle\langle\langle b, b\rangle, \langle b, b\rangle\rangle, a\rangle)$

— ...

En fait, $\mathsf{att}(\langle b, a\rangle, \langle\langle b, b\rangle, a\rangle)$ peut être ajouté de deux manières différentes, qui correspondent aux deux ordres possibles des émissions sur c_1 et c_2 (ou bien c_1 puis c_2, ou bien c_2 puis c_1). Puisqu'il y a deux manières d'obtenir ce fait, il ne sera jamais en exclusion mutuelle avec les états précédents. En particulier, le fait $\mathsf{att}(\langle b, a\rangle, \langle\langle b, b\rangle, a\rangle)$ et l'état indiquant que le processus sur le canal c_1 n'a

pas commencé ne sont pas en exclusion mutuelle, donc ils peuvent être utilisés pour déclencher les règles de planification qui mènent à

$$\mathsf{att}(\langle b, a\rangle, \langle\langle\langle b, b\rangle, \langle b, b\rangle\rangle, a\rangle)$$

Puisque les termes calculés du côté Q croissent à chaque étape, ce calcul ne s'arrête que lorsque le nombre d'étapes dépasse la borne donnée par le théorème 5.1.

Il est facile de voir que $\mathcal{K}_{\mathcal{P}}$ n'est pas en inclusion de trace avec $\mathcal{K}_{\mathcal{Q}}$, car un attaquant peut distinguer entre b et $\langle b, b\rangle$. Donc, dès qu'un message est émis, les trames résultantes ne sont plus en inclusion statique. Ainsi, une manière de couper l'exécution plus tôt consiste à forcer la procédure à s'arrêter dans l'exploration du graphe dès qu'une attaque est trouvée. Cependant, il est possible de modifier cet exemple pour empêcher toute attaque d'être détectée, et donc de forcer l'algorithme à atteindre la borne. L'idée est simplement de combiner l'exemple ci-dessus avec l'exemple 4.9 de la section 4.3.4 (reformulé dans le langage des protocoles). En effet, cet exemple démontre que le graphe de planification réalise une surapproximation de l'ensemble des états accessibles. Des faits de plus en plus profonds existeront dans le graphe, mais comme ils ne seront pas réellement accessibles (après vérification par le solveur SAT) la procédure ne détectera pas d'attaque et ne pourra s'arrêter que grâce à la borne.

Deuxième exemple. Plus concrètement, considérons les processus $P_0(c_0), P_1(c_0)$ et $Q_1(c_0)$ suivants, avec k un nom représentant une clé secrète et a, b, c des constantes publiques.

$$\begin{aligned} P_0(c_0) &= \mathsf{in}(c_0, x).\mathsf{out}(c_0, \mathsf{senc}(x, k)) \\ P_1(c_0) &= \mathsf{in}(c_0, \langle \mathsf{senc}(a, k), \mathsf{senc}(b, k), \mathsf{senc}(c, k)\rangle^3).P(c_0) \\ Q_1(c_0) &= \mathsf{in}(c_0, \langle \mathsf{senc}(a, k), \mathsf{senc}(b, k), \mathsf{senc}(c, k)\rangle^3).Q(c_0) \end{aligned}$$

Nous considérons les configurations :

$$\begin{aligned} \mathcal{K}'_{\mathcal{P}} &= \{P_0(c_0); P_0(c_1); P_1(c_2); P_1(c_3)\} \\ \mathcal{K}'_{\mathcal{Q}} &= \{P_0(c_0); P_0(c_1); Q_1(c_2); Q_1(c_3)\} \end{aligned}$$

avec c_0, c_1, c_2, c_3 des noms de canaux publics. Les processus P_0 sur les canaux c_0 et c_1 sont utilisés comme des oracles de chiffrement, afin que l'attaquant puisse chiffrer exactement deux des trois constantes a, b et c avec la clé k. De cette manière, nous modélisons les deux jetons de l'exemple 4.9. Comme dans cet exemple, la réception sur le canal c_2 (et donc aussi sur le canal c_3) est déclenchée dans le graphe de planification. Ensuite, les processus $P(c_2)$, $P(c_3)$, $Q(c_2)$ et $Q(c_3)$ nous ramènent au cas précédent, mais chaque fois que bad est ajouté au graphe, le solveur SAT répond qu'il n'est pas réellement accessible. Donc la procédure continue d'explorer le graphe jusqu'à atteindre la borne, puisqu'aucune attaque ne sera jamais trouvée (en effet, les protocoles sont en équivalence de trace).

5.5 Conclusion

Nous avons donné une traduction correcte et complète de l'inclusion de trace, pour des protocoles simples, sous forme de problème de planification. Toutes les optimisations sont démontrées, et le calcul d'une borne sur la longueur des plans, ainsi que la définition d'un oracle de planification, permettent d'appliquer la technique de résolution présentée au chapitre précédent. Dans leur travaux, Armando et Compagna [21] énoncent l'existence d'une borne en s'appuyant sur les résultats de Rusinowitch et Turuani [94]. En effet, Rusinowitch et Turuani bornent le nombre d'opérations que l'attaquant doit réaliser entre deux actions d'un agent honnête. Puisque le nombre d'actions des agents honnêtes est lui-même borné, il en découle une borne sur la longueur d'un plan. Nos calculs sont plus précis, car nous tenons compte de la possibilité de paralléliser et de retenir les clés déjà déduites.

Les optimisations présentées ici se font au prix d'une restriction assez importante de l'ensemble des protocoles qui peuvent être considérés. Même en admettant les limites théoriques des chapitres précédents, plusieurs pistes d'amélioration pourraient être envisagées. Il serait possible, par exemple, de rendre paramétrique l'ensemble de primitives considérées. Cependant, pour maintenir un niveau d'efficacité équivalent, il serait nécessaire d'automatiser les optimisations sur l'équivalence statique.

Une autre extension possible consisterait à relâcher l'hypothèse sur les protocoles simples. Ainsi, lorsque les protocoles sont déterministes, la difficulté principale consiste à trouver un appariement entre les actions des deux protocoles. Une méthode pour surmonter cet obstacle consisterait donc à considérer le cas des protocoles dont les canaux sont tous distincts. Dans cette situation, l'appariement est évident (il est donné par les canaux). Mais, pour un protocole déterministe, deux actions sur un même canal ne peuvent jamais être confondues. Il faudrait donc montrer que deux protocoles $\mathcal{P}$ et $\mathcal{Q}$ déterministes peuvent toujours se réécrire comme deux protocoles $\mathcal{P}'$ et $\mathcal{Q}'$ dont les canaux sont tous distincts, de sorte que $\mathcal{P}$ et $\mathcal{Q}$ sont équivalents si, et seulement si, $\mathcal{P}'$ et $\mathcal{Q}'$ sont équivalents. Cette extension permettrait d'obtenir une plus grande liberté de modélisation.

Dans le chapitre suivant, nous montrerons l'efficacité de notre démarche, à travers l'implémentation de l'outil SAT-Equiv, son application à plusieurs études de cas, qui permettent d'établir une comparaison expérimentale avec les autres outils pour l'équivalence de protocoles dans le cadre d'un nombre borné de sessions.

6 SAT-Equiv : Implémentation et études de cas

Dans le chapitre 4, nous avons présenté les problèmes de planification, et l'agorithme du graphe de planification, qui permet de les résoudre. Cet algorithme a été complété par la traduction de l'équivalence dans le langage de la planification, dans le chapitre 5. De cette manière, nous disposons d'une procédure de décision pour l'inclusion (approximative) de trace, et donc pour l'équivalence. Ce dernier chapitre commence par décrire l'outil SAT-Equiv, c'est-à-dire l'implémentation de la procédure de décision, à la section 6.1. La suite du chapitre est dédiée à l'évaluation de la méthode proposée. Dans la section 6.2, nous présentons donc neuf protocoles qui nous serviront de base d'évaluation. Remarquons que nous nous sommes limités à des protocoles qui pouvaient être modélisés par des processus simples. En particulier, nous avons laissé de côté les protocoles de vote électronique, car la modélisation du mélange des votes n'est pas déterministe, ou repose sur l'utilisation de primitives qui sortent de notre cadre, comme les preuves sans divulgation de connaissance. Nous étudions, pour chaque protocole, l'existence d'un système de types auquel il soit conforme dans la section 6.3, puisque c'est l'une des hypothèses du théorème 5.1. Ensuite, la section 6.4 présente les autres outils pour l'équivalence de traces dans le modèle borné, puis établit une comparaison entre ces outils et SAT-Equiv sur chacun des protocoles énumérés à la section 6.2. Enfin, la section 6.5 montre comment les performances de l'outil, associées à un résultat de petite attaque, permettent dans certains cas d'atteindre des preuves pour un nombre arbitraire de sessions.

6.1 Implémentation

Dans cette section, nous décrivons les détails de l'implémentation. La section 6.1.1 donne des précisions sur l'outil, ainsi qu'une rapide description de la sortie de SAT-Equiv. La section 6.1.2 décrit plus précisément les structures de données utilisées.

6.1.1 L'outil SAT-Equiv

L'implémentation a été réalisée en Ocaml. Elle contient environ 5000 lignes de code, et les sources, ainsi que tous les fichiers d'exemple dont il sera question dans ce chapitre, se trouvent sur la page de l'outil [3]. Dans ce chapitre, nous utilisons exclusivement la version `0.3`. L'ordre des opérations réalisées pour vérifier l'équivalence est le suivant :

1. **Analyse lexicale.**
2. **Analyse syntaxique.** La nature des primitives utilisées (Y a-t-il des primitives asymétriques ? Quelle est la taille maximale des tuples ?) est également détectée.
3. **Protocoles simples.** Il est vérifié que les protocoles sont simples, et les processus nuls sont éliminés. Si la vérification échoue, le programme lève une exception.
4. **Appariement des processus.** Les processus séquentiels sont appariés par canaux pour les inclusions $P \sqsubseteq_{at} Q$ et $Q \sqsubseteq_{at} P$.
5. **Inclusion de P dans Q.** Les opérations suivantes sont réalisées (dans l'ordre) :

— *Vérification que P est conforme à son système de type.* Si P n'est pas conforme, un message d'erreur est affiché mais l'analyse continue, puisqu'elle permet d'éliminer les attaques bien typées.

— *Traduction.* Les protocoles appariés pour $P \sqsubseteq_{at} Q$ sont traduits comme règles de planification.

— *Algorithme du graphe de planification.* La traduction comme formule SAT est effectuée après chaque étape de l'algorithme de planification, s'il y a une chance qu'il y ait une attaque (`bad` a été trouvé). Si une attaque est trouvée, le programme s'arrête et affiche l'attaque. Si aucune attaque n'est trouvée, un message indique que cette inclusion a été prouvée.

6. **Inclusion de Q dans P.** Les opérations sont effectuées dans le même ordre.

Quand aucune attaque n'est trouvée, la sortie se présente de la manière suivante :

```
fichier.pi
P include Q:

 Number of nodes: N
 Number of steps: s/S
 MiniSAT calls: c

No attack has been found.
Q include P:

 Number of nodes: N'
 Number of steps: s'/S'
 MiniSAT calls: c'

No attack has been found.
```

`fichier.pi` désigne le nom du fichier. `N` et `N'` désignent le nombre de noeuds (règles et faits) dans les graphes de planification. `s` et `s'` indiquent le nombre d'itérations réellement effectuées dans l'algorithme du graphe de planification, tandis que `S` et `S'` donnent la borne du théorème 5.1. Les nombres `c` et `c'` comptent les appels à minisat. Ainsi, $\mathtt{s}-\mathtt{c}+1$ donne le numéro de la première itération de l'algorithme du graphe de planification à laquelle `bad` est apparu.

Lorsqu'un témoin de non-inclusion $P \not\sqsubseteq_{at} Q$ est trouvé, la sortie affiche :

```
fichier.pi
P include Q:

 Number of nodes: N
 Number of steps: s/S
 MiniSAT calls: c

 [DESCRIPTION]

 Number of variables (approx.): V
 Number of clauses: C
```

`N`, `s`, `S` et `c` renvoient les mêmes informations que dans le cas où il n'y a pas d'attaque. `s` est donc le numéro de l'itération où l'attaque a été trouvée et `N` le nombre de noeuds du graphe de

planification à ce moment-là. `[DESCRIPTION]` contient une description de la trace d'attaque. Le nombre `V` est une approximation de nombre de variables utilisées pour la formule SAT qui a permis de trouver l'attaque. `V` est calculé par $\texttt{N} \times \texttt{S}$ qui est un majorant (le nombre exact de variables n'est jamais calculé explicitement par l'outil). `C` donne le nombre de clauses de la formule SAT.

6.1.2 Structures de données

La représentation interne du graphe de planification est donnée par un quintuplet composé de :

1. Un entier représentant le nombre de nœuds du graphe
2. Une fonction `fact -> int` qui associe un numéro de nœud à chaque fait du graphe.
3. Une fonction `rule_label -> int` qui associe un numéro de nœud à chaque règle du graphe, représentée par son étiquette qui l'identifie de manière unique.
4. Un tableau qui contient la description des nœuds à partir de leur numéro : dans la case `i`, ce tableau contient les informations suivantes sur le nœud `i` :
   - Le fait ou l'étiquette de la règle correspondant au nœud.
   - La liste Pre des numéros de nœuds reliés au nœud `i` par une flèche $\longrightarrow_{pre}$. Si le nœud `i` est un fait, la liste représente toutes les règles dont ce fait est une précondition. Si le nœud `i` est une règle, la liste représente toutes ses préconditions.
   - La liste Add des numéros de nœuds reliés au nœud `i` par une flèche $\longrightarrow_{add}$. Si le nœud `i` est un fait, la liste représente toutes les règles qui ajoutent ce fait. Si le nœud `i` est une règle, la liste contient les faits de son ensemble Add.
   - La liste Del des numéros de nœuds reliés au nœud `i` par une flèche $\longrightarrow_{del}$. Si le nœud `i` est un fait, la liste représente toutes les règles qui suppriment ce fait. Si le nœud `i` est une règle, la liste contient les faits de son ensemble Del.
   - Un entier qui représente la date d'appariation du fait dans le graphe.
5. Un tableau à double entrée qui représente les exclusions mutuelles.

Comme le nombre de nœuds du graphe n'est pas connu à l'avance, les dimensions des tableaux des points 4 et 5 sont amenées à évoluer au cours du temps. Comme chaque augmentation de taille amène à recopier le tableau, il est essentiel d'éviter d'effectuer cette opération trop souvent, et donc d'affecter suffisamment d'espace à chaque fois, sans toutefois utiliser une trop grande mémoire. Pour cette raison, chaque fois que les tableaux sont remplis, leurs tailles sont doublées. En particulier, l'expérience montre que le tableau représentant les mutex joue un rôle crucial dans les performances, et il est important d'en optimiser l'accès, ainsi que la taille.

L'implémentation des substitutions constitue un autre aspect important. Lorsque nous substituons une valeur à une variable, nous avons besoin d'un accès rapide à la valeur de cette variable : les structures de données dynamiques, comme les tableaux ou les tables de hachage, représentent donc de bons candidats. Mais les substitutions sont construites terme par terme, et nous voulons factoriser leur calcul. Par exemple, si nous cherchons à unifier les motifs u_1, u_2 avec les termes clos de l'ensemble de termes T, nous commençons par chercher tous les termes clos $t \in T$ qui s'unifient avec u_1, et nous obtenons un ensemble E de substitutions σ telles que $u_1\sigma \in T$. Ensuite, pour chaque substitution $\sigma \in E$, nous cherchons l'ensemble des σ' telles que $(u_2\sigma)\sigma' \in T$, et nous obtenons un ensemble E_σ de substitutions σ'. Les substitutions que nous cherchons sont donc les $\sigma' \circ \sigma$ avec $\sigma \in E$ et $\sigma' \in E_\sigma$. Si nous utilisons une structure de donnée dynamique, il est nécessaire de recopier σ pour calculer chacune des substitutions $\sigma'_1 \circ \sigma, \ldots, \sigma'_n \circ \sigma$ avec $E_\sigma = \{\sigma'_1, \ldots, \sigma'_n\}$, ce qui est très coûteux. Pour éviter ce problème, il est plus naturel de représenter une substitution par une structure de donnée récursive, utilisant des pointeurs, comme une liste. Après plusieurs

tentatives, la solution retenue est une solution hybride : il s'agit de représenter les substitutions comme listes de tables de hachage. Pour calculer $\sigma' \circ \sigma$, nous nous contentons de concaténer deux listes, tandis que pour rechercher un élément nous devons parcourir la liste et chercher dans chaque table de hachage. Comme, à chaque opération $\sigma' \circ \sigma$, la substitution σ' n'a jamais été fusionnée, la liste qui la représente est de longueur 1, et le calcul de $\sigma' \circ \sigma$ se fait en temps constant. Pour appliquer une subtitution à une variable, nous avons besoin de $\ell \times h$ opérations, où ℓ est la longueur de la liste qui représente une substitution (en principe pas plus de 2 ou 3) et h la complexité de la recherche dans la table de hachage.

6.2 Protocoles

Cette section décrit les protocoles que nous utiliserons dans ce chapitre, pour étudier la conformité et comparer les outils. Dans deux cas (les protocoles de Yahalom-Paulson et de Needham-Schroeder-Lowe) nous donnons également une version marquée, pour des raisons de conformité qui seront discutées plus précisément en section 6.3. Notons toutefois que ce sont les versions marquées qui seront considérées dans la section 6.4. Les protocoles de *Wide Mouth Frog* et Denning-Sacco sont donnés sans horodatage, car rien, ni dans notre modèle, ni dans celui des autres outils, ne permet d'en tenir compte. Nous n'utilisons pas non plus la phase pour éviter d'éliminer artificiellement deux outils, puisqu'Akiss est le seul (hormis SAT-Equiv) à l'autoriser.

