Test à partir de spécifications axiomatiques

Delphine Longuet

N° d'identification : 2007EVRY0020

UNIVERSITÉ D'ÉVRY-VAL D'ESSONNE
U.F.R. SCIENCES FONDAMENTALES ET APPLIQUÉES

THÈSE

présentée par

Delphine LONGUET

pour obtenir le grade de

DOCTEUR EN SCIENCES
DE L'UNIVERSITÉ D'ÉVRY-VAL D'ESSONNE

Spécialité : INFORMATIQUE

Test à partir de spécifications axiomatiques

Soutenue le 12 octobre 2007 devant la commission d'examen composée de

M. Marc AIGUIER	directeur de thèse
M. Yves BERTOT	rapporteur
M. Pierre-Louis CURIEN	
Mme Pascale LE GALL	
M. Vlad RUSU	rapporteur
M. Jan RUTTEN	

Remerciements

Ces trois années de thèse auront été pour moi l'occasion de découvrir et d'approfondir des thématiques de recherche aussi variées qu'enrichissantes. Je remercie mon encadrant et directeur de thèse Marc Aiguier de m'avoir donné cette opportunité, ainsi que de m'avoir guidée et soutenue tout au long du périple. Ses connaissances aussi bien que son énergie m'ont toujours été d'une grande aide.

En sa double qualité de directrice d'équipe et de membre du jury, je tiens à remercier Pascale Le Gall du recul précieux qu'elle a su apporter à mon travail. Nos collaborations fructueuses m'ont beaucoup apporté.

Des autres membres de mon jury, je tiens tout d'abord à remercier le président, Pierre-Louis Curien, de m'avoir fait cet honneur. Je remercie mes rapporteurs Vlad Rusu et Yves Bertot de s'être intéressé à mes travaux. Les remarques et questions très pertinentes qu'ils m'ont soumises ouvrent des perspectives de recherche qu'il me plaira très certainement d'explorer. Je tiens enfin à remercier tout particulièrement Jan Rutten de s'être déplacé jusqu'à Évry pour assister à ma soutenance. Sa générosité et son soutien m'ont permis d'être où je suis aujourd'hui. Je lui en suis extrêmement reconnaissante.

Rien de tout ceci ne se serait aussi bien passé sans la présence quotidienne des autres doctorants du laboratoire et celle, plus rare mais pas moins précieuse, de mes amis. Pour l'ambiance et les conversations incomparables du petit bain, je remercie en particulier, dans le désordre : Assia, ma co-burale préférée, Matthieu et sa rigueur maladive (quelqu'un qui me comprend...), Antoine et sa mauvaise humeur légendaire, François et son verbe particulier (et ces discussions atypiques que nous avons pu avoir) ; les petits nouveaux qui ne sont déjà plus si petits, François, Stéphane, Thomas et mon bon vieux Popo ; les anciens, qui ne se font pas tous oublier, Vincent, Nicolas, Jean-Marc. Mes amis, pour m'avoir aidé à apaiser mes peines et ma soif, et ce depuis des années : BK, Aurèl, Antoine, RV, Bidouille, Baptiste, Ismail. . . Je remercie Xavier, qui a eu en particulier la patience de relire ma thèse, mais qui m'a également prodigué nombre de précieux conseils et fromages.

Il est une présence à laquelle je dois tout, c'est celle de ma famille. Je remercie mes parents de m'avoir toujours suivie et supportée quels qu'aient été mes choix. Aurélien et Cristelle d'être simplement là, et c'est déjà beaucoup.

Ces remerciements seraient aussi incomplets qu'une Irlande sans Guinness si je manquais de remercier Fabrice, pour avoir partagé avec moi ces quatres dernières années. Il a été pour moi beaucoup plus qu'un soutien et qu'un réconfort. Je te remercie, Fabrice, d'avoir été là, et même si la roue tourne, je sais que nos chemins se feront pour longtemps encore côte à côte.

Sommaire

INTRODUCTION

DEPUIS la crise du logiciel de la fin des années soixante, il est clair qu'une approche méthodologique est indispensable à la conception de systèmes informatiques. Il s'est avéré à cette époque que la complexité des logiciels était devenue bien trop importante pour qu'il soit envisageable de livrer un logiciel correspondant à son cahier des charges dans des délais et pour un coût raisonnables. Parmi les exemples que l'on peut citer pour illustrer l'ampleur de l'impact des défaillances dues au manque de méthodologie, on trouve la sonde Mariner 1 vers Vénus : lancée le 22 juillet 1962, elle a été détruite à peine 5 minutes après son lancement car sa trajectoire était devenue incontrôlable par cause d'une erreur de programmation. Plus récemment, en 1996, l'explosion du lanceur Ariane 5, qui a coûté un demi milliard de dollars, est due à une faute logicielle d'un composant dont le fonctionnement n'était pas indispensable durant le vol [JM97].

Il s'est alors avéré nécessaire de définir un ensemble de méthodes et d'outils dédiés à la conception, au développement et à la maintenance des systèmes informatiques. Ces méthodes et outils forment ce qu'on appelle le génie logiciel. Tout au long du processus de conception, depuis la formulation informelle des besoins jusqu'au produit final, plusieurs types de méthodes de validation et de vérification sont utilisés pour assurer la qualité du produit final. Parmi ces méthodes, on trouve notamment les techniques de preuve, de test, de prototypage rapide ou de vérification de modèles (*model-checking*). Ce sont les techniques de test qui nous intéresserons plus particulièrement ici.