Pour tous les protocoles, à l'exception de *Passive Authentication*, la propriété considérée est l'indistinguabilité, à la fin de l'exécution, entre $\mathsf{senc}(m_0, k)$ et $\mathsf{senc}(m_1, x)$ où m_0 et m_1 sont des constantes publiques, k est un nom n'apparaissant nulle part ailleurs dans le protocole, qui représente une clé fraîche, et x est une variable, qui lors d'une exécution honnête, représente une clé ou un nonce supposé rester secret. Cette propriété est plus forte que le secret de x (si l'attaquant sait déduire x, il peut déchiffrer $\mathsf{senc}(m_1, x)$ mais pas $\mathsf{senc}(m_0, k)$, ce qui lui permet de faire la différence entre les deux protocoles), mais plus faible que le secret fort de x (l'attaquant peut par exemple connaître $\mathsf{hash}(x)$ sans être capable de distinguer entre les deux protocoles). Elle est utile pour démontrer que la clé qui instancie x peut être utilisée sans danger pour établir un canal chiffré. En effet, le secret faible ne suffit pas, et il est superflu de démontrer le secret fort de la clé alors qu'il sera cassé par le premier échange qui l'utilise comme clé. Nous donnerons des exemples de modélisation de cette propriété sur le protocole de Denning-Sacco et celui d'Otway-Rees. Dans le cas du protocole *Passive Authentication*, nous démontrons une variante de l'anonymat des données, car aucun nonce n'est supposé rester secret.

Pour les protocoles utilisant uniquement le chiffrement symétrique, les scénarios sont construits en envisageant un nombre croissant de sessions, jusqu'à parvenir à un scénario complet, qui est ensuite répliqué. Plus précisément, le scénario à trois sessions correspond à un échange honnête, avec une session de chaque rôle (Alice, Bob et le serveur de confiance). Pour le scénario à 6 sessions, nous doublons chaque rôle du scénario à 3 sessions. Pour les scénarios suivants, nous ajoutons un agent malhonnête Charlie. Le scénario à sept sessions est construit à partir du scénario à trois sessions, auquel nous ajoutons des échanges de chaque agent avec Charlie, plus les sessions correspondantes sur le serveur. Nous obtenons donc, en plus des trois sessions de départ : une session où Alice contacte Charlie, une session du serveur qui communique avec Alice et Charlie, une session où Bob répond à Charlie, et une session du serveur qui communique avec Bob et Charlie. Les sessions de Charlie lui-même sont jouées par l'attaquant, qui dispose des clés secrètes de Charlie. Pour parvenir à 10, 12 et 14 sessions, nous ajoutons des échanges supplémentaires entre les agents : un échange honnête complet pour le scénario à 10 sessions, et des échanges avec Charlie pour obtenir 12 et 14 sessions. À partir de 21 sessions, tous les scénarios considérés sont des réplications du scénarios à 7 sessions, à l'exception des scénarios pour Needham-Schroeder avec le chiffrement symétrique, dont une explication précise est donnée à la section 6.5. Tous les fichiers d'exemple utilisés sont

disponibles sur la page de l'outil [3].

Denning-Sacco. Nous considérons une variante sans horodatage du protocole de Denning-Sacco décrit par Clark et Jacob [57] :

$$\begin{array}{ll} A \to S: & A, B \\ S \to A: & \{B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \\ A \to B: & \{K_{ab}, A\}_{K_{bs}} \end{array}$$

Le but de ce protocole est d'établir une clé K_{ab} partagée entre Alice (A) et Bob (B). Au début, Alice et Bob partagent chacun une clé (respectivement K_{as} et K_{bs}) avec un serveur de confiance S. Alice envoie au serveur son nom et celui de Bob, qu'elle veut joindre. Le serveur crée une clé fraîche K_{ab}, puis chiffre un message $\{K_{ab}, A\}_{Kbs}$ à l'intention de B et l'inclut dans le message chiffré $\{B, K_{ab}, \{K_{ab}, A\}_{Kbs}\}_{Kas}$ qu'il envoie à Alice. Alice déchiffre le message avec la clé K_{as}, récupère la clé K_{ab} et transmet le message $\{K_{ab}, A\}_{Kbs}$ qu'elle ne peut pas déchiffrer. Bob ouvre ce message et récupère la clé K_{ab}. C'est sur cette clé que nous étudierons la propriété. Plus précisément, le protocole sera modélisé de la manière suivante, où c_1, c_2 et c_3 sont des canaux publics, a, b, m_0 et m_1 des constantes publiques, k_{as}, k_{bs}, k_{ab} et k des noms et $x_{AB}, x_B, y_{AB}, y'_{AB}$ des variables.

$$\begin{array}{rcl} P_A & = & \mathsf{out}(c_1, \langle a, b\rangle). \\ & & \mathsf{in}(c_1, \mathsf{senc}(\langle b, x_{AB}, x_B\rangle, k_{as})). \\ & & \mathsf{out}(c_1, x_B) \\ P_S & = & \mathsf{in}(c_2, \langle a, b\rangle). \\ & & \mathsf{out}(c_2, \mathsf{senc}(\langle b, k_{ab}, \mathsf{senc}(\langle k_{ab}, a\rangle, k_{bs})\rangle, k_{as})) \\ P_B & = & \mathsf{in}(c_3, \mathsf{senc}(\langle y_{AB}, a\rangle, k_{bs})).\mathsf{out}(c_3, \mathsf{senc}(m_0, y_{AB})) \\ Q_B & = & \mathsf{in}(c_3, \mathsf{senc}(\langle y'_{AB}, a\rangle, k_{bs})).\mathsf{out}(c_3, \mathsf{senc}(m_1, k)) \end{array}$$

Nous étudions l'équivalence de trace entre les protocoles $\mathcal{P} = \{P_A; P_S; P_B\}$ et $\mathcal{Q} = \{P_A; P_S; Q_B\}$. La propriété est codée par la dernière émission sur le canal c_3, tandis que toutes les autres instructions correspondent à la spécification du protocole.

Remarquons que dans cette version (sans horodatage), il existe une faille : l'attaquant peut effectuer une attaque par rejeu, en renvoyant le message $\{K_{ab}, A\}_{K_{bs}}$ lors d'une session ultérieure, comme décrit ci-dessous.

$$\begin{array}{ll} A \to S: & A, B \\ S \to A: & \{B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \\ A \to B: & \{K_{ab}, A\}_{K_{bs}} \\ I \to B: & \{K_{ab}, A\}_{K_{bs}} \end{array}$$

Il n'est pas très intéressant de comparer l'efficacité des outils en présence d'attaques, puisque cette efficacité peut dépendre par exemple de l'ordre dans lequel les traces sont énumérées, qui relève des détails d'implémentation voire de la spécification : si nous savons que l'attaque porte sur l'inclusion $Q \sqsubseteq_{at} P$, nous pouvons échanger les rôles des protocoles P et Q pour que SAT-Equiv découvre le témoin (beaucoup) plus rapidement. De même, l'ordre dans lequel les processus sont décrits peut modifier cette efficacité. Pour ne pas biaiser la comparaison, nous considérerons uniquement des scénarios où cette attaque n'existe pas.

Needham-Schroeder Symétrique. Nous considérons le protocole de Needham-Schroeder avec le chiffrement symétrique tel que décrit par Clark et Jacob [57].

$$\begin{array}{ll} A \to S : & A, B, N_a \\ S \to A : & \{B, N_a, K_{ab}, \{A, K_{ab}\}_{K_{bs}}\}_{K_{as}} \\ A \to B : & \{A, K_{ab}\}_{K_{bs}} \\ B \to A : & \{Req, N_b\}_{K_{ab}} \\ A \to B : & \{Rep, N_b\}_{K_{ab}} \end{array}$$

Le but de ce protocole est d'authentifier mutuellement Alice et Bob grâce à un serveur de confiance S, et d'établir une clé partagée K_{ab}, par exemple afin de s'identifier plus tard. Comme nous étudions seulement l'équivalence, nous codons simplement la propriété sur K_{ab}. Il s'agit d'une version légèrement complexifiée du protocole de Denning-Sacco, et la situation initiale est la même : Alice et Bob partagent chacun une clé secrète avec le serveur S (respectivement K_{as} et K_{bs}). Alice initie la communication en envoyant les deux noms des participants, plus un nonce frais N_a. Comme pour le protocole de Denning-Sacco, le serveur crée une clé fraîche K_{ab}, puis chiffre le message A, K_{ab} avec la clé K_{bs}. Ce premier chiffré est inclus dans le message $\{B, N_a, K_{ab}, \{A, K_{ab}\}_{K_{bs}}\}_{K_{as}}$ que le serveur envoie à Alice. Elle déchiffre ce message, récupère la clé K_{ab} et transmet le message $\{A, K_{ab}\}_{K_{bs}}$ à Bob. De cette manière, il récupère la clé K_{ab}, puis crée un nonce N_b. Il envoie alors $\{Req, N_b\}_{K_{ab}}$ à Alice, où *Req* (pour *requête*) est une constante publique. Alice lui répond $\{Rep, N_b\}_{K_{ab}}$ où *Rep* (pour *réponse*) est une autre constante publique. Comme pour le protocole de Denning-Sacco, l'attaquant peut tenter de rejouer la clé K_{ab} en envoyant le message $\{A, K_{ab}\}_{K_{bs}}$ obtenu lors d'une session précédente. Cependant, il n'est pas capable de déchiffrer le message $\{Req, N_b\}_{K_{ab}}$ de Bob et l'échange s'arrête là.

Wide Mouth Frog. Nous considérons une variante sans horodatage du protocole *Wide Mouth Frog* décrit par Clark et Jacob [57].

$$\begin{array}{ll} A \to S : & A,\ \{B, K_{ab}\}_{K_{as}} \\ S \to B : & \{A, K_{ab}\}_{K_{bs}} \end{array}$$

Comme pour le protocole de Denning-Sacco, l'objectif est d'établir une clé K_{ab} partagée entre Alice et Bob, sachant qu'au départ, Alice et Bob partagent chacun une clé avec le serveur S (respectivement K_{as} et K_{bs}). Alice crée une clé fraîche K_{ab}, puis la chiffre, ainsi que l'identité de Bob, avec la clé K_{as}. Elle envoie son identité, ainsi que le message $\{B, K_{ab}\}_{K_{as}}$, au serveur. Celui-ci déchiffre avec la clé K_{as}, puis émet le message $\{A, K_{ab}\}_{K_{bs}}$ vers Bob, qui le déchiffre et récupère à la fois la clé K_{ab} et l'identité A de son interlocuteur. Comme dans le cas du protocole de Denning-Sacco, cette version sans horodatage comporte une faille : l'attaquant peut rejouer le dernier message $\{A, K_{ab}\}_{K_{bs}}$. Encore une fois, nous considérerons uniquement des scénarios où cette attaque n'existe pas.

Yahalom-Paulson. Le protocole décrit par Clark et Jacob [57] est représenté comme suit :

$$\begin{array}{ll} A \to B : & A, N_a \\ B \to S : & B, N_b, \{A, N_a\}_{K_{bs}} \\ S \to A : & N_b, \{B, K_{ab}, N_a\}_{K_{as}}, \{A, B, K_{ab}, N_b\}_{K_{bs}} \\ A \to B : & \{A, B, K_{ab}, N_b\}_{K_{bs}}, \{N_b\}_{K_{ab}} \end{array}$$

Le but de ce protocole est d'établir une clé secrète K_{ab} partagée entre Alice et Bob. Au départ, chacun d'entre eux en partage une (K_{as} ou K_{bs}) avec le serveur de confiance. Alice envoie son identité ainsi qu'un nonce frais N_a à Bob. Bob contacte le serveur, et lui envoie son identité, un autre nonce frais N_b, ainsi que le message qu'il a reçu d'Alice, après l'avoir chiffré par K_{bs}. Le serveur crée une clé fraîche K_{ab}, puis s'adresse à Alice et lui envoie trois messages : le nonce N_b reçu de Bob ; un message chiffré par K_{as}, contenant l'identité de Bob, la clé K_{ab}, et le nonce N_a ; et un message chiffré par K_{bs}, à transmettre à Bob, contenant les deux identités A et B, la clé fraîche K_{ab} et le nonce N_b créé par Bob. Alice reçoit cette communication, déchiffre le second message et récupère la clé K_{ab}. Elle chiffre alors le nonce N_b avec cette clé, puis transmet le message $\{A, B, K_{ab}, N_b\}_{K_{bs}}$ reçu du serveur ainsi que $\{N_b\}_{K_{ab}}$.

Le protocole est relativement complexe. En particulier, au moment où Alice reçoit le message du serveur, elle ne connaît ni la clé K_{ab}, ni le nonce N_b. Le message $\{N_b\}_{K_{ab}}$ est donc un composé de deux messages reçus par Alice, ce qui rend la vérification plus difficile. En particulier, ce message s'unifie avec la plupart des sous-termes chiffrés, ce qui s'oppose à la conformité du protocole. Pour cette raison, nous considérerons la version marquée :

$$\begin{array}{ll} A \to B : & A, N_a \\ B \to S : & B, N_b, \{1, A, N_a\}_{K_{bs}} \\ S \to A : & N_b, \{2, B, K_{ab}, N_a\}_{K_{as}}, \{3, A, B, K_{ab}, N_b\}_{K_{bs}} \\ A \to B : & \{3, A, B, K_{ab}, N_b\}_{K_{bs}}, \{4, N_b\}_{K_{ab}} \end{array}$$

Otway-Rees. Le protocole d'Otway-Rees [89] a été décrit dans l'exemple 1.12. Nous rappelons sa définition :

$$\begin{array}{ll} A \to B : & M, A, B, \{N_a, M, A, B\}K_{as} \\ B \to S : & M, A, B, \{N_a, M, A, B\}K_{as}, \\ & \qquad \{N_b, M, A, B\}K_{bs} \\ S \to B : & M, \{N_a, K_{ab}\}K_{as}, \{N_b, K_{ab}\}K_{bs} \\ B \to A : & M, \{N_a, K_{ab}\}K_{as} \end{array}$$

L'objectif de ce protocole est d'établir la clé partagée K_{ab} entre Alice et Bob, sachant que, comme pour les autres protocoles reposant sur le chiffrement symétrique, ces deux agents partagent des clés K_{as} et K_{bs} avec le serveur de confiance. Pour ce protocole, la propriété porte sur la clé K_{ab} telle qu'elle est reçue par Alice. Remarquons que si les tuples sont modélisés avec des paires chaînées à droite ($\langle \cdot, \langle \cdot, \ldots \langle \cdot, \cdot \rangle \ldots \rangle$ pour $\langle \ldots \rangle$), il existe une attaque : l'attaquant peut remplacer (dans l'échange $S \to B$) $\{N_a, K_{ab}\}_{K_{as}}$ et $\{N_b, K_{ab}\}_{K_{bs}}$ par les messages $\{N_a, M, A, B\}_{K_{as}}$ et $\{N_b, M, A, B\}_{K_{bs}}$ qui ont été envoyés à l'étape précédente. En effet, Bob s'attend à recevoir $\mathsf{senc}(\langle n_b, z_{ab} \rangle, k_{bs})$ et reçoit $\mathsf{senc}(\langle n_b, \langle m, \langle a, b \rangle \rangle \rangle, k_{bs})$ qui s'unifie par $\{z_{ab} \triangleright \langle m, \langle a, b \rangle \rangle\}$. Cette attaque disparaît avec l'utilisation des tuples, ou simplement en utilisant des paires chaînées à gauche.

Passive Authentication. Ce protocole intervient dans la lecture du passeport électronique. La puce RFID (P) communique avec le lecteur de l'autorité (L). La clé k_{sign} appartient au pays émetteur du passeport, et ne se trouve donc pas sur la puce. Il a été décrit par l'ICAO (pour *International Civil Aviation Organization*) [88].

$$\begin{array}{ll} L \to P : & \{Read\}_{ks}, \mathsf{mac}(\{Read\}_{ks}, km) \\ P \to L : & \{data, Sig\}_{ks}, \mathsf{mac}(\{data, Sig\}_{ks}, km) \text{ avec } Sig = \mathsf{sign}(data, k_{sign}) \end{array}$$

Le but de ce protocole est que le lecteur de l'autorité de contrôle récupère les données du passeport, qui sont représentées ici par *data*. Comme pour le protocole BAC (voir l'introduction), le lecteur commence par récupérer les clés ks et km par lecture optique. Ensuite, il envoie un message $\{Read\}_{ks}, \mathsf{mac}(\{Read\}_{ks}, km)$ pour commencer l'échange. Ce message est essentiellement inutile, car il peut être rejoué par un attaquant une fois qu'il a été émis. La puce répond en envoyant les données biométriques *data* qu'elle contient, ainsi qu'une signature *Sig* de ces données, dans un message chiffré $\{data, Sig\}_{ks}$. Elle envoie également un MAC de ce message avec la clé km. Le lecteur peut alors déchiffrer ces La signature *Sig* est intégrée à la puce dès sa création par l'autorité émettrice, ce qui signifie que la puce ne possède pas la clé de signature k_{sign}. La présence de *data* avec *Sig* permet simplement à l'autorité de contrôle de lire directement les données *data*. Dans notre modèle, où getmsg permet de retrouver le message à partir de la signature, il est possible de remplacer $data, Sig$ par Sig.

Remarquons que le message $\{data, Sig\}_{ks}, \mathsf{mac}(\{data, Sig\}_{ks}, km)$ peut être calculé à l'avance, dès la création du passeport. Il est donc possible d'implémenter ce protocole de manière passive : c'est ce qui lui donne son nom. Cependant, cette propriété est une faiblesse, car le protocole n'est pas intraçable, puisque chaque exécution est complètement identique. De plus, le lecteur vérifie uniquement l'authenticité des données *data*, mais ne vérifie pas la validité du passeport : il est possible de copier un passeport. La propriété que nous vérifions est la suivante : nous demandons que l'attaquant soit incapable de repérer deux exécutions du protocole avec des données identiques mais des clés différentes. Elle peut sembler arbitraire, mais c'est une propriété analogue à la propriété classique d'anonymat, qui consiste dans le secret fort des données *data*. Nous n'avons pas retenu l'anonymat classique car il supposait de répéter toujours des échanges identiques pour augmenter le nombre de sessions.

Active Authentication. Ce protocole est également l'un des protocoles du passeport, décrit par l'ICAO [88]. La puce RFID (P) du passeport communique avec le lecteur de l'autorité de contrôle (L).

$$\begin{array}{ll} L \to P : & \{Init, N_l\}_{ks}, \mathsf{mac}(\{Init, N_l\}_{ks}, km) \\ P \to L : & \{\langle N_p, N_l\rangle, Sig\}_{ks}, \mathsf{mac}(\{Sig\}_{ks}) \text{ avec } Sig = \mathsf{sign}(\langle N_p, N_l\rangle, k_{sign}) \end{array}$$

Le but de ce protocole est d'authentifier le passeport vis-à-vis de l'autorité de contrôle et d'établir des secrets partagés N_l et N_p. Le passeport possède une clé de signature k_{sign}. Lecteur et passeport partagent des clés ks et km qui ont été obtenues par lecture optique. Le lecteur initie l'échange en émettant un message $Init$ (qui est une constante publique) et un nonce frais N_l chiffrés par la clé ks. Le lecteur envoie également un MAC pour assurer l'intégrité de l'échange. La puce vérifie le MAC, déchiffre le message et récupère le nonce N_l. Elle crée alors un nonce N_p et signe le message $\langle N_p, N_l\rangle$ avec la clé k_{sign}, et elle chiffre $\langle N_p, N_l\rangle, Sig$ avec la clé ks. Elle envoie alors ce message au lecteur, ainsi qu'un MAC qui permet de garantir l'intégrité. Sur ce protocole, nous formulons notre propriété sur le nonce N_p reçu par le lecteur.

Needham-Schroeder-Lowe. Il s'agit de la version corrigée par Lowe [84] du protocole de Needham-Schroeder [87].

$$\begin{array}{ll} A \to B : & \{N_a, A\}_{pk_B} \\ B \to A : & \{N_a, N_b, B\}_{pk_A} \\ A \to B : & \{N_b\}_{pk_B} \end{array}$$

Le rôle de ce protocole est d'authentifier mutuellement Alice et Bob, et d'établir un secret partagé N_b. Au début, Alice et Bob disposent chacun d'une clé secrète et d'une clé publique associées, et connaissent la clé de l'autre agent. Alice commence l'échange en envoyant un nonce frais N_a et son identité A chiffrés sous la clé pk_B. Bob reçoit ce message, le déchiffre, et récupère le nonce N_a. Il crée un nonce N_b et envoie le message $\{N_a, N_b, B\}_{pk_A}$. Alice le déchiffre, obtient le nonce N_b, et répond en chiffrant ce nonce avec la clé publique de Bob. Nous codons notre propriété sur le nonce N_b, du côté de Bob. Nous donnons également une version marquée de ce protocole :

$$\begin{array}{ll} A \to B : & \{1, N_a, A\}_{pk_B} \\ B \to A : & \{2, N_a, N_b, B\}_{pk_A} \\ A \to B : & \{3, N_b\}_{pk_B} \end{array}$$

Denning-Sacco avec signature. Ce protocole est une variante du protocole de Denning-Sacco proposée par Blanchet [36].

$$\begin{array}{ll} A \to B : & \{\mathsf{sign}(\langle A, B, K_a\rangle, sk_A)\}_{pk_B} \\ B \to A : & \{N_b\}_{K_a} \end{array}$$

Ce protocole a pour but d'authentifier mutuellement Alice et Bob et d'établir un nonce N_b et une clé K_a, secrets et partagés entre les deux interlocuteurs. Au début, Alice connaît la clé de chiffrement asymétrique de Bob et Bob connaît la clé de vérification de la signature d'Alice. Alice commence par créer une clé fraîche K_a. Elle ajoute son identité et celle de Bob, signe le message avec sa clé secrète de signature sk_A, et obtient donc $\mathsf{sign}(\langle A, B, K_a\rangle, sk_A)$. Elle chiffre alors ce message avec la clé publique de Bob, puis le lui envoie. Bob déchiffre avec sa clé secrète, puis récupère le message signé et vérifie la signature. Il obtient la clé K_a, crée un nonce frais N_b et envoie $\{N_b\}_{K_a}$ à Alice. Nous codons la propriété sur le nonce N_b.

6.3 Conformité

Dans cette section, nous étudions la conformité des protocoles définis à la section précédente. Tout protocole est conforme au système de types trivial qui ne contient qu'un seul type. Cependant, nous nous intéressons particulièrement aux systèmes de types structurés, qui sont les seuls pour lesquels nous avons établi la traduction au chapitre 5. Étant donné un protocole, plusieurs éléments peuvent modifier sa conformité vis-à-vis d'un système de types donné : le scénario et la propriété considérés, mais aussi les choix de modélisation.

Il a été montré, dans le cas des primitives symétriques, qu'un protocole est conforme à un système de types pour un nombre arbitraire de sessions s'il lui est conforme pour deux sessions [51]. Si rien n'indique que ce résultat soit faux pour les primitives asymétriques, il n'a pas été démontré dans ce cadre. Pour cette raison, il est naturel de fixer à deux le nombre de sessions considérées pour la conformité des protocoles utilisant exclusivement le chiffrement symétrique. De plus, comme toute autre limite que deux sessions est tout aussi arbitraire dans le cas des primitives asymétriques, nous utilisons un scénario qui correspond à deux exécutions complètes, quelque soient les primitives utilisées. Ce choix est d'autant moins important qu'il n'est utile que lorsque le protocole n'est conforme à aucun système de types structuré : dans le cas contraire, la conformité à un système

de type structuré a été vérifiée pour un grand nombre de sessions, indiqué à la figure 6.2 dans la colonne SAT-Equiv. En effet, avant de lancer la procédure de décision, l'outil analyse la conformité au système de type structuré indiqué par l'utilisateur.

Lorsque nous modélisons un protocole, le choix d'utiliser des tuples ou des paires chaînées peut changer la conformité à un système de types. Par exemple, si k est un nom, a et b sont des constantes et x et y sont des variables, le termes chiffrés $\mathsf{senc}(\langle a, \langle b, x\rangle^2\rangle^2, k)$ et $\mathsf{senc}(\langle a, y\rangle^2, k)$ sont unifiables avec la substitution $\{y \triangleright \langle\langle b, x\rangle^2\}$. Pourtant, $\mathsf{senc}(\langle a, b, x\rangle^3, k)$ et $\mathsf{senc}(\langle a, y\rangle^2)$ ne sont pas unifiables.

Dans le cas où les paires sont chaînées, il existe encore deux sous-cas, selon si les paires sont chaînées par la droite ou par la gauche, ce qui peut à nouveau influencer le résultat final. Or, le plus souvent, les paires représentent la concaténation, pour laquelle la longueur des messages attendus permet en pratique de connaître la taille des tuples. Pour ces deux raisons, à savoir par réalisme, et pour éviter de traiter un trop grand nombre de cas, nous nous contentons de donner les résultats sur les modélisations utilisant des tuples.

Lorsque les protocoles ne sont conformes à aucun système de types structuré, il est possible de modifier légèrement le protocole, en ajoutant des marques, pour remédier à ce problème. Ces résultats sont exposés à la figure 6.1. Dans tous les cas présentés ici, ajouter des marques suffit à garantir la conformité à un système de types structuré. Cet ajout ne peut jamais remettre en cause la conformité, et c'est pour cette raison que nous n'avons donné les versions marquées que des protocoles de Needham-Schroeder-Lowe et de Yahalom-Paulson.

6.4 Comparaison

Dans cette section, nous analysons les protocoles présentés en section 6.2 et nous comparons les résultats obtenus en utilisant plusieurs outils qui décident l'équivalence pour un nombre borné de sessions, à savoir SPEC, Akiss et Deepsec. Nous excluons les outils qui ne décident pas l'équivalence, tels que ProVerif [34] et Type-Equiv [64, 65]. Lorsqu'ils répondent, ces outils sont généralement beaucoup plus rapides que tous les outils bornés. Cependant, la terminaison de ProVerif n'est pas garantie, et il peut trouver de fausses attaques. De son côté, Type-Equiv n'est pas toujours capable de conclure.

Nos expériences ont été exécutées chacune sur un processeur Intel 3.1 Xeon, sur une machine qui dispose de 40 cœurs et de 378Go de RAM. Pour chacune de ces expériences, la mémoire a été limitée à 128Go et le temps d'exécution à 24h. Nous indiquons un dépassement de la mémoire par

	Conformité avec tuples	Conformité avec marques
Denning-Sacco	✓	✓
Needham-Schroeder Symétrique	✓	✓
Wide Mouth Frog	✓	✓
Yahalom-Paulson	×	✓
Otway-Rees	✓	✓
Passive authentication	✓	✓
Active authentication	✓	✓
Needham-Schroeder-Lowe	×	✓
Denning-Sacco avec signature	✓	✓

FIGURE 6.1 – Conformité des protocoles de la section 6.2 à un système de types structuré.

MO et un dépassement du temps par TO.

Deux outils (Deepsec [46] et Akiss [44]) permettent de bénéficier du parallélisme sur un grand nombre de cœurs, tandis que SPEC [99] ne se parallélise pas. SAT-Equiv, de son côté, procède par double inclusion, et il serait donc très facile de bénéficier du parallélisme en le lançant sur deux cœurs distincts. Dans la mesure où nous nous intéressons principalement aux complexités asymptotiques, l'utilisation de plusieurs cœurs divise seulement les temps par un facteur constant qui ne change pas fondamentalement l'interprétation des résultats. De plus, lancer les outils en parallèle limite le nombre d'analyse qu'il est possible de réaliser dans un temps défini. Nous avons donc exclu complètement de recourir au parallélisme.

Pour chaque protocole et chaque outil, le nombre de sessions a été progressivement augmenté jusqu'à atteindre un dépassement des limites en mémoire ou en temps, ou bien à dépasser tous les autres outils par un facteur deux à trois en nombre de sessions.

6.4.1 Outils

Cette section présente les trois outils avec lesquels nous allons effectuer la comparaison. Chacun d'entre eux dispose d'une sémantique particulière, et tous permettent de modéliser une algèbre de processus plus large que celle de SAT-Equiv (en particulier, aucun n'est restreint aux protocoles simples ou conformes).

SPEC modélise un ensemble fixé de primitives cryptographiques [99], qui contient toutes les primitives classiques (chiffrements symétrique et asymétrique, paire, signatures et fonctions de hachage). La procédure est correcte et complète par rapport à la bissimulation ouverte, qui est strictement plus forte que l'équivalence de trace [98], et sa terminaison a été démontrée. Nous avons utilisé la version 0.3 [4] (telle qu'au 19 janvier 2017), qui autorise les primitives asymétriques mais n'a pas été formellement démontrée.

Akiss implémente la procédure décrite par Chadha, Ciobâcă et Kremer [44]. Cet outil laisse l'utilisateur définir la théorie équationnelle à l'intérieur d'une large classe, qui comprend toutes les primitives classiques. La procédure de décision s'appuie sur les clauses de Horn et permet de bénéficier du parallélisme. Akiss réalise une surapproximation de l'équivalence de traces qui coïncide avec elle pour la classe des protocoles simples. La terminaison d'Akiss a été démontrée pour l'ensemble de primitives que nous utilisons [43]. Les techniques de réduction d'ordre partiel ont été adaptées à l'outil afin d'améliorer ses performances sur la classe des protocoles déterministes. Nous utilisons la version qui était disponible sur le dépôt git le 24 octobre 2017. Ce dépôt peut être trouvé à partir de la page de l'outil [1].

Deepsec est l'outil le plus récent. Il subsume son prédécesseur, APTE, qui a été exclu de notre comparaison pour cette raison. Deepsec [46] utilise les clauses de Horn, et sa procédure de décision consiste à construire un arbre de partition à partir de deux protocoles. Cet outil peut également profiter du parallélisme, et permet de modéliser une théorie équationnelle définie par l'utilisateur, et il termine dans tous les cas. Nous utilisons la version de l'outil telle qu'elle se trouvait sur le dépôt git [2] le 22 avril 2018.

6.4.2 Résultats

	Spec	Akiss	Deepsec	Sat-Eq	
Denning-Sacco	7	10	35	> 210	(4h)
Needham-Schroeder sym	6	6	21	94^*	(20h30)
Wide Mouth Frog	7	12	28	> 210	(6min)
Yahalom-Paulson	6	6	12	> 28	(7h)
Otway-Rees	6	6	14	> 42	(30min)
Passive Authentication	6	8	46	> 400	(98s)
Active Authentication	6	8	50	> 400	(78s)
Needham-Schroeder-Lowe	4	6	16	> 64	(11min)
Denning-Sacco signature	8	8	18	> 64	(100s)

(*) Voir section 6.5

FIGURE 6.2 – Comparaison de SAT-Equiv avec les autres outils. Nous indiquons le nombre de sessions pour lequel l'outil échoue, soit parce qu'il dépasse les limites de ressources en temps ou en mémoire, soit pour d'autres raisons. Quand les limites de l'outil n'ont pas été atteintes, nous écrivons $> k$ pour indiquer que l'outil peut analyser plus de k sessions, et nous donnons le temps d'analyse pour k sessions.

Cette section expose les résultats des expériences. La figure 6.2 synthétise les résultats pour tous les protocoles et tous les outils. Dans chaque case, nous indiquons : soit une borne supérieure sur le nombre de sessions que l'outil est capable de traiter, c'est-à-dire le premier nombre de sessions testé pour lequel l'outil dépasse les limites de ressources (24h, 128Go et la taille de la pile d'appels de fonction) sans conclure ; soit, dans le cas de SAT-Equiv, le nombre maximal de sessions testé sans dépasser les limites, ainsi que le temps de calcul pour ce nombre de sessions. Le scénario maximal pour le protocole de Needham-Schroeder symétrique est particulier, car certains processus ne sont intégrés que partiellement. Plus de détails seront fournis à ce sujet dans la section suivante. La suite de cette section contient également le détail des résultats pour chaque protocole. Pour SAT-Equiv, nous indiquons également la borne théorique obtenue grâce au théorème 5.1. Lorsque cette borne n'est pas identique pour les deux inclusions, nous n'avons conservé que la plus élevée. Sur tous ces exemples, SAT-Equiv est beaucoup plus efficace que les trois autres outils, y compris Deepsec. Dans le pire des cas, qui correspond au protocole de Yahalom-Paulson, SAT-Equiv met 143s à répondre quand Deepsec dépasse la limite de temps ; dans le meilleur des cas (*Wide Mouth Frog*) SAT-Equiv répond quasi-instantanément (en 400ms) pour 28 sessions, alors que Deepsec n'arrive pas à répondre en 24h. Rappelons tout de même que Deepsec traite une classe de protocoles beaucoup plus étendue (sans aucune hypothèse ni de déterminisme, ni de conformité, et une algèbre de processus plus large, qui autorise par exemple les branches *else*).

Dans tous les tableaux qui suivent, TO (pour *Time Out*) représente un dépassement de la limite de 24h, MO (pour *Memory Out*) signifie que plus de 128 Go de RAM ont été utilisés, et SO (pour *Stack Overflow*) indique que la taille de la pile d'appels de fonction a été dépassée. Le tableau suivant présente les résultats pour le protocole de Denning-Sacco.

DS	Spec	Akiss	Deepsec	Sat-Eq	
3	12s	80ms	<0.01s	70ms	42
6	5h	9s	<0.01s	100ms	64
7	MO	75s	<0.01s	200ms	74
10		SO	0.01s	300ms	114
12			0.04s	400ms	134
14			0.2s	500ms	152
21			18s	1.3s	216
28			25m	3s	280
35			TO	6s	344
42				10s	408
91				4m15s	856
140				35m	1304
210				4h20m	1944

Le tableau suivant expose les résultats pour le protocole de Needham-Schroeder avec le chiffrement symétrique. Les scénarios 47 af. et 94 af. (pour *affiné*) sont des scénarios spéciaux, où certaines sessions sont interrompues avant leur terme. Plus d'informations seront données dans la section suivante.

NSS	Spec	Akiss	Deepsec	Sat-Eq	
3	1m	4s	<10ms	80ms	50
6	MO	TO	10ms	400ms	88
7			50ms	1.2s	98
10			600ms	3s	132
12			7s	4s	155
14			120s	11s	176
21			TO	51s	346
28				178s	452
47 af.				48m	527
47				129m	759
94 af.				20h30m	1024
94				MO	1489

Les résultats pour le protocole *Wide Mouth Frog* sont présentés dans le tableau suivant.

WMF	Spec	Akiss	Deepsec	Sat-Eq	
3	6s	40ms	<10ms	10ms	29
6	58m	1.6s	<10ms	20ms	42
7	TO	5s	<10ms	70ms	47
10		8m30s	60ms	100ms	60
12		SO	440ms	200ms	69
14			5s	350ms	78
21			21m	200ms	106
28			TO	400ms	137
35				600ms	168
70				5s	323
140				55s	633
210				6m	943

Les résultats pour le protocole de Yahalom-Paulson sont présentés dans le tableau suivant.

YP	Spec	Akiss	Deepsec	Sat-Eq	
3	23m	7s	<10ms	400ms	73
6	MO	TO	900ms	5s	122
7			6s	17s	136
10			85m	63s	185
12			TO	143s	214
14				6m	248
21				155m	360
28				7h	472

Le tableau suivant expose les résultats pour le protocole d'Otway-Rees. Pour SPEC, les tuples ont été modélisés comme des paires chaînées à gauche pour éviter l'attaque décrite dans la section 6.2.