Le test est l'un des moyens les plus utilisés pour la validation du logiciel. L'activité de test consiste à exécuter le logiciel sur un sous-ensemble de ses entrées possibles de manière à déceler d'éventuelles erreurs. La présence d'erreurs est établie par confrontation du comportement du logiciel avec un objet de référence.

Le processus de test est généralement décomposé en trois phases : la sélection, la soumission et la décision. Ces trois phases consistent respectivement à sélectionner le sous-ensemble des entrées sur lequel le logiciel sera exécuté, à soumettre ces entrées au logiciel en collectant les sorties c'est-à-dire les réponses du logiciel, et à décider de l'adéquation de ces sorties avec les sorties attendues.

La sélection des données à soumettre au logiciel peut être effectuée selon différentes approches. Elles peuvent par exemple être choisies aléatoirement parmi les données possibles, ou bien selon certains critères représentatifs des caractéristiques du logiciel qu'on veut tester. Ces critères sont choisis à partir d'un objet de référence qui peut être une spécification informelle, une spécification formelle ou un prototype. Une spécification formelle d'un système informatique est une description des fonctionnalités et/ou du comportement attendus de ce système exprimée dans un langage rigoureux, c'est-à-dire mathématiquement défini. On distingue les spécifications formelles des spécifications dites informelles qui utilisent un langage ambigu

pouvant donner lieu à différentes interprétations, comme le langage naturel par exemple. On parle également de spécifications semi-formelles, notamment au sujet d'UML, qui tend vers une formalisation de notations graphiques tout en laissant relativement libre leur interprétation.

Le langage utilisé pour spécifier formellement un système peut être de différentes natures, suivant les aspects du système à décrire. On peut utiliser, entre autres, un formalisme logique, une algèbre de processus, un langage orienté modèles ou un langage muni d'une représentation graphique comme les systèmes de transitions ou les réseaux de Petri. Certains de ces formalismes de spécification sont plus adaptés à décrire les aspects fonctionnels du système (comportement sur les données) ou ses aspects dynamiques (actions et communications avec l'environnement). Les spécifications logiques sont généralement utilisées pour décrire les fonctionnalités du système, tandis que les algèbres de processus et les systèmes de transitions sont utilisés pour spécifier l'aspect dynamique des systèmes.

Pendant longtemps, cette distinction entre formalismes orientés données et formalismes orientés dynamique a été très nette. Cependant, sont apparues récemment des extensions de chacun de ces deux types de formalismes afin de prendre également en compte les aspects de l'autre. On trouve par exemple des extensions des formalismes dynamiques aux données : FSP (pour *Finite State Processes*) est une algèbre de processus étendue aux données, tandis que les IOSTS (pour *Input Output Symbolic Transition Systems*) sont une extension aux données des automates communicants. Inversement, les formalismes logiques, dits également axiomatiques, peuvent être utilisés pour spécifier les aspects dynamiques des systèmes. Les logiques considérées pour la spécification des aspects fonctionnels des systèmes sont en général des restrictions ou des extensions de la logique du premier ordre. Il est alors possible de profiter de la vision dynamique offerte par les logiques modales pour spécifier, de façon plus abstraite que les algèbres de processus ou les systèmes de transitions, le comportement d'un système. Ainsi, les extensions des logiques modales au premier ordre permettent d'allier spécification comportementale et spécification des données.

Lorsque la phase de sélection d'un jeu de tests est opérée à partir d'un objet de référence décrivant plus ou moins formellement le comportement du logiciel, sans connaissance de l'implantation elle-même, on parle de test « boîte noire ». Différentes approches de test boîte noire ont été développées pour chacun des formalismes mentionnés ci-dessus. Parmi ces approches, nous en distinguons deux qui illustrent la distinction entre données et dynamique que nous avons mentionnée plus haut.

Afin de tester les aspects fonctionnels d'un système, une des théories de test qui a été définie a pour objet des systèmes décrits à l'aide de spécifications équationnelles, dites aussi algébriques [Ber91, BGM91]. Dans ce cadre, deux hypothèses fondamentales sont posées : le système sous test est considéré comme un modèle du formalisme logique dans lequel la spécification est exprimée, c'est-à-dire une algèbre ; les cas de tests sont des équations qui ne contiennent pas de variables pour pouvoir être évaluées par le système sous test. L'interprétation des cas de test est alors définie comme la satisfaction de ces formules par le système en tant que modèle, une formule étant satisfaite par un modèle si elle est vraie pour une certaine notion de vérité dans ce modèle. La notion fondamentale de correction d'un système par rapport à sa spécification est alors définie dans ce cadre de la façon suivante : un système est correct s'il satisfait exactement les mêmes formules sans variables que la spécification. Le jeu de tests composé de toutes ces formules est alors dit exhaustif puisqu'il permettrait de montrer la correction du système s'il pouvait être soumis dans son intégralité. Le problème réside dans le fait que cet ensemble de tests est de très grande taille, voire infini. Il est donc nécessaire d'en sélectionner un sous-ensemble de taille raisonnable à soumettre effectivement au système. Une des méthodes de sélection qui a été très étudiée dans le cadre des spécifications équationnelles

est celle appelée dépliage des axiomes [Mar91, Mar95, AABLM05]. Son principe est de diviser l'ensemble exhaustif de tests en sous-ensembles selon des critères dérivés des axiomes de la spécification. Il reste donc une phase de génération consistant ensuite à choisir des cas de test dans chacun de ces sous-ensembles afin de construire un jeu de tests fini couvrant le jeu de tests exhaustif de départ.

Lorsqu'il s'agit de tester les aspects dynamiques des systèmes, la théorie de test boîte noire qui a été définie est appelée test de conformité [LY96, Tre95]. Dans le cadre du test de conformité, la spécification du système sous test peut être une machine à états finis ou un système de transitions étiquetées. L'hypothèse de test fondamentale est d'assimiler le système sous test à un objet de même nature que la spécification. Le comportement du système, exprimé en termes de trace d'exécution c'est-à-dire de suite d'états, d'entrées et de sorties, est alors confronté au comportement décrit par la spécification. Un système est dit correct dans ce cadre si tous ses comportements sont autorisés par la spécification. Par exemple, lorsque le système et la spécification sont des systèmes de transitions étiquetées, la correction du système peut être définie par la relation *ioco* [Tre95] : le système est conforme à sa spécification pour *ioco* si toute trace de la spécification exécutée par le système donne une des sorties autorisées par la spécification pour cette trace. La sélection des tests dans ce cadre peut être effectuée par un parcours du système de transitions permettant de construire pas à pas des traces à soumettre au système sous test. Elle peut également être conduite par un objectif de test sous la forme d'un système de transitions représentant un ensemble de comportements à tester. Ce sont les traces de la spécification correspondant à cet objectif de test qui seront alors soumises au système.

La barrière entre formalismes de spécification dédiés aux données et formalismes dédiés à la dynamique s'étant estompée grâce aux extensions de ces formalismes aux aspects dynamiques et aux données respectivement, des techniques de test ont pu être développées pour ces nouveaux formalismes.

Récemment, les approches du test de conformité ont été étendues afin de prendre également en compte à la fois l'aspect fonctionnel des systèmes. Les formalismes utilisés sont alors des machines à états finis étendues aux données comme les EFSM (pour *Extended Finite State Machines*) [MS98] ou des systèmes de transitions symboliques comme les STS (pour *Symbolic Transition Systems*) [FTW04, PBG99] ou les IOSTS [RdBJ00, GLRT06]. Les approches fondées sur les extensions symboliques des modèles dynamiques permettent par exemple d'utiliser une technique d'analyse particulière appelée exécution symbolique afin de définir une stratégie de sélection de tests. Le principe est d'utiliser des symboles comme données d'entrée au lieu de données concrètes et de dériver un arbre d'exécution symbolique afin de décrire toutes les exécutions possibles du système de façon symbolique. Les objectifs de test sont alors choisis parmi les sous-arbres de cet arbre d'exécution à l'aide de certains critères selon les caractéristiques du système à tester.

Cependant, il n'existe pas à ce jour de travaux proposant des techniques de test dédiées à des systèmes mêlant dynamique et données et spécifiés à l'aide de formalismes axiomatiques. La spécification d'un système dans un tel formalisme étant un ensemble de propriétés, elle donne de fait une vision plus abstraite du système qu'un système de transitions qui est une spécification exécutable. Le travail de cette thèse cherche alors à étendre à des systèmes dynamiques le cadre de test défini pour des systèmes décrits à l'aide de spécifications algébriques. L'objectif est d'aborder le test de ce type de systèmes de manière plus abstraite que l'approche du test de conformité. Comme on l'a dit plus haut, parmi les formalismes de spécification qui permettent de lier la description des aspects fonctionnels et dynamiques d'un système se trouvent les extensions des modèles dynamiques aux données, mais également des extensions de la logique du premier ordre à des aspects dynamiques comme les logiques modales. Le cadre de test à partir de spécifications algébriques ayant été défini de façon générale pour des spécifications axiomatiques, notre approche est de faire entrer les systèmes

dynamiques dans ce cadre en les décrivant par de telles spécifications. La logique modale du premier ordre étant très adaptée à décrire ce type de systèmes, c'est cette logique que nous avons choisie comme formalisme axiomatique pour la spécification de ces systèmes. Le système sous test est alors spécifié, non pas à l'aide d'un automate ou d'un système de transitions, qui ne représentent qu'un seul modèle du système, mais à l'aide d'un ensemble d'axiomes exprimés en logique modale du premier ordre. Cela permet de se placer à un niveau d'abstraction plus élevé, une spécification modale dénotant une classe de systèmes. C'est ce qu'on appelle de la sous-spécification : on ne signifie rien de plus que ce qui est effectivement dit. De plus, les systèmes ainsi spécifiés sont interprétés par des modèles mathématiques appelés coalgèbres [Kur00, Rut00, Jac] qui sont une généralisation des systèmes de transitions.