OR	Spec	Akiss	Deepsec	Sat-Eq	
3	20min	38s	50ms	140ms	39
6	MO	TO	60ms	520ms	62
7			220ms	1s	70
10			28s	2s	93
12			350s	4s	106
14			TO	11s	124
21				51s	178
42				30min	340

Les résultats pour les protocoles *Passive Authentication* et *Active Authentication* sont donnés dans les deux tableaux ci-dessous.

PA	Spec	Akiss	Deepsec	Sat-Eq	
2	3s	100ms	<10ms	10ms	27
4	14m	4s	10ms	30ms	43
6	MO	10m	20ms	60ms	59
8		TO	20ms	90ms	75
10			80ms	100ms	91
20			3s	300ms	171
40			3h20m	800ms	332
46			TO	1s	380
50				1.1s	412
60				1.7s	492
120				6s	972
200				20s	1612
400				98s	3212

AA	Spec	Akiss	Deepsec	Sat-Eq	
2	3s	100ms	<10ms	10ms	37
4	15m	3s	<10ms	40ms	61
6	MO	5m	20ms	70ms	85
8		SO	20ms	70ms	109
10			60ms	100ms	133
20			2s	300ms	253
40			2h30m	800ms	493
46			23h50m	900ms	565
50			TO	1.1s	613
120				6s	1453
200				18s	2413
400				78s	4813

Le tableau suivant présente les résultats pour le protocole de Needham-Schroeder-Lowe.

NSL	Spec	Akiss	Deepsec	Sat-Eq	
2	11s	40ms	10ms	20ms	27
4	MO	2s	10ms	40ms	43
6		SO	120ms	200ms	59
8			7s	500ms	75
12			58m	0.9s	107
16			TO	3s	139
32				35s	267
48				162s	395
60				6m	491
64				11m	523

Enfin, les résultats pour le protocole de Denning-Sacco avec signature sont donnés dans le tableau suivant.

DS sig.	Spec	Akiss	Deepsec	Sat-Eq	
2	1.5s	60ms	<10ms	0.01s	39
4	4m	1.5s	<10ms	0.02s	55
8	MO	SO	240ms	0.2s	87
16			4h50m	0.9s	151
18			TO	1.1s	167
20				1.2s	183
24				3s	215
32				8s	279
48				66s	407
64				100s	535

6.5 Un nombre arbitraire de sessions

Quoique l'équivalence soit indécidable en général pour un nombre non borné de sessions, Chrétien, Cortier et Delaune [53] ont montré un résultat de décidabilité pour des protocoles conformes dont le *graphe de dépendance* est acyclique. Intuitivement, ce graphe de dépendance permet de représenter la manière dont un message à recevoir peut être construit, et donc dépendre, de messages envoyés, de sorte qu'il contient nécessairement un témoin minimal d'attaque, s'il existe. Ce résultat permet à la fois de démontrer la décidabilité dès lors que le graphe est acyclique, et par une analyse fine des protocoles de Denning-Sacco et de Needham-Schroeder avec chiffrement symétrique, de déduire qu'il est suffisant d'analyser respectivement 42 et 94 sessions. Grâce à l'efficacité de SAT-Equiv, nous pouvons facilement vérifier 42 sessions de Denning-Sacco (en 10s). Nous pouvons donc en déduire, d'après le résultat de décidabilité [53], que le protocole est sûr, y compris lorsque les sessions considérées sont répliquées un nombre arbitraire de fois. Le cas du protocole de Needham-Schroeder nécessite plus de travail, puisque SAT-Equiv n'est pas capable d'analyser autant de sessions. Cependant, nous pouvons lire sur le graphe de dépendance qu'il n'est pas nécessaire d'analyser 94 sessions complètes : certaines sessions peuvent être tronquées avant la fin, puisqu'une attaque minimale n'utilisera qu'une partie du processus, par exemple la première étape. SAT-Equiv est capable de démontrer l'équivalence pour ces 94 sessions affinées, en 20h30min. Nous en déduisons donc que ce protocole reste sûr y compris lorsque les sessions considérées sont répliquées un nombre arbitraire de fois.

6.6 Conclusion

Ce chapitre a présenté l'outil SAT-Equiv qui implémente la traduction et l'algorithme du graphe de planification, tels qu'ils ont été décrits dans les deux chapitres précédents. La comparaison avec les autres outils montre que dans la classe de protocole considérée ici, notre démarche mène à un traitement très efficace de l'équivalence, ce qui encourage à continuer d'améliorer l'outil. Pour ce qui est ses performances, l'adaptation de la réduction d'ordre partiel [27] représente une possibilité. Cependant, dans la mesure où SAT-Equiv ne calcule pas tous les entrelacements, l'aspect le plus intéressant consiste à regrouper des ensembles d'émissions ou de réceptions, et les progrès à attendre de cette méthode sont relativement limités. Une autre voie d'amélioration superficielle concerne l'interface : d'une part, il serait utile de refondre complètement le langage d'entrée pour le rapprocher des formats plus classiques de Deepsec ou ProVerif ; d'autre part, la description des témoins d'attaque gagnerait à être rendue plus explicite.

Une extension plus profonde consisterait à calculer automatiquement le système de type structuré le plus fin auquel le protocole est conforme. En effet, ce système de types s'obtient comme solution d'un problème d'unification : il faut garantir que les sous-termes chiffrés unifiables aient le même type ; si nous considérons les types des variables comme des variables de type, il s'agit donc d'unifier les types des sous-termes chiffrés. Lorsque le problème d'unification n'a pas de solution, le protocole n'est conforme à aucun système de types structuré. Enfin, si nous pouvions établir automatiquement la borne de [53], il nous serait possible, à partir d'une spécification du protocole, de calculer le scénario désiré afin d'obtenir une preuve pour un nombre arbitraire de sessions. De plus, un travail d'optimisation de cette borne est possible. Ces résultats, ainsi que leur implémentation dans SAT-Equiv, représentent une perspective de recherche.

Conclusion

La structure de cette thèse repose sur trois étapes majeures. Dans un premier temps, après avoir exposé le modèle, au chapitre 1, nous avons démontré des résultats de petites attaques, qui permettent de restreindre l'espace de recherche pour les témoins de non-inclusion. Nous avons commencé par donner des résultats de typage, au chapitre 2, pour l'accessibilité et pour l'équivalence, avant de réduire le nombre de constantes nécessaires à l'attaquant indépendamment du protocole, dans le chapitre 3. Cette étape culmine avec le théorème 3.1, qui constitue le principal énoncé nécessaire pour la suite. Dans un second temps, nous avons présenté les problèmes de planification (chapitre 4), ainsi que leur résolution à travers le graphe de planification et le codage comme problème SAT. La traduction de l'équivalence de protocoles dans le langage de la planification a été abordée dans le chapitre 5. De cette manière, nous sommes parvenus jusqu'au théorème 5.1, qui démontre la correction de la traduction dans le langage de la planification tout en énonçant une borne sur la longueur des plans. Dans un dernier temps, nous avons présenté l'implémentation de la démarche dans SAT-Equiv, et l'évaluation de la méthode par la comparaison avec les outils similaires SPEC, Akiss et Deepsec. D'abors, nous reviendrons plus en détail sur chacune de ces étapes, puis nous conclurons en proposant des perspectives.

Résumé

Cette section résume les résultats présentés, à commencer par les résultats de petites attaques, puis la traduction de l'équivalence de trace en termes de planification. Elle s'achève par l'évaluation de la démarche.

Petites attaques. L'efficacité de la démarche complète repose en grande partie sur ces résultats. Nous tentons d'abord de caractériser un témoin symbolique minimal (en nombre d'unifications) à travers le résultat de typage, puis parmi les instances de ces témoins nous recherchons ceux qui sont minimaux (en nombre de constantes). Plus précisément, nous avons d'abord établi que, pour des protocoles déterministes et conformes, il existe une attaque si, et seulement si, il existe une attaque *bien typée*, pour des propriétés d'accessibilité (théorème 2.1) et d'équivalence (théorème 2.2). Concernant l'accessibilité, le degré de généralité du seul autre résultat [11] à porter sur une théorie paramétrique est incomparable avec le nôtre ; pour ce qui est de l'équivalence, nous proposons le premier résultat pour une telle théorie paramétrique, contenant toutes les primitives classiques. Le chapitre 6 montre que la classe de protocoles considérée contient des exemples intéressants issus de la littérature. Ensuite, nous avons prouvé un corollaire de ce résultat : l'attaquant peut utiliser une trace bien typée qui ne contient que des recettes simples. Enfin, nous avons démontré qu'indépendamment du protocole, il suffit à l'attaquant de connaître trois constantes, en plus de celles qui apparaissent explicitement. Le cadre théorique, et en particulier les restrictions sur les algèbres de termes (sortes, contours et théories quasi-linéaires) et de processus (absence de branche *else*, déterminisme), constituent les principales contraintes pour la suite. Nous nous sommes attachés à discuter ces hypothèses, et à les justifier par des contre-exemples aussi souvent que possible. En particulier, en présence de branches *else* ou de non-déterminisme, aucune borne sur le nombre de constantes de l'attaquant n'est calculable.

Planification et traduction. Pour commencer, nous avons présenté les problèmes de planification tels que nous les utilisons, ainsi que la méthode de résolution reposant sur l'algorithme du graphe de planification. Cet algorithme construit le graphe progressivement, à partir de l'ensemble des faits initiaux. À chaque étape, les règles qui peuvent être déclenchées sont ajoutées au graphe, ainsi que les faits qui découlent de ces règles. Les incompatibilités entre les faits ou les règles sont signalées par une relation d'exclusion mutuelle. L'ensemble des états obtenus dans le graphe surapproxime les états réellement accessibles, ce qui mène à de fausses attaques. Lorsqu'une attaque est trouvée, il est donc nécessaire de la vérifier, ce que nous faisons en codant le problème comme formule SAT. Nous traduisons alors le problème de l'inclusion de trace dans le langage de la planification, d'abord pour un attaquant passif, c'est-à-dire pour l'inclusion statique, puis pour un attaquant actif. Cette traduction repose sur l'utilisation de règles symboliques, qui sont ensuite concrétisées. Dans le cas des règles de réception du protocole, nous utilisons la technique de l'aplatissement, ce qui nous permet de nous passer de règles de synthèse pour la déduction de l'attaquant, puisqu'il suffit de considérer des recettes simples. L'équivalence entre le problème d'origine et sa traduction sont démontrés par le théorème 5.1. Ce résultat prouve également l'existence d'une solution de longueur bornée, ce qui permet d'assurer la terminaison, dans tous les cas, de l'algorithme du graphe de planification. Nous avons également fourni des exemples justifiant la nécessité de cette borne. Les problèmes de planification que nous utilisons comportent une infinité de règles, et nous définissons également un oracle de planification pour extraire, à chaque étape, l'ensemble fini des règles accessibles dans un état donné.

Implémentation et évaluation. La dernière partie de cette thèse, intégralement contenue dans le chapitre 6, a été consacrée à l'implémentation de SAT-Equiv et à l'évaluation de la démarche. Nous avons d'abord présenté l'outil, à travers l'ordre des opérations qu'il réalise et les structures de données les plus critiques. Ensuite, une liste de protocoles tirés de la littérature, et qui peuvent être modélisés comme des protocoles simples, a été présentée. Nous avons étudié la conformité de chacun de ces protocoles, et nous avons ajouté un système de marquage lorsque c'était nécessaire. Enfin, les outils SPEC, Akiss, Deepsec et SAT-Equiv, qui permettent tous de vérifier l'équivalence de protocoles pour un nombre borné de sessions, ont été comparés sur ces exemples. Les autres outils, en particulier Deepsec et Akiss, peuvent considérer beaucoup plus de protocoles que SAT-Equiv, mais sur cet échantillon, SAT-Equiv démontre une efficacité largement supérieure, et traite au moins deux fois plus de sessions pour chaque protocole. Ces performances, associées à un résultat de décidabilité, permettent de réaliser des preuves pour un nombre arbitraire de sessions.

En comparaison de SATMC [17], qui a été publié pour la première fois en 2004 [22, 20], SAT-Equiv est plus récent et n'a pas été utilisé dans la vérification de protocoles d'envergure industrielle. En effet, SATMC a permis de découvrir plusieurs failles [18, 16, 40, 19], grâce à des extensions, au fil des années, à des domaines variés. En particulier, cet outil intègre désormais la logique temporelle linéaire (LTL) [16], et au-delà des protocoles cryptographiques, a également été appliqué à la vérification de la sécurité de processus commerciaux [23, 25] (une synthèse de ces développements pourra être trouvée dans la description de l'outil [17] présentée à TACAS en 2014). Cependant, du côté de SAT-Equiv, une attention spécifique a été portée à la démonstration de l'existence de témoins bien typés dans notre modèle, alors qu'Armando et Compagna s'appuient sur un résultat de Heather, Lowe et Schneider [76] sans jamais garantir que le protocole analysé vérifie les hypothèses du résultat. En d'autres termes, SATMC est très efficace dans la recherche d'attaques, mais ne garantit jamais la sécurité d'un protocole, y compris pour un nombre borné de sessions, sauf à vérifier à la main les hypothèses du résultat de typage [76]. Une démarche à envisager consisterait à tenter de rapprocher les formalismes, afin de disposer d'un seul outil complètement prouvé, valable pour l'équivalence, et disposant du large champ d'application de SATMC.

Perspectives

Dans cette dernière section, nous présentons différentes perspectives d'extension ou d'application des résultats exposés dans cette thèse. D'abord, nous revenons en détail sur les suggestions du chapitre 3. Nous proposons ensuite deux applications du résultat de typage pour l'équivalence (chapitre 2). Enfin, nous présentons une piste pour établir un résultat de typage sans hypothèse de conformité, et nous terminons en proposant une utilisation plus large des solveurs SAT.

Borner les données. Plusieurs résultats [52, 60] reposent sur la technique de preuve exposée dans le chapitre 3. Dans le cadre le plus général [60], il a été démontré que pour tout protocole déterministe, il existe une borne que l'on peut calculer directement à partir de la théorie à constructeurs considérée, indépendamment du protocole. Ce modèle autorise donc une large classe de règles de réécriture, qui permettent par exemple de modéliser des disjonctions ($x = x'$ ou $y = y'$) à l'aide de paires critiques. Cependant, la borne n'est pas optimale, même dans des cas relativement simples. Une première étape consisterait donc à l'améliorer. Il s'agit de trouver le nombre minimal de constantes nécessaires pour conserver un certain nombre de différences. Or, un ensemble de n constantes différentes $\{c_1, \ldots, c_n\}$ permet d'obtenir $n(n-1)/2$ différences $c_i \neq c_j$ (pour $i < j$), ce qui revient à dire que le nombre de constantes nécessaires est de l'ordre de $\sqrt{d}$, où d est le nombre de différences à préserver (pour le moment, l'ordre de grandeur est de d constantes, puisque nous en conservons une pour chaque différence à préserver). Un travail supplémentaire devrait permettre de fournir une présentation unifiée entre les différents résultats. Par exemple, le résultat qui porte sur le nombre d'agents utilise des symboles privés pour éviter la confusion entre agents honnêtes, agents malhonnêtes, constantes et noms (afin d'éviter que l'attaquant ne puisse se servir de constantes pour remplacer les identités des agents), mais il serait sans doute plus naturel, et plus uniforme avec le résultat sur les constantes, d'utiliser un système de types pour représenter cette contrainte. Un résultat unifié fournirait naturellement une borne commune, du type $n_a + n_c + n_n \leqslant 3$, où n_a est le nombre d'agents, n_c le nombre de constantes, et n_n le nombre d'instance de chaque nonce, ce qui améliorerait l'efficacité des procédures.

Applications du typage. L'une des applications déjà évoquées consiste à étendre aux primitives asymétriques le résultat de décidabilité [53], qui permet de décider l'équivalence de trace pour un nombre arbitraire de sessions à partir d'une preuve pour un nombre fixé (voir section 6.5). Une telle extension de ce résultat permettrait également de revenir sur la borne obtenue afin de la calculer le plus finement possible. En dehors de la recherche de classes décidables, les résultats de typage ont également été utilisés pour démontrer que la composition de protocoles ne crée pas d'attaques lorsque les protocoles sont prouvés sûrs, autrement dit, que lorsque P et P' sont sûrs, la composition parallèle $(P \mid P')$ est sûre également. De tels résultats existent aussi bien pour l'accessibilité [63, 55, 11] que pour l'équivalence [13]. Il serait possible de montrer, à partir de notre résultat de typage, que pour des propriétés d'équivalence, et pour une classe importante de protocoles (déterministes et conformes) et de primitives, la composition de protocoles est sûre pourvu que les protocoles le soient, et qu'il n'y ait pas de confusion entre les types des deux protocoles.

Extension du typage. Dans le chapitre 2, le résultat technique principal, à savoir la proposition 2.2, démontre que, pour vérifier l'équivalence, il suffit de considérer les exécutions qui n'unifient que des sous-termes chiffrés. Ce résultat est valable sans hypothèse de conformité. Au-delà de la possibilité d'en déduire que l'équivalence est décidable pour un nombre borné de sessions, ce résultat devrait également permettre d'établir un résultat de typage pour tout protocole. Plus précisément, puisque nous savons qu'il suffit de considérer des exécutions qui unifient des sous-termes chiffrés, nous connaissons la forme des termes qui peuvent s'unifier avec chaque variable : par exemple, si les termes dont nous disposons sont $\mathsf{senc}(x, a)$, $\mathsf{senc}(\mathsf{senc}(y, a), a)$ et $\mathsf{senc}(y, a)$ (avec x, y des variables et a une constante), nous savons que la variable x peut s'unifier exclusivement avec $\mathsf{senc}(y, a)$ ou y (et pas avec $\mathsf{aenc}(y, \mathsf{pub}(a))$ ou n'importe quel autre terme). Il n'existe pas de système de type structuré qui permette d'exprimer ces restrictions, puisque $\delta(x) = \delta(y)$ et $\delta(x) = \delta(\mathsf{senc}(y, a))$ imposent $\delta(y) = \mathsf{senc}(\delta(y), \delta(a))$. Une solution consiste à étendre la notion de systèmes de types, et à décrire le type de y par la grammaire $\tau_y = \tau_0 \mid \mathsf{senc}(\tau_y, \tau_a)$ où τ_0 est un type de base et $\tau_a = \delta(a)$. Un tel type décrit une infinité de termes, mais permet tout de même de restreindre l'espace de recherche, y compris pour un nombre arbitraire de sessions.