Dans le cadre de test défini initialement pour des spécifications axiomatiques, la méthode de sélection d'un jeu de tests effectif à soumettre au système par dépliage des axiomes a été étudiée de façon approfondie pour des spécifications équationnelles conditionnelles positives [Ber91, Mar91, BGM91, Mar95, AABLM05]. Ces spécifications étant pourvues de techniques de preuve bien connues et efficaces, l'idée fondamentale est d'utiliser ces techniques pour la sélection de cas de test. En effet, les cas de test doivent être des conséquences de la spécification, c'est-à-dire des formules prouvées être des théorèmes dans la théorie définie par la spécification. Ils peuvent alors être déduits des axiomes de la spécification, en utilisant le calcul correspondant au formalisme de la spécification. La procédure de dépliage consiste alors à raffiner le jeu de tests exhaustif initial en remplaçant ces tests par des ensembles de formules choisies dans leurs arbres de preuve. Les jeux de tests ainsi obtenus doivent être aussi puissants que le jeu de test exhaustif : aucun test ne doit être perdu (stratégie correcte) et aucun test ne doit être ajouté (stratégie complète). Afin de montrer ce résultat, une solution consiste à montrer que tout arbre de preuve peut être transformé de façon à respecter une certaine structure qui rend possible la stratégie de choix, c'est-à-dire prouver la normalisation de la transformation.

Afin d'étendre ce cadre de test à des spécifications exprimées en logique modale du premier ordre, notre démarche est la suivante. La première étape consiste à généraliser la méthode dépliage aux spécifications du premier ordre. Une restriction s'impose alors quant à l'utilisation du quantificateur existentiel dans les axiomes de la spécification. Il est en effet possible de montrer que tester une formule quantifiée existentiellement revient à prouver cette formule dans le système sous test. Ce type de formules ne peut donc pas être utilisé dans le cadre du test. Cependant, hormis cette restriction mineure, les résultats d'exhaustivité ainsi que la méthode de sélection par dépliage trouvent une généralisation satisfaisante aux spécifications du premier ordre sans quantificateurs [AALL07].

Il s'agit ensuite d'adapter ce cadre de test à des spécifications modales du premier ordre. Le premier formalisme de spécification que nous avons choisi est une extension modale de la logique conditionnelle positive pour laquelle le cadre de test a été initialement défini. Cette restriction sur la forme des formules modales permet d'étudier l'influence de la présence de modalités dans le processus de sélection par dépliage des axiomes ainsi que la dualité des résultats d'exhaustivité avec ceux du cadre original. En effet, les coalgèbres dans lesquelles les spécifications modales du premier ordre sont interprétées sont des constructions duales, dans un sens que nous définirons, des modèles algébriques. L'exhaustivité étant une notion liée à la sémantique des spécifications utilisées pour décrire le système sous test, la dualité des modèles algébriques et coalgébriques se retrouve dans les preuves d'existence d'un jeu de tests exhaustif dans les cadres algébrique et modal.

Une fois le cadre de test adapté aux spécifications modales conditionnelles positives [AL07], nous pouvons aborder la généralisation aux spécifications modales du premier ordre. Pour les mêmes raisons que

dans le cas des spécifications du premier ordre, les formules sont restreintes aux formules sans quantificateurs. La logique modale du premier ordre choisie est la logique sous-jacente au langage de spécifications CoCasl [MSRR06], tout comme la logique du premier ordre est la logique sous-jacente au langage de spécifications algébriques Casl [ABK+02, CoFI04]. CoCasl est une extension coalgébrique de Casl qui permet de spécifier conjointement des types de données algébriques et des processus de nature coalgébrique. La logique modale de CoCasl permet d'exprimer des propriétés sur de tels processus, par exemple des propriétés de vivacité et de sûreté. L'extension du cadre de test à des spécifications modales du premier ordre [LA07] permet alors de tester de telles propriétés.

La thèse se compose de trois parties organisées comme suit :

Les préliminaires contiennent les notions fondamentales nécessaires à la présentation de la théorie du test à partir de spécifications axiomatiques. Cette théorie étant définie indépendamment de toute logique, nous présentons tout d'abord le cadre abstrait des logiques générales afin de fixer les notions et les notations des différentes composantes d'une logique dont nous aurons besoin dans la présentation du cadre général de test. Nous définissons ensuite les deux types de logiques dans lesquels nous étudions le test tout au long de cette thèse, la logique du premier ordre et la logique modale du premier ordre, que nous présentons comme des instances de logiques générales. Nous terminons cette partie par la présentation de la théorie du test dans le cadre des logiques générales.

La première partie est consacrée au test à partir de spécifications du premier ordre. Nous présentons tout d'abord un état de l'art de la méthode de sélection par dépliage des axiomes ainsi qu'une contribution à l'étude des conditions sous lesquelles l'existence d'un jeu de tests exhaustif est assurée. Ces résultats ainsi que la procédure de dépliage sont ensuite généralisés aux spécifications du premier ordre sans quantificateurs. Nous montrons également la raison de cette restriction sur les quantificateurs.

La seconde partie présente les extensions de la méthode de sélection de la première partie aux spécifications modales du premier ordre ainsi que les résultats d'exhaustivité, de correction et de complétude de la procédure assurant la pertinence de la sélection. La première extension se situe dans le cadre des spécifications modales conditionnelles positives, ce qui permet d'adapter la procédure de sélection originale en se focalisant uniquement sur la présence de modalités. La procédure est ensuite généralisée aux spécifications modales du premier ordre sans quantificateurs. Dans les deux cadres, des résultats d'exhaustivité sont montrés. Une comparaison avec l'approche du test de systèmes dynamiques par le test de conformité est également proposée à la suite de la présentation de l'adaptation du cadre de test aux spécifications modales conditionnelles positives.

L'annexe A est un bref récapitulatif des notions de la théorie des catégories nécessaires à la lecture de ce document. Sont présentées succinctement les notions de catégorie, de morphisme, de foncteur, de sous-catégorie, de catégorie duale, ainsi que la vision catégorique de notions de théorie des ensembles et les concepts de limites et colimites.

L'annexe B présente des résultats de normalisation d'arbres de preuve que nous avons établis. Ce travail a été conduit de façon parallèle et n'est donc pas directement lié à la problématique de la thèse, mais il fournit des résultats utilisés dans plusieurs des preuves de complétude de la procédure de dépliage.

Préliminaires

Théorie du test à partir de spécifications axiomatiques

LA THÉORIE du test que nous allons présenter a été définie pour des systèmes spécifiés dans des formalismes axiomatiques, appelés aussi logiques. Les logiques générales, définies à partir des institutions, fournissent un cadre unifié pour la définition de tels formalismes. Elles permettent en effet de définir de façon abstraite les différentes composantes d'une logique que sont la syntaxe, la sémantique et le calcul, ainsi que les liens entre ces trois notions. Nous profitons alors de ce cadre général de description des logiques pour présenter les notions et les résultats dont nous aurons besoin dans la suite et qui peuvent être définis indépendamment de la logique manipulée.

Nous suivons ensuite le schéma de présentation donné par les logiques générales pour présenter les deux classes de logiques dans lesquelles nous étudions le test, à savoir la logique du premier ordre et la logique modale du premier ordre. Ces deux logiques sont munies d'un calcul des séquents approprié qui sera utilisé pour la sélection de tests à partir des spécifications exprimées dans ces logiques. Les logiques équationnelle et conditionnelle positive sont alors définies comme des restrictions de la logique du premier ordre tandis que la logique modale conditionnelle positive est définie comme une restriction de la logique modale du premier ordre. La définition de la sémantique de la logique modale du premier ordre nécessite l'introduction de la notion de coalgèbre comme abstraction des modèles dynamiques tels que les systèmes de transitions. Dans les deux cadres logiques, des résultats liés à l'existence d'un modèle particulier, initial dans le cas algébrique et terminal dans le cas modal, sont présentés.

Enfin, la théorie du test à partir de spécifications axiomatiques est présentée de façon abstraite dans le cadre des logiques générales. Ce cadre théorique est celui dans lequel tous les travaux présentés dans cette thèse se situent. Nous définissons alors les notions fondamentales de correction d'un système sous test, d'exhaustivité d'un jeu de tests ainsi que la notion de critère de sélection et ses propriétés de correction et de complétude.

Chapitre I

Logiques générales

Tout formalisme logique, dit aussi formalisme axiomatique, comporte trois aspects intimement liés.

La syntaxe. Le premier objectif d'un formalisme est d'offrir la possibilité d'exprimer des propriétés permettant de décrire des systèmes. Pour que le procédé de validation de ces propriétés soit vérifiable par un ordinateur, il faut que leur énoncé lui-même soit mis sous forme directement « compréhensible » par un ordinateur, c'est-à-dire sous forme symbolique. La collection de tous les énoncés à établir sur le système décrit la spécification de ce dernier, ce qu'on peut voir comme un cahier des charges formel. La définition des suites de symboles reconnaissables pour un formalisme en caractérise la syntaxe. L'énoncé des propriétés respectant cette syntaxe est usuellement appelé une formule.

La sémantique. L'énoncé symbolique des propriétés ne suffit pas à lui seul, puisqu'il n'est qu'une suite de caractères et de symboles. Il est donc nécessaire de donner de façon rigoureuse un sens aux symboles et aux suites de symboles du langage, correspondant au phénomène que l'on veut garantir par ces propriétés. Ce sens est donné par un ensemble de modèles mathématiques : la sémantique. Ces modèles peuvent être vus comme des abstractions mathématiques des systèmes dont on veut établir la correction. Pour établir cette correction, on munit la sémantique d'une relation binaire entre les modèles et les formules, appelée relation de satisfaction. Grâce à cette relation, il est alors possible de dire si une certaine propriété est ou non validée par un modèle, c'est-à-dire si cette propriété est vraie dans ce modèle pour une certaine interprétation.

Le calcul. À partir de la sémantique et de sa relation de satisfaction, la correction d'un système informatique par rapport à sa spécification peut être abordée de façon rigoureuse mais non formelle. La formalisation de cette preuve de correction, et donc sa certification mécanique, nécessite au préalable de traduire symboliquement les preuves sémantiques. En effet, manipuler des suites de symboles est la seule chose que sache faire un ordinateur. Il faut alors définir des règles de manipulation syntaxique des formules et comment elles peuvent conduire à obtenir une propriété. Cet ensemble de règles est appelé un système formel, ou bien un calcul.

Ces trois aspects (syntaxe, sémantique, et calcul) sont loin d'être indépendants. La syntaxe définit le domaine dans lequel travaillent la sémantique et le calcul en fixant le langage, c'est-à-dire les symboles utilisés

et les constructions autorisées. La sémantique, quant à elle, doit être capable de donner un sens à toutes les constructions définies par la syntaxe. Enfin, le calcul n'étant qu'une manipulation symbolique *a priori* sans aucun sens, il est indispensable de s'assurer que ce dernier représente aussi fidèlement que possible le monde sémantique choisi, que les manipulations symboliques sont en accord avec la sémantique. On parle alors de correction du calcul. Lorsqu'elle traduit totalement ce monde sémantique, on parle de complétude. Bien entendu, la première propriété est absolument nécessaire, car proposer un calcul sans définir en quel sens il est correct, ne pas fournir de sémantique, enlève de fait toute crédibilité à une logique. C'est seulement une fois établie la correction des règles du calcul par rapport à la sémantique que l'on peut faire confiance à un calcul. On peut alors s'appuyer sur une compréhension intuitive de la sémantique pour guider les preuves formelles. La complétude en revanche est seulement souhaitable car, s'il est intéressant de traduire totalement la sémantique par des règles de calcul, cela n'est pas toujours possible (une conséquence du théorème d'incomplétude de Gödel).