Solveurs SAT et QBF. Les formules SAT permettent de modéliser des propriétés logiques relativement complexes, dont nous pourrions tirer parti pour traiter une algèbre de processus plus étendue : par exemple, le *choix non déterministe* $(+)$ représente une piste d'extension possible. En effet, le choix non-déterministe repose sur le principe que l'attaquant peut choisir l'un ou l'autre chemin, et cette notion de choix se représente naturellement sous forme de formule SAT. Plus généralement, la présence d'une alternance de quantificateurs dans la définition de l'équivalence de protocoles (pour *toute* exécution de P, il *existe* une exécution de Q de même trace) constitue le principal obstacle à l'utilisation de formules SAT dans le cas des protocoles non-déterministes. Une solution naturelle consiste à utiliser des formules QBF (pour *Quantified Boolean Formula*), qui permettent de quantifier les variables. Les solveurs QBF ne bénéficient pas des mêmes performances que les solveurs SAT, mais le cas qui nous occupe ne contient qu'une seule alternance de quantificateurs, ce qui laisse espérer une efficacité raisonnable. Cette extension nécessiterait néanmoins que le résultat de typage soit démontré dans le cadre plus général des protocoles non-déterministes. Il serait alors facile d'obtenir une borne sur le nombre de constantes, à condition d'accepter qu'elle dépende du protocole considéré, puisque nous considérons des processus sans réplication. Si le résultat de typage s'avérait faux dans ce contexte, il resterait possible de transformer SAT-Equiv en un outil de recherche d'attaques, qui ne démontre pas l'équivalence. Nous devrions alors tenter d'exploiter cet outil, puisque la démarche ne pourrait être validée que par la découverte de failles, qui seraient d'autant plus convaincantes qu'elles seraient exploitables.

Bibliographie

[1] Akiss. `http://akiss.gforge.inria.fr`.

[2] Deepsec. `http://deepsec-prover.github.io`.

[3] SAT-Equiv. `https://projects.lsv.ens-cachan.fr/satequiv`.

[4] Spec. `https://github.com/spec-project/SPEC-0.3/`.

[5] Martín Abadi, Bruno Blanchet et Cédric Fournet : The applied pi calculus: mobile values, new names, and secure communication. *Journal of the ACM*, 65(1):1, 2017.

[6] Martín Abadi et Véronique Cortier : Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 367(1):2–32, 2006.

[7] Martín Abadi et Cédric Fournet : Mobile Values, New Names, and Secure Communication. *In Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

[8] Martín Abadi et Andrew Gordon : A calculus for cryptographic protocols: The spi calculus. *In Proceedings of the 4th ACM Conference on Computer and Communications Security (CCS'97)*, pages 36–47, 1997.

[9] Martín Abadi et Phillip Rogaway : Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[10] Ben Adida : Helios: Web-based Open-Audit Voting. *In Proceedings of the 17th USENIX Security Symposium (USENIX Security'08)*, pages 335–348, 2008.

[11] Omar Almousa, Sebastian Mödersheim, Paolo Modesti et Luca Viganò : Typing and Compositionality for Security Protocols: A Generalization to the Geometric Fragment. *In Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*, pages 209–229, 2015.

[12] Roberto Amadio, Denis Lugiez et Vincent Vanackère : On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 1(290):695–740, 2003.

[13] Myrto Arapinis, Vincent Cheval et Stéphanie Delaune : Composing security protocols: from confidentiality to privacy. *In Proceedings of the 4th International Conference on Principles of Security and Trust (POST'15)*, pages 324–343, 2015.

[14] Myrto Arapinis, Tom Chothia, Eike Ritter et Mark Ryan : Analysing Unlinkability and Anonymity Using the Applied Pi Calculus. *In Proceedings of the 23rd Computer Security Foundations Symposium (CSF'10)*, pages 107–121, 2010.

[15] Myrto Arapinis et Marie Duflot : Bounding messages for free in security protocols. *In Proceedings of the 27th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, pages 376–387, 2007.

[16] Alessandro Armando, Roberto Carbone et Luca Compagna : LTL model checking for security protocols. *Journal of Applied Non-Classical Logics*, 19(4):403–429, 2009.

[17] Alessandro Armando, Roberto Carbone et Luca Compagna : SATMC: a SAT-based Model Checker for Security-critical Systems. *In Proceedings of the 20th international Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, pages 31–45, 2014.

[18] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar et Llanos Tobarra : Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based Single Sign-On for Google apps. *In Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE'08)*, pages 1–10, 2008.

[19] Alessandro Armando, Roberto Carbone et Luca Zanetti : Formal modeling and automatic security analysis of two-factor and two-channel authentication protocols. *In Proceedings of the 7th International Conference on Network and System Security (NSS'13)*, pages 728–734, 2013.

[20] Alessandro Armando et Luca Compagna : SATMC: a SAT-based model checker for security protocols. *In Proceedings of the 2004 European Workshop on Logics in Artificial Intelligence (JELIA'04)*, pages 730–733, 2004.

[21] Alessandro Armando et Luca Compagna : SAT-based model-checking for security protocols analysis. *International Journal of Information Security*, 7:3–32, 2008.

[22] Alessandro Armando, Luca Compagna et Pierre Ganty : SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. *In Proceedings of the 12th International Symposium of Formal Methods Europe (FME'03)*, pages 875–893. 2003.

[23] Alessandro Armando et Serena Ponta : Model checking of security-sensitive business processes. *In Proceedings of the 2009 International Workshop on Formal Aspects in Security and Trust (FAST'09)*, pages 66–80, 2009.

[24] Sanjeev Arora et Boaz Barak : *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[25] Wihem Arsac, Luca Compagna, Giancarlo Pellegrino et Serena Ponta : Security validation of business processes via model-checking. *In Proceedings of the 2011 International Symposium on Engineering Secure Software and Systems (ESSoS'11)*, pages 29–42, 2011.

[26] David Baelde, Stéphanie Delaune, Ivan Gazeau et Steve Kremer : Symbolic verification of privacy-type properties for security protocols with XOR. *In Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 234–248, 2017.

[27] David Baelde, Stéphanie Delaune et Lucca Hirschi : Partial Order Reduction for Security Protocols. *In Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, pages 497–510, 2015.

[28] Tomáš Balyo, Marijn Heule et Matti Järvisalo : Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions, 2017.

[29] Gergei Bana et Hubert Comon-Lundh : Towards unconditional soundness: Computationally complete symbolic attacker. *In Proceedings of the 1st International Conference on Principles of Security and Trust (POST'12)*, pages 189–208, 2012.

[30] David Basin, Jannick Dreier et Ralf Sasse : Automated symbolic proofs of observational equivalence. *In Proceedings of the 22nd International Conference on Computer and Communications Security (CCS'15)*, pages 1144–1155, 2015.

[31] Mathieu Baudet : Deciding security of protocols against off-line guessing attacks. *In Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*, pages 16–25, 2005.

[32] Roberto Bayardo, Jr. et Robert Schrag : Using CSP Look-back Techniques to Solve Real-world SAT Instances. *In Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence (AAAI'97/IAAI'97)*, pages 203–208, 1997.

[33] Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub et Jean Zinzindohoue : A messy state of the union: Taming the composite state machines of TLS. *In Proceedings of the 2015 IEEE Symposium on Security and Privacy (S&P'15)* , pages 535–552, 2015.

[34] Bruno Blanchet : An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. *In Proceedings of the 14th IEEE Computer Security Foundations Symposium (CSF'14)*, pages 82–96, Cape Breton, Nova Scotia, Canada, juin 2001. IEEE Computer Society.

[35] Bruno Blanchet : A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, 2008.

[36] Bruno Blanchet : *Vérification automatique de protocoles cryptographiques : modèle formel et modèle calculatoire. Automatic verification of security protocols: formal model and computational model.* Mémoire d'habilitation à diriger des recherches, Université Paris-Dauphine, novembre 2008.

[37] Bruno Blanchet, Martín Abadi et Cédric Fournet : Automated Verification of Selected Equivalences for Security Protocols. *Journal of Logic and Algebraic Programming*, 75(1):3–51, 2008.

[38] Bruno Blanchet et Andreas Podelski : Verification of Cryptographic Protocols: Tagging Enforces Termination. *In Proceedings of the 6th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'03)*, pages 136–152, 2003.

[39] Avrim Blum et Merrick Furst : Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997.

[40] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi et Graham Steel : Attacking and fixing PKCS# 11 security tokens. *In Proceedings of the 17th ACM conference on Computer and Communications Security (CCS'10)*, pages 260–269, 2010.

[41] Tom Bylander : The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[42] Iliano Cervesato, Nancy Durgin, Patrick Lincoln, John Mitchell et Andre Scedrov : A meta-notation for protocol analysis. *In Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pages 55–69, 1999.

[43] Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă et Steve Kremer : Automated verification of equivalence properties of cryptographic protocol. *ACM Transactions on Computational Logic*, 2016. To appear.

[44] Rohit Chadha, Ştefan Ciobâcă et Steve Kremer : Automated verification of equivalence properties of cryptographic protocols. *In Proceedings of the 21th European Symposium on Programming (ESOP'12)*, pages 108–127, 2012.

[45] Vincent Cheval : APTE: an Algorithm for Proving Trace Equivalence. *In Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14)*, pages 587–592, 2014.

[46] Vincent Cheval, Steve Kremer et Itsaka Rakotonirina : DEEPSEC: Deciding Equivalence Properties in Security Protocols - Theory and Practice. *In Proceedings of the 2018 IEEE Symposium on Security and Privacy (S&P'18)*, pages 525–542, 2018.

[47] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch et Mathieu Turuani : *Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In* Proceedings of the 23rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03), pages 124–135, 2003.

[48] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch et Mathieu Turuani : *An NP decision procedure for protocol insecurity with XOR. Theoretical Computer Science*, 338(1-3):247–274, 2005.

[49] Yannick Chevalier et Michaël Rusinowitch : Decidability of equivalence of symbolic derivations. *Journal of Automated Reasoning*, 48(2):263–292, 2012.

[50] Tom Chothia et Vitaliy Smirnov : A traceability attack against e-passports. *In Proceedings of the 2010 International Conference on Financial Cryptography and Data Security (FC'10)*, pages 20–34, 2010.

[51] Rémy Chrétien, Véronique Cortier et Stéphanie Delaune : Typing messages for free in security protocols: the case of equivalence properties. *In Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, pages 372–386, 2014.

[52] Rémy Chrétien, Véronique Cortier et Stéphanie Delaune : Checking trace equivalence: How to get rid of nonces? *In Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15)*, Vienna, Austria, 2015.

[53] Rémy Chrétien, Véronique Cortier et Stéphanie Delaune : Decidability of trace equivalence for protocols with nonces. *In Proceedings of the 28th IEEE Computer Security Foundations Symposium (CSF'15)*, pages 170–184, 2015.

[54] Rémy Chrétien, Véronique Cortier et Stéphanie Delaune : From security protocols to pushdown automata. *ACM Transactions on Computational Logic*, 17(1 :3), 2015.

[55] Ştefan Ciobâcă et Véronique Cortier : Protocol composition for arbitrary primitives. *In Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 322–336, 2010.

[56] Ştefan Ciobâcă, Stéphanie Delaune et Steve Kremer : Computing knowledge in security protocols under convergent equational theories. *In Proceedings of the 22nd International Conference on Automated Deduction (CADE'09)*, pages 355–370, 2009.

[57] John Clark et Jeremy Jacob : A survey of authentication protocol literature: Version 1.0, 1997.

[58] Hubert Comon-Lundh et Véronique Cortier : Security Properties: Two Agents are Sufficient. *Science of Computer Programming*, 50(1-3):51–71, 2004.

[59] Bruno Conchinha, David Basin et Carlos Caleiro : Efficient decision procedures for message deducibility and static equivalence. *In Proceedings of the 2010 International Workshop on Formal Aspects in Security and Trust (FAST'10)*, pages 34–49, 2010.

[60] Véronique Cortier, Antoine Dallon et Stéphanie Delaune : Bounding the number of agents, for equivalence too. *In Proceedings of the 5th International Conference on Principles of Security and Trust (POST'16)*, pages 211–232, 2016.

[61] Véronique Cortier, Antoine Dallon et Stéphanie Delaune : SAT-Equiv: an efficient tool for equivalence properties. *In Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 481–494, 2017.

[62] Véronique Cortier, Antoine Dallon et Stéphanie Delaune : Efficiently deciding equivalence for standard primitives and phases. *In Proceedings of the 23rd European Symposium on Research in Computer Security (ESORICS'18)*, 2018. À paraître.

[63] Véronique Cortier et Stéphanie Delaune : Safely Composing Security Protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.

[64] Véronique Cortier, Niklas Grimm, Joseph Lallemand et Matteo Maffei : A type system for privacy properties. *In Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS'17)*, pages 409–423, 2017.

[65] Véronique Cortier, Niklas Grimm, Joseph Lallemand et Matteo Maffei : Equivalence properties by typing in cryptographic branching protocols. *In Proceedings of the 7th International Conference on Principles of Security and Trust (POST'18)*, pages 160–187, 2018.

[66] Véronique Cortier et Cyrille Wiedling : A formal analysis of the Norwegian E-voting protocol. *Journal of Computer Security*, 25(15777):21–57, 2017.

[67] Whitfield Diffie et Martin Hellman : New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[68] Danni Dolev et Andrew Yao : On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[69] Nancy Durgin, Patrick Lincoln, John Mitchell et Andre Scedrov : Undecidability of bounded security protocols. *In Proceedings of the 1999 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[70] Nancy Durgin, Patrick Lincoln, John Mitchell et Andre Scedrov : Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[71] Niklas Een et Niklas Sörensson : An Extensible SAT-solver. *In Proceedings of the 2003 International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.

[72] Michael Ernst, Todd Millstein et Daniel Weld : Automatic SAT-compilation of planning problems. *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1169–1176, 1997.

[73] Santiago Escobar, Catherine Meadows et José Meseguer : A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.

[74] Javier Fábrega, Jonathan Herzog et Joshua Guttman : Strand spaces: Why is a security protocol correct? *In Proceedings of the 1998 IEEE Symposium on Security and Privacy (S&P'98)*, pages 160–171, 1998.

[75] Richard Fikes et Nils Nilsson : STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

[76] James Heather, Gavin Lowe et Steve Schneider : How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[77] Nevin Heintze et Doug Tygar : A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, 1996.

[78] Andreas Hess et Sebastian Mödersheim : Formalizing and Proving a Typing Result for Security Protocols in Isabelle/HOL. *In Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*, pages 451–463, 2017.

[79] Lucca Hirschi, David Baelde et Stéphanie Delaune : A method for verifying privacy-type properties: the unbounded case. *In Proceedings of the 2016 IEEE Symposium on Security and Privacy (S&P'16)*, pages 564–581, 2016.

[80] David Kahn : *The Codebreakers: a Comprehensive History of Secret Communication from Ancient Times to the Internet, Revised and Updated.* Scribner, 1996.

[81] Henry Kautz et Bart Selman : Unifying SAT-based and graph-based planning. *In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325, 1999.

[82] Henry Kautz, Bart Selman *et al.* : Planning as Satisfiability. *In Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

[83] Gavin Lowe : An Attack on the Needham-Schroeder Public-Key Authentication Protocol. *Information Processing Letters*, 56(3), 1995.

[84] Gavin Lowe : Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Software - Concepts and Tools*, 17(3):93–102, 1996.

[85] João Marques-Silva et Karem Sakallah : GRASP - a New Search Algorithm for Satisfiability. *In Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'96)*, pages 220–227, 1996.

[86] Simon Meier, Benedikt Schmidt, Cas Cremers et David Basin : The TAMARIN Prover for the Symbolic Analysis of Security Protocols. *In Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*, pages 696–701, 2013.

[87] Roger Needham et Michael Schroeder : Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[88] Technical Advisory Group on Machine Readable Travel Documents (TAG/MRTD) : Revision of Doc 9303 - Machine Readable Travel Documents. Technical report, International Civil Aviation Organization, Montréal, 2014. TAG/MRTD/22-WP/3.

[89] David Otway et Owen Rees : Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[90] Emil Post : A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

[91] Ramaswamy Ramanujam et S. P. Suresh : Tagging Makes Secrecy Decidable with Unbounded Nonces as Well. *In Proceedings of the 23rd Conference of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, pages 363–374, 2003.

[92] Jussi Rintanen, Keijo Heljanko et Ilkka Niemelä : Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[93] Ronald Rivest, Adi Shamir et Leonard Adleman : A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[94] Michaël Rusinowitch et Mathieu Turuani : Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, April 2003.

[95] Sonia Santiago, Santiago Escobar, Catherine Meadows et José Meseguer : A Formal Definition of Protocol Indistinguishability and Its Verification Using Maude-NPA. *In Proceedings of the 10th Security and Trust Management Workshop (STM'14)*, pages 162–177, 2014.

[96] Adi Shamir : How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[97] Claude Shannon : Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[98] Alwen Tiu : A Trace Based Bisimulation for the Spi Calculus. *In Proceedings of the 5th Asian Symposium for Programming Languages and Systems (APLAS'07)*, pages 367–382, 2007.

[99] Alwen Tiu et Jeremy Dawson : Automating open bisimulation checking for the spi-calculus. *In Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10)*, pages 307–321, 2010.