L'introduction d'un nouveau formalisme logique se fait toujours par la donnée de sa syntaxe, de sa sémantique et de son calcul. Même si la définition de ces différents éléments est propre à chaque langage, de nombreux points communs existent traduisant l'essence même d'un formalisme de spécification. En pratique, le volet syntaxique d'une logique repose sur la construction inductive de formules à partir de symboles de fonctions, de prédicats, de connecteurs propositionnels, de quantificateurs, etc. On remarque également qu'une logique contient des « parties fixes » (connecteurs propositionnels, quantificateurs, définition inductive des termes et des formules, etc.) et des « parties utilisateurs » que l'on introduit dans le but de résoudre un problème donné. Ces dernières sont classiquement appelées des signatures. Elles caractérisent les éléments de base spécifiques à un problème donné. Elles représentent donc l'interface du problème traité. Dans le cadre d'un système informatique, cette interface est importante. En effet, c'est souvent la seule partie visible d'un système pour les utilisateurs. De plus, un système informatique est amené à voir son comportement évoluer au long de son cycle de vie. Cette évolution se caractérise souvent en premier lieu par une modification de son interface (par ajout de fonctionnalités par exemple) puis par une modification de son comportement (par exemple en ajoutant des propriétés pour caractériser les nouvelles fonctionnalités). Pour ces raisons, la notion de signature, ainsi qu'un moyen de comparer ces signatures au niveau syntaxique puis au niveau sémantique, doivent apparaître dans le cadre d'une description générale de logiques dédiées à l'informatique. Pour mieux appréhender les liens entre les logiques et leur application à l'informatique, la théorie des institutions [GB92] a alors été introduite par Joseph Goguen et Rod Burstall, puis étendue aux preuves formelles par José Meseguer dans sa définition des logiques générales [Mes89].

Nous allons présenter tout d'abord le volet sémantique des logiques, au travers des institutions, ainsi que la notion de spécification. Nous présenterons ensuite le volet calculatoire en introduisant les logiques générales et les notions associées aux systèmes formels.

1 Institutions

1.1 Définitions élémentaires

Une institution capture de manière formelle le concept de système logique en donnant :

– une catégorie de signatures qui fournissent le vocabulaire nécessaire à la description d'un problème donné. Intuitivement, les signatures sont aux systèmes logiques ce que les interfaces sont aux pro-

grammes. Les morphismes de signatures, qui représentent les relations entre signatures, sont à rapprocher de la notion de modification d'interface (enrichissement, surcharge...) ;
- un moyen d'associer, à chaque signature, un ensemble de formules, qui sont les formules bien formées sur cette signature, auquel sont étendus les morphismes de signatures. Intuitivement, les morphismes de signatures transportés aux formules sont des fonctions permettant de renommer les symboles de la signature utilisés dans les formules ;
- un moyen d'associer, à chaque signature, une catégorie de modèles, à laquelle sont étendus les morphismes de signatures. Les morphismes de signature sont transportés aux modèles de façon à pouvoir restreindre un modèle associé à une signature en un modèle d'une signature plus pauvre ;
- un moyen d'associer, à chaque signature, une relation entre les modèles de cette signature et les formules sur cette signature appelée relation de satisfaction, permettant de dire qu'un modèle valide ou non une formule.

Une condition sur ces éléments impose la préservation de la relation de satisfaction par changement de signature, c'est-à-dire qu'un changement de signature selon un morphisme de signatures ne doit pas affecter la satisfaction des formules par les modèles. Par analogie avec les langages de programmation, cela traduit le fait qu'un programme peut faire quelque chose de plus simple que ce pour quoi il a été conçu. Par exemple, un programme permettant de manipuler des piles et en particulier d'en calculer la hauteur est capable de faire de l'arithmétique élémentaire sur les entiers.

Ainsi, sans détailler la structure des formules ni celle des modèles, une institution est dédiée à la description des rapports existant entre syntaxe et sémantique, c'est-à-dire entre les ensembles de formules définis sur des signatures et les catégories de modèles associées à celles-ci. C'est en quelque sorte une logique à laquelle manquent tous les mécanismes d'inférence, mais qui permet de capturer formellement la notion de système logique d'un point de vue théorie des modèles.

DÉFINITION I.1 — *Institution.*

Une *institution* est un quadruplet $(\mathit{Sig}, \mathit{For}, \mathit{Mod}, \models)$ où :
- Sig est une catégorie dont les objets sont appelés *signatures* ;
- $\mathit{For} : \mathit{Sig} \to \mathcal{S}et$ est un foncteur[1] qui à toute signature $\Sigma \in |\mathit{Sig}|$ associe[2] l'ensemble $\mathit{For}(\Sigma)$ des *formules* sur Σ ;
- $\mathit{Mod} : \mathit{Sig} \to \mathcal{C}at^{op}$ est un foncteur[3] qui à chaque signature $\Sigma \in |\mathit{Sig}|$ associe la catégorie $\mathit{Mod}(\Sigma)$ des *modèles* de Σ ;
- $\models = (\models_\Sigma)_{\Sigma \in |\mathit{Sig}|}$ est la famille de relations $\models_\Sigma$ sur $\mathit{Mod}(\Sigma) \times \mathit{For}(\Sigma)$ appelées *relations de satisfaction*.

De plus, pour tout morphisme de signatures $\sigma : \Sigma \to \Sigma'$, pour tout modèle $\mathcal{M}' \in |\mathit{Mod}(\Sigma')|$, pour toute formule $\varphi \in \mathit{For}(\Sigma)$, on a la propriété suivante, appelée *condition de satisfaction* :

$$\mathcal{M}' \models_{\Sigma'} \mathit{For}(\sigma)(\varphi) \iff \mathit{Mod}(\sigma)(\mathcal{M}') \models_\Sigma \varphi$$

Toutes les logiques que nous allons présenter par la suite vérifient cette condition. Toutefois, cette dernière ne sera pas nécessaire pour établir les résultats liés à la sélection de tests à partir de spécifications

[1] $\mathcal{S}et$ est la catégorie dont les objets sont les ensembles et dont les morphismes sont les fonctions.

[2] La notation $|\mathit{Sig}|$ désigne la classe des objets de la catégorie Sig, voir l'annexe A.

[3] $\mathcal{C}at$ est la catégorie dont les objets sont des catégories et dont les morphismes sont les foncteurs. $\mathcal{C}at^{op}$ est la catégorie duale de $\mathcal{C}at$, voir l'annexe A.

axiomatiques. Nous nous servirons uniquement du cadre unifié sous lequel les institutions permettent de présenter l'aspect sémantique des logiques.

La définition d'une institution n'implique aucune contrainte sur la construction et la forme des formules. Dans la pratique cependant, la structure des formules n'est pas quelconque. Comme nous le verrons dans les deux instances d'institutions que nous présentons aux chapitres suivants, la définition des formules pour une logique particulière repose sur la donnée d'un ensemble d'éléments de base (les formules atomiques) et d'opérateurs sur cet ensemble (connecteurs logiques, quantificateurs, modalités, etc.). Bien que le cadre abstrait des institutions ne permette pas de faire référence explicitement à ces opérateurs, il est possible de les définir de façon interne à une institution donnée. Les résultats dont nous aurons besoin nécessitant uniquement l'existence de la négation au sein de l'institution considérée, nous ne donnons ici que la définition de la négation sémantique.

DÉFINITION I.2 — *Négation sémantique.*

Soit une institution $\mathcal{I} = (Sig, For, Mod, \models)$. Soit Σ une signature. Une formule $\varphi' \in For(\Sigma)$ est la *négation sémantique* d'une formule $\varphi \in For(\Sigma)$ si et seulement si

$$\forall \mathcal{M} \in |\, Mod(\Sigma)|, \mathcal{M} \models_\Sigma \varphi' \Leftrightarrow \mathcal{M} \not\models_\Sigma \varphi$$

On dira que $\mathcal{I}$ *possède la négation* si pour toute signature Σ, toute formule $\varphi \in For(\Sigma)$ possède une négation sémantique.

Dans le reste de cette section, on se place dans une institution $\mathcal{I} = (Sig, For, Mod, \models)$ fixée possédant la négation.

Pour une formule φ donnée, il est possible de considérer l'ensemble des modèles de Σ qui valident cette formule. On dit alors d'un modèle appartenant à cet ensemble qu'il est un modèle de φ. Plus généralement, étant donné un ensemble de formules, on définit un modèle associé à cet ensemble de la façon suivante.

DÉFINITION I.3 — *Modèle d'un ensemble de formules.*

Soit Σ une signature de $|\, Sig\,|$ et $\Psi \subseteq For(\Sigma)$ un ensemble de formules sur Σ. Un *modèle de* Ψ est un Σ-modèle $\mathcal{M}$ qui valide toutes les formules de Ψ, c'est-à-dire tel que

$$\forall \varphi \in \Psi, \mathcal{M} \models \varphi$$

On note $Mod(\Psi)$ la sous-catégorie pleine de $Mod(\Sigma)$ dont les objets sont les modèles de Ψ.

Étant donné un ensemble de formules Ψ, un modèle de Ψ est donc un modèle qui, en particulier, valide les formules de Ψ, mais en général, valide aussi d'autres formules. Parmi celles-ci, il en existe qui sont également validées par d'autres modèles de Ψ. Si ces formules sont vérifiées par tout modèle vérifiant les formules de Ψ, on dit que ce sont des conséquences sémantiques de Ψ.

DÉFINITION I.4 — *Conséquence sémantique d'un ensemble de formules.*

Soit Σ une signature de $|\, Sig\,|$. Soit $\Psi \subseteq For(\Sigma)$ un ensemble de formules sur Σ. L'ensemble des *conséquences sémantiques* de Ψ, noté $\Psi^\bullet$, est l'ensemble des formules validées par tous les modèles de Ψ :

$$\Psi^\bullet = \{\varphi \in For(\Sigma) \mid \forall \mathcal{M} \in Mod(\Psi), \mathcal{M} \models \varphi\}$$

Si $\varphi \in \Psi^\bullet$, on note $\Psi \models \varphi$ la relation de conséquence sémantique.