Index

A Démonstrations pour l'aplatissement

Cette annexe contient les preuves de l'aplatissement qui ont été laissées de côté dans le chapitre 5. Les démonstrations formelles pour la traduction complète se trouvent dans la section 5.3.5.

Commençons par démontrer le lemme 5.9.

Lemme 5.9. *Soit* $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ *une règle. Soit* σ *une substitution telle que* $r\sigma$ *est close et* $\delta(x\sigma) \preccurlyeq \delta(x)$ *pour chaque* $x \in vars_{\mathsf{left}}(r)$*. Soit* C *un contexte constructeur tel que* $u\sigma = C[u_1, \ldots, u_n]$ *et* $v\sigma = C[v_1, \ldots, v_n]$*. Il existe* $r' \in \mathsf{Flat}(r)$ *:*

$$r' = \mathsf{Pre}', \mathsf{att}(u'_1, v'_1), \ldots, \mathsf{att}(u'_n, v'_n) \to \mathsf{Add}'; \mathsf{Del}'$$

et σ' *une substitution telle que :*

1. $r'\sigma'$ *est une règle close.*
2. $\delta(x\sigma') \preccurlyeq \delta(x)$ *pour chaque* $x \in vars_{\mathsf{left}}(r')$ *;*
3. $(\mathsf{Pre}', \mathsf{Add}', \mathsf{Del}')\sigma' = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})\sigma$ *;*
4. $\mathsf{att}(u, v)\sigma = \mathsf{att}(C[u'_1, \ldots, u'_n], C[v'_1, \ldots, v'_n])\sigma'$.

Démonstration. Établissons d'abord le fait suivant :

Fait. Soit $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ une règle. Soit σ une substitution telle que $r\sigma$ est close et $\delta(x\sigma) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r)$. Soit $\mathsf{f} \in \Sigma_c$ tel que $u\sigma = \mathsf{f}(u_1, u_2, \ldots, u_n)$ et $v\sigma = \mathsf{f}(v_1, v_2, \ldots, v_n)$ pour certains termes $u_1, u_2, \ldots, u_n,\ v_1, v_2, \ldots, v_n$. Alors u est décomposable, et $r_1 = \mathsf{decom}(r, \mathsf{att}(u, v))$ est de la forme suivante :

$$r_1 = \mathsf{Pre}_1, \mathsf{att}(u'_1, v'_1), \mathsf{att}(u'_2, v'_2), \ldots, \mathsf{att}(u'_n, v'_n) \to \mathsf{Add}_1; \mathsf{Del}_1$$

De plus, il existe une substitution σ_1 telle que :

1. $r_1\sigma_1$ est close.
2. $\delta(x\sigma_1) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r_1)$;
3. $(\mathsf{Pre}_1, \mathsf{Add}_1, \mathsf{Del}_1)\sigma_1 = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})\sigma$;
4. $u\sigma = \mathsf{f}(u'_1, u'_2, \ldots, u'_n)\sigma_1$ et $v\sigma = \mathsf{f}(v'_1, v'_2, \ldots, v'_n)\sigma_1$.

Preuve du fait. Pour alléger le formalisme, nous supposons que f est d'arité deux. Le cas général se traite de la même manière. Puisque $u\sigma = \mathsf{f}(u_1, u_2)$ et que $\delta(x\sigma) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r)$, u est décomposable, ce qui permet d'écrire :

$$\mathsf{split}(\mathsf{att}(u, v)) = (\mathsf{f}; \{\mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2)\}; \sigma_{\mathcal{P}}; \sigma_{\mathcal{Q}})$$

avec $\delta(x_1) = \tau_1$, $\delta(x_2) = \tau_2$, $\sigma_{\mathcal{P}} = mgu(u, \mathsf{f}(x_1, x_2))$ est quasiment typée, et $\sigma_{\mathcal{Q}} = mgu(v, \mathsf{f}(y_1, y_2))$. Remarquons que $\sigma_{\mathcal{Q}} \neq \bot$, puisque $v\sigma = \mathsf{f}(v_1, v_2)$. De plus, quand u (resp v) n'est pas une variable, nous supposons, sans perte de généralité, que x_1, x_2 (resp. y_1, y_2) n'apparaissent pas dans $img(\sigma_{\mathcal{P}})$ (resp. $img(\sigma_{\mathcal{Q}})$). Soit $r_1 = \mathsf{decom}(r, \mathsf{att}(u, v))$. Nous avons :

$$r_1 = [\mathsf{Pre}, \mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2) \to \mathsf{Add}; \mathsf{Del}](\sigma_{\mathcal{P}} \uplus \sigma_{\mathcal{Q}})$$

Soit $\sigma^{\mathsf{left}} = \sigma|_{vars_{\mathsf{left}}(r)}$ et $\sigma^{\mathsf{right}} = \sigma|_{vars_{\mathsf{right}}(r)}$. Comme $vars_{\mathsf{left}}(r) \cap vars_{\mathsf{right}}(r) = \emptyset$, nous obtenons $\sigma = \sigma^{\mathsf{left}} \uplus \sigma^{\mathsf{right}}$.

Opérons maintenant une distinction suivant que u (resp. v) est une variable ou non. Dans le cas où u est une variable (appelons-là z_u), nous avons $\sigma_{\mathcal{P}} = \{z_u \mapsto \mathsf{f}(x_1, x_2)\}$, et nous définissons $\sigma_1^{\mathsf{left}} = \sigma^{\mathsf{left}} \uplus \{x_1 \mapsto u_1, x_2 \mapsto u_2\}$. Sinon, nous avons $u = \mathsf{f}(a_1, a_2)$, et $\sigma_{\mathcal{P}} = \{x_1 \mapsto a_1, x_2 \mapsto a_2\}$, et nous définissons $\sigma_1^{\mathsf{left}} = \sigma^{\mathsf{left}}$. Nous procédons similairement pour v. Il reste à montrer que r_1 et $\sigma_1 = \sigma_1^{\mathsf{left}} \uplus \sigma_1^{\mathsf{right}}$, comme définis plus haut, satisfont les conditions du fait. Chaque propriété s'établit séparément.

1. Nous avons $\delta(x\sigma_1) = \delta(x\sigma_1^{\mathsf{left}}) = \delta(x\sigma) \preccurlyeq \delta(x)$ pour chaque $x \in \mathit{vars}_{\mathsf{left}}(r_1) \setminus \{x_1, x_2\}$; et $\delta(x_i\sigma_1) = \delta(x_i\sigma_1^{\mathsf{left}}) = \delta(u_i) = \tau_i \preccurlyeq \delta(x_i)$ pour $i \in \{1, 2\}$. D'où le résultat.
2. Dans le cas où u est une variable, nous avons $(z_u\sigma_{\mathcal{P}})\sigma_1^{\mathsf{left}} = z_u\sigma$, et similairement pour v. Il est donc facile de voir que
$$(\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})(\sigma_{\mathcal{P}} \uplus \sigma_{\mathcal{Q}})\sigma_1 = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})\sigma.$$
3. Nous avons $\mathsf{f}(u_1', u_2')\sigma_1 = \mathsf{f}(x_1\sigma_{\mathcal{P}}, x_2\sigma_{\mathcal{P}})\sigma_1 = u\sigma$, et de même pour v.

Ceci conclut la preuve du fait.

Il reste maintenant à démontrer le résultat principal par récurrence structurelle sur C. L'initialisation, c'est-à-dire le cas où C est vide, est évidente : il suffit de choisir $r' = r$. Supposons maintenant que $C = \mathsf{f}(C_1, C_2, \ldots, C_n)$. Pour alléger le formalisme, nous supposons que f est d'arité 2. La preuve complète s'effectue de la même manière. D'après le fait démontré, nous obtenons $r_1 = \mathsf{decom}(r, \mathsf{att}(u, v))$ de la forme
$$r_1 = \mathsf{Pre}_1, \mathsf{att}(u_1', v_1'), \mathsf{att}(u_2', v_2') \to \mathsf{Add}_1; \mathsf{Del}_1$$
et une substitution σ_1 telle que :

1. $r_1\sigma_1$ est close
2. $\delta(x\sigma_1) \preccurlyeq \delta(x)$ pour chaque $x \in \mathit{vars}_{\mathsf{left}}(r_1)$;
3. $(\mathsf{Pre}_1, \mathsf{Add}_1, \mathsf{Del}_1)\sigma_1 = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})\sigma$;
4. $u\sigma = \mathsf{f}(u_1', u_2')\sigma_1$ et $v\sigma = \mathsf{f}(v_1', v_2')\sigma_1$.

En particulier $C_i[u_1, \ldots, u_n] = u_i'\sigma_1$ et $C_i[v_1, \ldots, v_n] = v_i'\sigma_1$ pour chaque $i \in \{1, 2\}$.
Maintenant écrivons
$$r_1 = \mathsf{Pre}_1', \mathsf{att}(u_2', v_2') \to \mathsf{Add}_1; \mathsf{Del}_1$$
et appliquons l'hypothèse de récurrence avec le contexte C_2 et la substitution σ_1. Il existe donc une règle $r_2 \in \mathsf{Flat}(r_1)$ telle que
$$r_2 = \mathsf{Pre}_2, \mathsf{att}(u_1^2, v_1^2), \ldots, \mathsf{att}(u_n^2, v_n^2) \to \mathsf{Add}_2; \mathsf{Del}_2$$
et une substitution σ_2 telle que

1. $r_2\sigma_2$ est close
2. $\delta(x\sigma_2) \preccurlyeq \delta(x)$ pour chaque $x \in \mathit{vars}_{\mathsf{left}}(r_2)$;
3. $(\mathsf{Pre}_2, \mathsf{Add}_2, \mathsf{Del}_2)\sigma_2 = (\mathsf{Pre}_1', \mathsf{Add}_1, \mathsf{Del}_1)\sigma_1$;
4. $u_2'\sigma_1 = C_2[u_1^2, \ldots, u_n^2]\sigma_2$, et de même $v_2'\sigma_1 = C_2[v_1^2, \ldots, v_n^2]\sigma_2$.

Nous pouvons écrire :
$$r_2 = \mathsf{Pre}_2', \mathsf{att}(u_1'', v_1'') \to \mathsf{Add}_2; \mathsf{Del}_2$$
où $u_1''\sigma_2 = u_1'\sigma_1$ et $v_1''\sigma_2 = v_1'\sigma_1$ et nous appliquons l'hypothèse de récurrence avec le contexte C_1 et la substitution σ_2. Nous obtenons une règle $r_3 \in \mathsf{Flat}(r_2)$ telle que
$$r_3 = \mathsf{Pre}_3, \mathsf{att}(u_1^3, v_1^3), \ldots, \mathsf{att}(u_n^3, v_n^3) \to \mathsf{Add}_3; \mathsf{Del}_3$$
et une substitution σ_3

1. $r_3\sigma_3$ est close
2. $\delta(x\sigma_3) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r_3)$;
3. $(\mathsf{Pre}_3, \mathsf{Add}_3, \mathsf{Del}_3)\sigma_3 = (\mathsf{Pre}'_2, \mathsf{Add}_2, \mathsf{Del}_2)\sigma_2$,
4. $u''_1\sigma_2 = C_1[u^3_1, \ldots, u^3_n]\sigma_3$, et de même $v''_2\sigma_2 = C_1[v^3_1, \ldots, v^3_n]\sigma_3$.

Nous avons $r_3 \in \mathsf{Flat}(r)$ et il reste à vérifier que r_3 et σ_3, comme définis plus haut, satisfont les conditions du lemme. Nous avons $\delta(x\sigma_3) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r_3)$. De plus, nous avons également :

$$\begin{array}{rcl}(\mathsf{Add}_3, \mathsf{Del}_3)\sigma_3 & = & (\mathsf{Add}_2, \mathsf{Del}_2)\sigma_2 \\ & = & (\mathsf{Add}_1, \mathsf{Del}_1)\sigma_1 \\ & = & (\mathsf{Add}, \mathsf{Del})\sigma\end{array}$$

Comme $\mathsf{Pre}_3\sigma_3 = \mathsf{Pre}'_2\sigma_2$, nous avons

$$\mathsf{Pre}_3 = \mathsf{Pre}'_3, \mathsf{att}(u^4_1, v^4_1), \ldots, \mathsf{att}(u^4_n, v^4_n)$$

pour un certain Pre'_3 où $u^4_i\sigma_3 = u^2_i\sigma_2$ et $v^4_i\sigma_3 = v^2_i\sigma_2$ pour chaque $i \in \{1, \ldots, n\}$. D'où :

$$\begin{array}{l}\mathsf{Pre}'_3\sigma_3, \mathsf{att}(u^4_1, v^4_1)\sigma_3, \ldots, \mathsf{att}(u^4_n, v^4_n)\sigma_3, \mathsf{att}(u''_1, v''_1)\sigma_2 \\ = \mathsf{Pre}_3\sigma_3, \mathsf{att}(u''_1, v''_1)\sigma_2 \\ = \mathsf{Pre}'_2\sigma_2, \mathsf{att}(u''_1, v''_1)\sigma_2 \\ = \mathsf{Pre}_2\sigma_2, \mathsf{att}(u^2_1, v^2_1)\sigma_2, \ldots, \mathsf{att}(u^2_n, v^2_n)\sigma_2 \\ = \mathsf{Pre}'_1\sigma_1, \mathsf{att}(u^2_1, v^2_1)\sigma_2, \ldots, \mathsf{att}(u^2_n, v^2_n)\sigma_2 \\ = \mathsf{Pre}_1\sigma_1, \mathsf{att}(u'_1, v'_1)\sigma_1, \mathsf{att}(u^2_1, v^2_1)\sigma_2, \ldots, \mathsf{att}(u^2_n, v^2_n)\sigma_2 \\ = \mathsf{Pre}\,\sigma, \mathsf{att}(u'_1, v'_1)\sigma_1, \mathsf{att}(u^2_1, v^2_1)\sigma_2, \ldots, \mathsf{att}(u^2_n, v^2_n)\sigma_2\end{array}$$

Donc $\mathsf{Pre}'_3\sigma_3 = \mathsf{Pre}\,\sigma$. Pour finir :

$$\begin{array}{rl}u\sigma & = \mathsf{f}(u'_1, u'_2)\sigma_1 \\ & = \mathsf{f}(u''_1\sigma_2, C_2[u^2_1, \ldots, u^2_n]\sigma_2) \\ & = \mathsf{f}(C_1[u^3_1, \ldots, u^3_n], C_2[u^4_1, \ldots, u^4_n])\sigma_3.\end{array}$$

De même, il est possible de montrer que $v\sigma = \mathsf{f}(C_1[v^3_1, \ldots, v^3_n], C_2[v^4_1, \ldots, v^4_n])\sigma_3$, ce qui conclut la preuve. □

Démontrons maintenant le lemme 5.10.

Lemme 5.10. *Soit r une règle de protocole, et $r' \in \mathsf{Flat}(r)$ telle que*

$$r' = \mathsf{state}, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_n, v_n) \to \mathsf{Add}; \mathsf{Del}$$

— *Ou bien* $\mathsf{bad\text{-}flat} \notin \mathsf{Add}$*, et alors il existe un contexte constructeur C et une substitution τ tels que*

$$r\tau = \mathsf{Pre}, \mathsf{att}(C[u_1, \ldots, u_n], C[v_1, \ldots, v_n]) \to \mathsf{Add}; \mathsf{Del}$$

— *Ou bien* $\mathsf{Add} = \mathsf{bad\text{-}flat}$*,* $\mathsf{Del} = \emptyset$ *et il existe une substitution τ, un terme v et deux ensembles* Add_0 *et* Del_0 *tels que* $r\tau = \mathsf{state}, \mathsf{att}(C[u_1, \ldots, u_n], v) \to \mathsf{Add}_0; \mathsf{Del}_0$ *mais v ne s'unifie pas avec C.*

Démonstration. Comme $r' \in \mathsf{Flat}(r)$, il existe une séquence $r_0, \ldots, r_n$ de règles, et une séquence $f_0, \ldots, f_{n-1}$ de faits, telles que $r_0 = r$, $r_n = r'$ et pour chaque $0 \leqslant i \leqslant n-1$, nous avons $r_{i+1} = \mathsf{decom}(r_i, f_i)$. Montrons le résultat par récurrence sur n. L'initialisation, qui correspond à $n = 0$, est évidente. Supposons que nous avons le résultat pour n. Soit

$$r_n = \mathsf{state}_n, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_m, v_m) \to \mathsf{Add}_n; \mathsf{Del}_n.$$

et $r_{n+1} = \mathsf{decom}(r_n, f_n) = \mathsf{Pre}_{n+1} \to \mathsf{Add}_{n+1}; \mathsf{Del}_{n+1}$. Sans perte de généralité, supposons que $f_n = \mathsf{att}(u_m, v_m)$.

Premier cas. $\mathsf{bad\text{-}flat} \notin \mathsf{Add}_{n+1}$. Donc, comme $r_{n+1} = \mathsf{decom}(r_n, f_n)$, $\mathsf{bad\text{-}flat} \notin \mathsf{Add}_n$. Par hypothèse de récurrence, il existe C et τ_n tels que :

$$r\tau_n = \mathsf{state}_n, \mathsf{att}(u_0, v_0) \to \mathsf{Add}_n; \mathsf{Del}_n$$

où $u_0 = C_n[u_1, \ldots, u_m]$ et $v_0 = C_n[v_1, \ldots, v_m]$.

Nous avons $\mathsf{split}(f_n) = (\mathsf{f}, S, \sigma_\mathcal{P}, \sigma_\mathcal{Q})$ et, en supposant, pour alléger le formalisme, que f est d'arité 2, $S = \{\mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2)\}$, $\sigma_\mathcal{P} = mgu(u_m, \mathsf{f}(x_1, x_2))$, et $\sigma_\mathcal{Q} = mgu(v_m, \mathsf{f}(y_1, y_2))$. De plus, puisque $\mathsf{bad\text{-}flat} \notin \mathsf{Add}_{n+1}$, nous avons $\sigma_\mathcal{Q} \neq \bot$. Nous obtenons la règle r_{n+1} :

$$\begin{aligned}(&\mathsf{state}_n, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_{m-1}, v_{m-1}),\\ &\quad \mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2) \to \mathsf{Add}_n; \mathsf{Del}_n)(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q})\end{aligned}$$

Soit $\tau = \tau_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q})$ et $C = C_n[\_, \ldots, \mathsf{f}(\_, \_)]$. Nous avons :

$$\begin{aligned}&r\tau_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q})\\ &= \mathsf{state}_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}), \mathsf{att}(u_0\sigma_\mathcal{P}, v_0\sigma_\mathcal{Q}) \to \mathsf{Add}_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q});\\ &\qquad\qquad \mathsf{Del}_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}).\end{aligned}$$

Il reste seulement à établir que $u_0\sigma_\mathcal{P} = C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}]$ (et de même pour v_0). Nous avons :

$$\begin{aligned}&u_0\sigma_\mathcal{P}\\ &= C_n[u_1, \ldots, u_m]\sigma_\mathcal{P}\\ &= C_n[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, \mathsf{f}(x_1, x_2)\sigma_\mathcal{P}]\\ &= C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}]\end{aligned}$$

Ce qui conclut la preuve de ce cas.

Second cas. $\mathsf{bad\text{-}flat} \in \mathsf{Add}_{n+1}$ mais $\mathsf{bad\text{-}flat} \notin \mathsf{Add}_n$. Par hypothèse de récurrence, il existe C et τ_n tels que :

$$r\tau_n = \mathsf{state}_n, \mathsf{att}(u_0, v_0) \to \mathsf{Add}_n; \mathsf{Del}_n$$

où $u_0 = C_n[u_1, \ldots, u_m]$ et $v_0 = C_n[v_1, \ldots, v_m]$.

Nous avons $\mathsf{split}(f_n) = (\mathsf{f}, S, \sigma_\mathcal{P}, \bot)$ avec $S = \{\mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2)\}$, $\sigma_\mathcal{P} = mgu(u_m, \mathsf{f}(x_1, x_2))$ en supposant que f est d'arité 2 (les autres cas sont similaires). Nous obtenons la règle suivante r_{n+1} :

$$\begin{aligned}(&\mathsf{state}_n, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_{m-1}, v_{m-1}),\\ &\qquad\qquad \mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2) \to \mathsf{bad\text{-}flat})\sigma_\mathcal{P}\end{aligned}$$

Soit $\tau = \tau_n.\sigma_\mathcal{P}$ et $C = C_n[\_, \ldots, \mathsf{f}(\_, \_)]$.

Nous avons

$$r\tau_n\sigma_\mathcal{P} = \mathsf{state}_n\sigma_\mathcal{P}, \mathsf{att}(u_0\sigma_\mathcal{P}, v_0) \to \mathsf{Add}_n\sigma_\mathcal{P}; \mathsf{Del}_n\sigma_\mathcal{P}$$

Il reste à montrer que $u_0\sigma_\mathcal{P} = C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}]$ mais que v_0 ne s'unifie pas avec C. Nous avons :

$$\begin{aligned} & u_0\sigma_\mathcal{P} \\ &= C_n[u_1, \ldots, u_m]\sigma_\mathcal{P} \\ &= C_n[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, \mathsf{f}(x_1, x_2)\sigma_\mathcal{P}] \\ &= C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}] \end{aligned}$$

et $v_0 = C_n[v_1, \ldots, v_m]$, donc si v_0 s'unifie avec C alors v_m s'unifie $\mathsf{f}(\_, \_)$ et donc avec $\mathsf{f}(y_1, y_2)$ comme y_1, y_2 sont des variables. Mais c'est impossible puisque $\sigma_\mathcal{Q} = \bot$. Donc v_0 ne s'unifie pas avec C, ce qui conclut la preuve de ce second cas.

Troisième cas. $\mathsf{bad\text{-}flat} \in \mathsf{Add}_{n+1}$ et $\mathsf{bad\text{-}flat} \in \mathsf{Add}_n$. Par hypothèse de récurrence, il existe C et τ_n tels que :

$$r\tau_n = \mathsf{state}_n, \mathsf{att}(u_0, v_0) \to \mathsf{Add}_0; \mathsf{Del}_0$$

où $u_0 = C_n[u_1, \ldots, u_m]$ et v_0 ne s'unifie pas avec C_n.

Nous avons $\mathsf{split}(f_n) = (\mathsf{f}, S, \sigma_\mathcal{P}, \sigma_\mathcal{Q})$, et en supposant que f est d'arité 2, $S = \{\mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2)\}$, $\sigma_\mathcal{P} = mgu(u_m, \mathsf{f}(x_1, x_2))$ et $\sigma_\mathcal{Q} = mgu(v_m, \mathsf{f}(y_1, y_2))$.

Sous-cas 3.1. $\sigma_\mathcal{Q} \neq \bot$. Nous obtenons la règle suivante r_{n+1} :

$$\begin{aligned} (&\mathsf{state}_n, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_{m-1}, v_{m-1}), \\ &\quad \mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2) \to \mathsf{bad\text{-}flat})(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}) \end{aligned}$$

Soit $\tau = \tau_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q})$ et $C = C_n[\_, \ldots, \mathsf{f}(\_, \_)]$. Nous avons

$$\begin{aligned} & r\tau_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}) \\ &= \mathsf{state}_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}), \mathsf{att}(u_0\sigma_\mathcal{P}, v_0\sigma_\mathcal{Q}) \to \mathsf{Add}_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}); \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{Del}_n(\sigma_\mathcal{P} \uplus \sigma_\mathcal{Q}). \end{aligned}$$

Il reste à montrer que $u_0\sigma_\mathcal{P} = C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}]$ mais que $v_0\sigma_\mathcal{Q}$ ne s'unifie pas avec C. Nous avons :

$$\begin{aligned} & u_0\sigma_\mathcal{P} \\ &= C_n[u_1, \ldots, u_m]\sigma_\mathcal{P} \\ &= C_n[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, \mathsf{f}(x_1, x_2)\sigma_\mathcal{P}] \\ &= C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}] \end{aligned}$$

et $v_0\sigma_\mathcal{Q}$ ne s'unifie pas avec C_n, donc $v_0\sigma_\mathcal{Q}$ ne s'unifie pas avec C, ce qui conclut ce sous-cas.

Sous-cas 3.2. $\sigma_\mathcal{Q} = \bot$. Nous obtenons la règle suivante r_{n+1} :

$$\begin{aligned} (&\mathsf{state}_n, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_{m-1}, v_{m-1}), \\ &\quad \mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2) \to \mathsf{bad\text{-}flat})\sigma_\mathcal{P} \end{aligned}$$

Soit $\tau = \tau_n\sigma_\mathcal{P}$ et $C = C_n[\_, \ldots, \mathsf{f}(\_, \_)]$. Nous avons

$$\begin{aligned} & r\tau_n\sigma_\mathcal{P} \\ &= \mathsf{state}_n\sigma_\mathcal{P}, \mathsf{att}(u_0\sigma_\mathcal{P}, v_0) \to \mathsf{Add}_n\sigma_\mathcal{P}; \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{Del}_n\sigma_\mathcal{P}. \end{aligned}$$

Il reste à montrer que $u_0\sigma_\mathcal{P} = C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}]$ mais que v_0 ne s'unifie pas avec C. Nous avons :

$$\begin{aligned} & u_0\sigma_\mathcal{P} \\ &= C_n[u_1, \ldots, u_m]\sigma_\mathcal{P} \\ &= C_n[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, \mathsf{f}(x_1, x_2)\sigma_\mathcal{P}] \\ &= C[u_1\sigma_\mathcal{P}, \ldots, u_{m-1}\sigma_\mathcal{P}, x_1\sigma_\mathcal{P}, x_2\sigma_\mathcal{P}] \end{aligned}$$

et v_0 ne s'unifie pas avec C_n, donc v_0 ne s'unifie pas avec C, ce qui conclut la preuve de ce sous-cas et donc du lemme. □

Le lemme suivant permet de traiter une seule décomposition, quand $\mathsf{Add} = \{\mathsf{bad}\}$ ou quand la décomposition côté Q ne peut pas suivre celle qui a lieu côté P.

Lemme A.1. *Soit r une règle, $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$. Soit σ une substitution telle que, pour chaque $x \in vars_{\mathsf{left}}(r)$, $x\sigma$ est clos et $\delta(x\sigma) \preccurlyeq \delta(x)$. Soit f un symbole constructeur tel que $u\sigma = \mathsf{f}(u_1, u_2, \ldots, u_n)$. Supposons que $\mathsf{Add} = \{\mathsf{bad}\}$ ou v ne s'unifie pas avec $\mathsf{f}(\_, \_, \ldots, \_)$. Alors u est décomposable. Soit r' définie par*

$$r' = \mathsf{decom}(r, \mathsf{att}(u, v)) = \mathsf{Pre}', \mathsf{att}(u'_1, v'_1), \mathsf{att}(u'_2, v'_2), \ldots, \mathsf{att}(u'_n, v'_n) \to \mathsf{bad}$$

Si $\mathsf{Add} \neq \{\mathsf{bad}\}$ alors il existe une substitution σ' telle que pour tout $x \in vars_{\mathsf{left}}(r)$, $x\sigma'$ est clos et $\delta(x\sigma') \preccurlyeq \delta(x)$, $\mathsf{Pre}'\sigma' = \mathsf{Pre}\sigma$, $\mathsf{f}(u'_1, u'_2, \ldots u'_n)\sigma' = u\sigma$ et $v'_1, v'_2, \ldots, v'_n$ sont des variables distinctes qui n'apparaissent pas ailleurs dans r'.

Si $\mathsf{Add} = \mathsf{bad}$ et $\mathsf{Del} = \emptyset$, le resultat est encore vrai, sauf qu'aucune condition n'est garantie sur $v'_1, v'_2, \ldots, v'_n$ si v s'unifie avec $\mathsf{f}(\_, \_, \ldots, \_)$.

Démonstration. Dans cette preuve, nous supposerons, pour alléger le formalisme, que le symbole f est d'arité 2. Le cas général se traite de la même manière. $u\sigma = \mathsf{f}(u_1, u_2)$ donc ou bien u n'est pas atomique ; ou bien u est une variable et $\delta(u\sigma) \preccurlyeq \delta(u) = \mathsf{f}(\delta(u_1), \delta(u_2))$. Dans tous les cas u est décomposable.

Considérons d'abord le cas où $\mathsf{Add} \neq \mathsf{bad}$. Comme v ne s'unifie pas avec $\mathsf{f}(v_1, v_2)$ pour chaque v_1, v_2,

$$\mathsf{split}(\mathsf{att}(u, v)) = (\mathsf{f}, \{\mathsf{att}(x_1, y_1), \mathsf{att}(x_2, y_2)\}, \sigma_{\mathcal{P}}, \bot)$$

où $\sigma_{\mathcal{P}} = mgu(u, \mathsf{f}(x_1, x_2))$ et $\delta(x_i) = \delta(u_i)$.

Premier cas. u est une variable. Nous avons $\sigma_{\mathcal{P}} = \{u \mapsto \mathsf{f}(x_1, x_2)\}$. Soit la règle r' definie par :

$$r' = \mathsf{decom}(r, \mathsf{att}(u, v)) = \mathsf{Pre}\sigma_{\mathcal{P}}, \mathsf{att}(x_1\sigma_{\mathcal{P}}, y_1), \mathsf{att}(x_2\sigma_{\mathcal{P}}, y_2) \to \mathsf{bad}$$

et la substitution σ' définie par $\sigma' = \{x_1 \mapsto u_1; x_2 \mapsto u_2\} \cup \sigma\}$

Comme $x_1, x_2 \notin dom(\sigma)$, σ' est bien définie. Nous avons $\delta(x\sigma') \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(\sigma')$. De plus, $\mathsf{Pre}'\sigma' = \mathsf{Pre}\sigma_{\mathcal{P}}\sigma' = \mathsf{Pre}\sigma$ comme σ' coïncide avec σ sur $dom(\sigma)$ et $u\sigma_{\mathcal{P}}.\sigma' = u\sigma$. Enfin, y_1, y_2 sont des variables qui n'apparaissent nulle part ailleurs dans r'.

Ce qui conclut la preuve pour ce cas.

Second cas. $u = \mathsf{f}(a_1, a_2)$ pour un certain a_i avec $a_i\sigma = u_i$ pour chaque $i \in \{1, 2\}$. Nous avons $\sigma_{\mathcal{P}} = \{x_1 \mapsto a_1; x_2 \mapsto a_2\}$. Soit r' définie par

$$r' = \mathsf{decom}(r, \mathsf{att}(u, v)) = \mathsf{Pre}\sigma_{\mathcal{P}}, \mathsf{att}(x_1\sigma_{\mathcal{P}}, y_1), \mathsf{att}(x_2\sigma_{\mathcal{P}}, y_2) \to \mathsf{bad}$$

et σ' définie par $\sigma' = \sigma$.

Nous avons $\delta(x\sigma') \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(\sigma')$. De plus, $\mathsf{Pre}'\sigma' = \mathsf{Pre}\sigma_{\mathcal{P}}\sigma' = \mathsf{Pre}\sigma$ comme $\mathsf{Pre}\sigma_{\mathcal{P}} = \mathsf{Pre}$ parce que $dom(\sigma_{\mathcal{P}}) \cap vars(\mathsf{Pre}) = \emptyset$. Enfin, y_1, y_2 sont des variables qui n'appraissant pas ailleurs dans r', ce qui conclut la preuve du résultat principal.

Considérons le cas où $\mathsf{Add} = \mathsf{bad}$. Maintenant, si v s'unifie avec $\mathsf{f}(\_, \_)$, alors les hypothèses du lemme 5.9 sont satisfaites, donc nous obtenons ses conclusions, ce qui implique le résultat énoncé. □

Le lemme suivant démontre que, pour une règle de départ qui donne déjà bad, lorsqu'il existe un contexte côté P qui s'unifie avec la règle de départ, mais qu'il n'en existe pas côté Q, l'aplatissement mène encore à une règle qui donne bad. La preuve consiste à appeler récursivement le lemme A.1 dans le cas $\mathsf{Add} = \{\mathsf{bad}\}$.

Lemme A.2. *Soit* $r = \mathsf{Pre}, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_n, v_n) \to \mathsf{bad}$. *Soit* σ *une substitution telle que, pour tout* $x \in vars_{\mathsf{left}}(r)$, $x\sigma$ *est clos et* $\delta(x\sigma) \preccurlyeq \delta(x)$ *pour chaque* $x \in vars_{\mathsf{left}}(r)$.

Soit $C_1, \ldots, C_n$ *des contextes constructeurs. Supposons que* $u_i\sigma = C_i[u_1^i, \ldots, u_{k_i}^i]$. *Alors il existe une règle* $r' \in \mathsf{Flat}(r)$ *telle que* $r' = \mathsf{Pre}', \mathsf{att}(s_1^1, t_1^1), \ldots, \mathsf{att}(s_{k_n}^n, t_{k_n}^n) \to \mathsf{bad}$ *et* σ' *tel que* $u_i^j = s_i^j\sigma'$ *et* $\mathsf{Pre}'\sigma' =^{\mathsf{left}} \mathsf{Pre}\sigma$ *et les* t_i^j *sont des termes quelconques.*

Démonstration. Nous commençons par démontrer le fait suivant :

Fait. Soit $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{bad}$. Soit σ une substitution telle que pour tout $x \in vars_{\mathsf{left}}(r)$, $x\sigma$ est clos et $\delta(x\sigma) \preccurlyeq \delta(x)$. Soit C un contexte constructeur. Nous supposons que $u\sigma = C[u_1, \ldots, u_n]$. Alors il existe $r' \in \mathsf{Flat}(r)$ tel que $r' = \mathsf{Pre}', \mathsf{att}(u_1', v_1), \ldots, \mathsf{att}(u_n', v_n) \to \mathsf{bad}$ et σ' tel que $u_i = u_i'\sigma'$ et $\mathsf{Pre}'\sigma' =^{\mathsf{left}} \mathsf{Pre}\sigma$.

Preuve du fait. Nous procédons par récurrence sur C. L'initialisation est évidente. Nous supposons $C = \mathsf{f}(C_1, C_2)$ avec f d'arité 2. Le cas de l'arité arbitraire se démontre de la même manière. D'après le lemme A.1, le terme u est décomposable, et $r_1 = \mathsf{decom}(r, \mathsf{att}(u, v)) = \mathsf{Pre}_1, \mathsf{att}(u_1^1, v_1), \mathsf{att}(u_2^1, v_2) \to \mathsf{bad}$. Il existe une substitution σ_1 pour chaque $x \in vars_{\mathsf{left}}(r_1)$, $x\sigma_1$ est clos et $\mathsf{Pre}_1\sigma_1 = \mathsf{Pre}\sigma$, $\mathsf{f}(u_1', u_2')\sigma_1 = u\sigma$.

Rappelons que $u\sigma = C[u_1, \ldots, u_n] = \mathsf{f}(C_1[u_1, \ldots, u_m], C_2[u_{m+1}, \ldots, u_n])$. Par hypothèse de récurrence sur C_1 avec $r_1 = \mathsf{Pre}_1', \mathsf{att}(u_1', v_1) \to \mathsf{bad}$, il existe une règle $r_2 \in \mathsf{Flat}(r_1) \subset \mathsf{Flat}(r)$ telle que $r_2 = \mathsf{Pre}_2, \mathsf{att}(u_1^2, v_1'), \ldots, \mathsf{att}(u_m^2, v_m') \to \mathsf{bad}$ et une substitution σ_2 telle que $u_i^2\sigma_2 = u_i$ pour chaque $i \leqslant m$, et $\mathsf{Pre}_2\sigma_2 =^{\mathsf{left}} \mathsf{Pre}_1'\sigma_1$.

Par hypothèse de récurrence sur C_2 avec $r_2 = \mathsf{Pre}_2', \mathsf{att}(u_2', v_2) \to \mathsf{bad}$, il existe une règle $r_3 \in \mathsf{Flat}(r_2) \subset \mathsf{Flat}(r)$ telle que $r_3 = \mathsf{Pre}_3, \mathsf{att}(u_{m+1}^3, v_{m+1}'), \ldots, \mathsf{att}(u_n^3, v_n') \to \mathsf{bad}$ et une substitution σ_3 telle que $u_i^3\sigma_3 = u_i$ pour $i > m$, et $\mathsf{Pre}_3\sigma_3 =^{\mathsf{left}} \mathsf{Pre}_2'\sigma_2$.

Donc $\mathsf{Pre}_3 =^{\mathsf{left}} \mathsf{Pre}_3', \mathsf{att}(u_1^3, v_1''), \ldots, \mathsf{att}(u_m^3, v_m'')$ pour certains Pre_3' où $u_i^3\sigma_3 = u_i^2\sigma_2$ pour chaque $i \leqslant m$.

Donc nous avons

$$\begin{aligned}
&\mathsf{Pre}_3'\sigma_3, \mathsf{att}(u_1^3, v_1'')\sigma_3, \ldots, \mathsf{att}(u_m^4, v_m'')\sigma_3, \mathsf{att}(u_2', v_2)\sigma_2 \\
&= \mathsf{Pre}_3\sigma_3, \mathsf{att}(u_2', v_2)\sigma_2 \\
&=^{\mathsf{left}} \mathsf{Pre}_2'\sigma_2, \mathsf{att}(u_2', v_2)\sigma_2 \\
&=^{\mathsf{left}} \mathsf{Pre}_2\sigma_2, \mathsf{att}(u_1^2, v_1')\sigma_2, \ldots, \mathsf{att}(u_m^2, v_m')\sigma_2 \\
&=^{\mathsf{left}} \mathsf{Pre}_1'\sigma_1, \mathsf{att}(u_1^2, v_1')\sigma_2, \ldots, \mathsf{att}(u_m^2, v_m')\sigma_2 \\
&=^{\mathsf{left}} \mathsf{Pre}_1\sigma_1, \mathsf{att}(u_2^1, v_2)\sigma_1, \mathsf{att}(u_1^2, v_1')\sigma_2, \ldots, \mathsf{att}(u_m^2, v_m')\sigma_2 \\
&=^{\mathsf{left}} \mathsf{Pre}\sigma, \mathsf{att}(u_2^1, v_2)\sigma_1, \mathsf{att}(u_1^2, v_1')\sigma_2, \ldots, \mathsf{att}(u_m^2, v_m')\sigma_2
\end{aligned}$$

Nous en déduisons que $\mathsf{Pre}\sigma =^{\mathsf{left}} \mathsf{Pre}_3'\sigma_3$ et $u_i = u_i^3\sigma_3$ pour chaque i. Ce qui conclut la preuve du fait.

Il s'agit maintenant de démontrer le résultat principal, par récurrence sur le nombre n de contextes. L'initialisation ($n = 1$) provient du fait. Considérons l'hérédité.

Nous avons $r = \mathsf{Pre}, \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_{n+1}, v_{n+1}) \to \mathsf{bad}$. Définissons $\mathsf{Pre}' = \mathsf{Pre}, \mathsf{att}(u_{n+1}, v_{n+1})$. L'hypothèse de récurrence s'applique à $r = \mathsf{Pre}', \mathsf{att}(u_1, v_1), \ldots, \mathsf{att}(u_n, v_n) \to \mathsf{bad}$ et nous obtenons une règle $r_1 \in \mathsf{Flat}(r)$ telle que $r_1 = \mathsf{Pre}_1, \mathsf{att}(s_1^1, t_1^1), \ldots, \mathsf{att}(s_{k_n}^n, t_{k_n}^n) \to \mathsf{bad}$ et σ_1 telle que $u_i^j = s_i^j\sigma_1$ et $\mathsf{Pre}_1\sigma_1 =^{\mathsf{left}} \mathsf{Pre}'\sigma$.

Le fait qui a été démontré s'applique à $r_1 = \mathsf{Pre}'_1, \mathsf{att}(u'_{n+1}, v'_{n+1}) \to \mathsf{bad}$ pour un certain Pre'_1 bien choisi, où $u'_{n+1}\sigma_1 = u_{n+1}\sigma$ et $v'_{n+1}\sigma_1 = v_{n+1}\sigma_1$. Nous obtenons une règle

$$r_2 = \mathsf{Pre}_2, \mathsf{att}(\alpha_1, \beta_1), \dots, \mathsf{att}(\alpha_{k_{n+1}}, \beta_{k_{n+1}}) \to \mathsf{bad}$$

et une substitution σ_2 telle que $\mathsf{Pre}_2\sigma_2 =^{\mathsf{left}} \mathsf{Pre}'_1\sigma_1$ et $C_{n+1}[\alpha_1, \dots, \alpha_{k_{n+1}}]\sigma_2 = u'_{n+1}\sigma_1$. Donc

$$\mathsf{Pre}_2 = \mathsf{Pre}'_2, \mathsf{att}(\gamma^1_1, \delta^1_1), \dots, \mathsf{att}(\gamma^n_{k_n}, \delta^n_{k_n})$$

où $\gamma^j_i\sigma_2 = s^j_i\sigma_1$ pour chaque i, j. D'où :

$$\mathsf{Pre}'_2\sigma_2 =^{\mathsf{left}} \mathsf{Pre}_1\sigma_1 \backslash \{\mathsf{att}(u_{n+1}\sigma_1, v_{n+1}\sigma_1)\} =^{\mathsf{left}} \mathsf{Pre}\sigma$$

□

Démontrons le lemme 5.11.

Lemme 5.11. *Soient $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ une règle, σ une substitution telle que pour tout $x \in vars_{\mathsf{left}}(r)$, $x\sigma$ est clos et $\delta(x\sigma) \preccurlyeq \delta(x)$, et C un contexte linéaire construit sur Σ_c. Supposons que $u\sigma = C[u_1, \dots, u_n]$ et v et C ne sont pas unifiables.*

Alors il existe $r' \in \mathsf{Flat}(r)$ tel que $r' = \mathsf{Pre}', \mathsf{att}(u'_1, v_1), \dots, \mathsf{att}(u'_n, v_n) \to \mathsf{bad}$ et σ' tel que $u_i = u'_i\sigma'$ et $\mathsf{Pre}'\sigma' =^{\mathsf{left}} \mathsf{Pre}\sigma$.

Démonstration. Soit $r = \mathsf{Pre}, \mathsf{att}(u, v) \to \mathsf{Add}; \mathsf{Del}$ une règle, σ une substitution telle que pour tout $x \in vars_{\mathsf{left}}(r)$ $\delta(x\sigma) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r)$, et C un contexte linéaire construit sur Σ_c.

Supposons que $u\sigma = C[u_1, \dots, u_n]$ tandis que v et C ne sont pas unifiables. Prenons C' un préfixe maximal de C tel que v et C' sont unifiables. Nous pouvons définir la substitution σ' telle que $\sigma' = \sigma$ sur $vars_{\mathsf{left}}(r)$ et σ' unifie v et C' : $v\sigma' = C'[v_1, \dots, v_n]$.

D'après le lemme 5.9, il existe $r_1 \in \mathsf{Flat}(r)$ de la forme :

$$r_1 = \mathsf{Pre}_1, \mathsf{att}(u'_1, v'_1), \dots, \mathsf{att}(u'_n, v'_n) \to \mathsf{Add}_1; \mathsf{Del}_1$$

et σ_1 une substitution telle que :

1. Pour tout $x \in dom(\sigma)$, $x\sigma$ est clos
2. $\delta(x\sigma_1) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r_1)$;
3. $(\mathsf{Pre}_1, \mathsf{Add}_1, \mathsf{Del}_1)\sigma_1 = (\mathsf{Pre}, \mathsf{Add}, \mathsf{Del})\sigma'$;
4. $\mathsf{att}(u, v)\sigma' = \mathsf{att}(C'[u'_1, \dots, u'_n], C'[v'_1, \dots, v'_n])\sigma_1$.

Rappelons que $C = C'[C_1, \dots, C_n]$. Mais v et C ne sont pas unifiables, et par maximalité de C', nous supposons sans perte de généralité que v'_1 et C_1 ne sont pas unifiables.

Comme C_1 est un contexte construit sur Σ_c, C_1 n'est pas une feuille (sinon v'_1 serait unifiable avec C_1). Donc $C_1 = \mathsf{f}(C''_1, C''_2, \dots, C''_n)$. Par maximalité de C', v'_1 n'est même pas unifiable avec $\mathsf{f}(\_, \_, \dots, \_)$. u'_1 est unifiable avec C_1 comme $u\sigma = C[u_1, \dots, u_n]$.

Le lemme A.1 s'applique à la règle $r_1 = \mathsf{Pre}'_1, \mathsf{att}(u'_1, v'_1) \to \mathsf{Add}_1; \mathsf{Del}_1$, u'_1 est décomposable et la règle suivante est bien définie :

$$r_2 = \mathsf{decom}(r_1, \mathsf{att}(u'_1, v'_1)) = \mathsf{Pre}_2, \mathsf{att}(u''_1, v''_1), \mathsf{att}(u''_2, v''_2), \dots, \mathsf{att}(u''_n, v''_n) \to \mathsf{bad}$$

De plus, il existe une substitution σ_2 telle que $\delta(x\sigma_2) \preccurlyeq \delta(x)$ pour chaque $x \in vars_{\mathsf{left}}(r_2)$, $\mathsf{Pre}_2\sigma_2 = \mathsf{Pre}'_1\sigma_1$, $\mathsf{f}(u''_1, u''_2, \dots, u''_n)\sigma' = u'_1\sigma$ et $v''_1, v''_2, \dots, v''_n$ sont des variables distinctes qui n'apparaissent pas ailleurs dans r_2.

Maintenant nous écrivons $r_2 = \mathsf{Pre}'_2, \mathsf{att}(u''_1, v''_1), \mathsf{att}(u''_2, v''_2), \ldots, \mathsf{att}(u''_n, v''_n), \mathsf{att}(\alpha_2, \beta_2), \ldots, \mathsf{att}(\alpha_n, \beta_n) \to \mathsf{bad}$, tandis que $\alpha_i\sigma_2 = u'_i\sigma_1$ et $\beta_i\sigma_2 = v'_i\sigma_1$ pour chaque $i \geqslant 2$. Le lemme A.2 s'applique aux contextes $C''_1, C''_2, C_2, \ldots, C_n$. Nous obtenons une règle

$$
\begin{aligned}
& r_3 = \\
& \mathsf{Pre}_3, \mathsf{att}(\gamma^1_1, \delta^1_1), \ldots, \mathsf{att}(\gamma^1_{k_1}, \delta^1_{k_1}), \\
& \mathsf{att}(\gamma^2_1, \delta^2_1), \ldots, \mathsf{att}(\gamma^2_{k_2}, \delta^2_{k_2}), \\
& \ldots \\
& \mathsf{att}(\gamma^n_1, \delta^n_1), \ldots, \mathsf{att}(\gamma^n_{k_n}, \delta^n_{k_n}), \\
& \mathsf{att}(\eta^2_1, \theta^2_1), \ldots, \mathsf{att}(\eta^2_{j_2}, \theta^2_{j_2}) \\
& \ldots \\
& \mathsf{att}(\eta^n_1, \theta^n_1), \ldots, \mathsf{att}(\eta^n_{j_n}, \theta^n_{j_n}) \\
& \to \mathsf{bad}
\end{aligned}
$$

et une substitution σ_3 où $\mathsf{Pre}_3\sigma_3 =^{\mathsf{left}} \mathsf{Pre}'_2\sigma_2$ et $u''_i\sigma_2 = C'''_i[\gamma^i_1, \ldots, \gamma^i_{k_i}]\sigma_3$ pour chaque i, $\alpha_i\sigma_2 = C_i[\eta^i_1, \ldots, \eta^i_{j_i}]\sigma_3$ pour chaque $i \geqslant 2$ et les δ^i_j, θ^i_j sont des termes quelconques.

Nous avons :

$$
\begin{aligned}
u\sigma &= C'[u'_1, \ldots, u'_n]\sigma_1 \\
&= C'[\mathsf{f}(u''_1, u''_2), u'_2, \ldots, u'_n]\sigma_2 \\
&= C'[\mathsf{f}(C'''_1[(\gamma^1_i)_{1\leqslant i\leqslant k_1}], C'''_2[(\gamma^2_i)_{1\leqslant i\leqslant k_2}], \\
&\quad C_2[(\eta^2_i)_{1\leqslant i\leqslant j_2}], \ldots, C_n[(\eta^n_i)_{1\leqslant i\leqslant k_n}]]\sigma_3 \\
&= C[(\gamma^1_i)_{1\leqslant i\leqslant k_1}, (\gamma^2_i)_{1\leqslant i\leqslant k_2}, (\eta^2_i)_{1\leqslant i\leqslant j_2}, \ldots, (\eta^n_i)_{1\leqslant i\leqslant k_n}]\sigma_3
\end{aligned}
$$

De plus, comme $u'_i\sigma_1 = \alpha_i\sigma_2$ pour chaque $i \geqslant 2$, nous avons :

$$
\begin{aligned}
\mathsf{Pre}_2\sigma_2 &=^{\mathsf{left}} \mathsf{Pre}_1\sigma_1, \mathsf{att}(u'_2, v'_2)\sigma_1, \ldots, \mathsf{att}(u'_n, v'_n)\sigma_1 \\
&=^{\mathsf{left}} \mathsf{Pre}'_2\sigma_2, \mathsf{att}(\alpha_2, \beta_2)\sigma_2, \ldots, \mathsf{att}(\alpha_n, \beta_n)\sigma_2
\end{aligned}
$$

donc $\mathsf{Pre}_1\sigma_1 =^{\mathsf{left}} \mathsf{Pre}'_2\sigma_2 =^{\mathsf{left}} \mathsf{Pre}_3\sigma_3$. Comme $\mathsf{Pre}_1\sigma_1 = \mathsf{Pre}\sigma$ nous obtenons $\mathsf{Pre}\sigma =^{\mathsf{left}} \mathsf{Pre}_3\sigma_3$, ce qui conclut la preuve. □

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Vérification de propriétés d'indistinguabilité pour les protocoles cryptographiques.

Résumé : Cette thèse s'inscrit dans le domaine de la vérification de protocoles cryptographiques dans le modèle symbolique. Plus précisément, il s'agit de s'assurer, à l'aide de méthodes formelles, que de petits programmes distribués satisfont à des propriétés d'indistinguabilité, c'est-à-dire qu'un attaquant n'est pas capable de deviner quel protocole (parmi deux) il observe. Ce formalisme permet d'exprimer des propriétés de sécurité comme le secret fort, l'intraçabilité ou l'anonymat. De plus, les protocoles sont exécutés simultanément par un grand nombre d'agents, à plusieurs reprises, si bien que nous nous heurtons très rapidement à des résultats d'indécidabilité. Dès lors, il faut ou bien tenir compte du nombre arbitraire de sessions, et rechercher des méthodes de semi-décision ou identifier des classes décidables ; ou bien établir des procédures de décision pour un nombre fini de sessions.
Au moment où nous avons commencé les travaux présentés dans cette thèse, les outils de vérification de l'indistinguabilité pour un nombre borné de sessions ne permettaient de traiter que très peu de sessions : dans certains cas il était tout juste possible de modéliser un échange complet. Cette thèse établit des procédures de décision efficaces dans ce cadre. Dans un premier temps, nous établissons des résultats de petite attaque. Pour des protocoles déterministes, nous démontrons qu'il existe une attaque si, et seulement si, il existe une attaque bien typée, lorsque toute confusion entre les types des variables est évitée. De plus, nous prouvons que, lorqu'il existe une attaque, l'attaquant peut la trouver en utilisant au plus trois constantes. Dans un second temps, nous traduisons le problème de distinguer entre deux protocoles en termes d'accessibilité dans un système de planification, qui est résolu par l'algorithme du graphe de planification associé à un codage SAT. Dans un dernier temps, l'implémentation de l'outil SAT-Equiv nous permet de confirmer l'efficacité de la démarche à travers une comparaison vis-à-vis des outils comparables.

Mots clés : *Protocoles cryptographiques, Méthodes formelles, Modèle symbolique*

Verification of indistinguishability properties in cryptographic protocols.

Abstract: This thesis presents methods to verify cryptographic protocols in the symbolic model: formal methods allow to verify that small distributed programs satisfy equivalence properties. Those properties state that an attacker cannot decide what protocol is beeing executed. Strong secrecy, and privacy type properties, like anonymity and unlinkeability, can be modelled through this formalism. Moreover, protocols are executed simultaneously by an unbounded number of agents, for an unbounded number of sessions, which leads to indecidability results. So, we have either to consider an arbitrary number of sessions, and search for semi-decision procedures and decidable classes; or to establish decision procedures for a finite number of sessions.
When we started the work presented in this thesis, the existing equivalence checkers in the bounded model were highly limited. They could only handle a very small number of sessions (sometimes no more than three). This thesis finds efficient decision procedures for bounded verification of equivalence properties. Our first step is to provide small attack results. First, for deterministic processes, there exists an attack if, and ony if, there is a well-typed attack, assuming that there is no confusion between variable types. Second, when there exists a flaw, the attacker needs at most three constants to find it. Then, our second step is to translate the indistinguishability problem as a reachability problem in a planning system. We solve this second problem through planning graph algorithm and SAT encoding. In a final step, we present the implementation of the SAT-Equiv tool, which allows us to evaluate our approach. In particular, a benchmark with comparable tools proves the efficiency of SAT-Equiv.

Keywords: *Cryptographic protocols, Formal methods, Symbolic model*

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France