Un ensemble de formules est alors dit consistant s'il existe des formules qui ne sont pas des conséquences sémantiques de cet ensemble.

DÉFINITION I.5 — *Consistance d'un ensemble de formules.*
Soit Σ une signature. Un ensemble de formules $\Psi \subseteq For(\Sigma)$ est dit *consistant* si $\Psi^\bullet \neq For(\Sigma)$, et *inconsistant* dans le cas contraire.

Un ensemble de formules inconsistant peut donc avoir pour conséquence sémantique à la fois une formule et sa négation. Un modèle ne pouvant valider qu'une seule de ces deux formules, un tel ensemble de formules ne peut pas avoir de modèles.

PROPOSITION I.1. [Bar05] *Soit Σ une signature telle que $For(\Sigma) \neq \emptyset$. Un ensemble de formules $\Psi \subseteq For(\Sigma)$ est consistant si et seulement si $Mod(\Psi) \neq \emptyset$.*

Une classe de modèles d'une même signature peut être caractérisé par l'ensemble des formules qui sont validées par tous les modèles de cette classe. Ces formules forment ce qu'on appelle une théorie.

DÉFINITION I.6 — *Théorie.*
Soit Σ une signature. Soit $\mathscr{M} \subseteq |Mod(\Sigma)|$ un ensemble de modèles de Σ. La *théorie de* $\mathscr{M}$, notée $Th(\mathscr{M})$, est l'ensemble de formules suivant :

$$Th(\mathscr{M}) = \{\varphi \in For(\Sigma) \mid \forall \mathcal{M} \in \mathscr{M}, \mathcal{M} \models_\Sigma \varphi\}$$

On remarque que pour un ensemble de formules Ψ, l'ensemble des conséquences sémantiques de Ψ est la théorie de la classe des modèles de Ψ :

$$\Psi^\bullet = Th(Mod(\Psi))$$

On dit qu'une théorie est complète si c'est le plus grand ensemble de formules qui soit consistant. Intuitivement, tout formule n'appartenant pas à cette théorie est la négation d'une formule de la théorie.

DÉFINITION I.7 — *Complétude d'une théorie.*
Soit Σ une signature. Soit $T \subseteq For(\Sigma)$ une théorie. La théorie T est dite *complète* ou *maximale consistante* si et seulement si, pour toute formule $\varphi \in For(\Sigma)$, une des deux conditions suivantes est vérifiée :
- $T \cup \{\varphi\}$ est inconsistante ;
- $\varphi \in T$.

En particulier, l'ensemble des formules validées par un modèle donné est une théorie complète.

PROPOSITION I.2. [Bar05] *Soit Σ une signature telle que $For(\Sigma) \neq \emptyset$. Pour tout $\mathcal{M}$ modèle de Σ, $Th(\mathcal{M})$ est une théorie complète.*

Pour une signature donnée, il existe une multitude de modèles entre lesquels il n'est possible d'établir de distinction qu'au travers des formules qu'ils valident. Deux modèles d'une même signature qui valident exactement les mêmes formules sont alors dits élémentairement équivalents.

DÉFINITION I.8 — *Équivalence élémentaire de modèles.*

Soit Σ une signature de $|\ Sig\ |$. Soient $\mathcal{M}, \mathcal{M}' \in Mod(\Sigma)$ deux modèles de Σ et $\Psi \subseteq For(\Sigma)$. On dit que $\mathcal{M}$ et $\mathcal{M}'$ sont *élémentairement équivalents par rapport à* Ψ, noté $\mathcal{M} \equiv_\Psi \mathcal{M}'$, si et seulement s'ils valident exactement les mêmes formules de Ψ :

$$\forall \varphi \in \Psi, \mathcal{M} \models \varphi \Leftrightarrow \mathcal{M}' \models \varphi$$

Lorsque $\Psi = For(\Sigma)$, on dit que $\mathcal{M}$ et $\mathcal{M}'$ sont *élémentairement équivalents*, noté $\mathcal{M} \equiv \mathcal{M}'$.

Cette notion s'exprime également au niveau des formules. Deux formules représentent la même propriété si elles sont validées exactement par les mêmes modèles.

DÉFINITION I.9 — *Équivalence élémentaire de formules.*

Soient φ et ψ deux formules sur Σ. On dit que φ et ψ sont *élémentairement équivalentes*, noté $\varphi \equiv \psi$, si et seulement si elles admettent exactement les mêmes modèles :

$$\forall \mathcal{M} \in Mod(\Sigma), \mathcal{M} \models \varphi \Leftrightarrow \mathcal{M} \models \psi$$

La notion d'équivalence élémentaire de modèles nous permet d'exprimer la propriété suivante sur les modèles d'une théorie complète.

THÉORÈME I.1. [Bar05] *Soit Σ une signature. Soit $T \subseteq For(\Sigma)$ une théorie complète. Soit Ψ l'ensemble des formules closes sur Σ. Alors pour tous $\mathcal{M}$, $\mathcal{M}'$ modèles de T non vides, $\mathcal{M} \equiv_\Psi \mathcal{M}'$.*

1.2 Spécifications

Une fois fixée une logique sous la forme d'une institution, il est possible de l'utiliser comme formalisme de spécification. Une spécification axiomatique, est alors composée de l'ensemble des symboles spécifiques au système que l'on veut décrire (une signature de la logique considérée) et d'un ensemble de formules construites à partir de cette signature : les axiomes.

DÉFINITION I.10 — *Spécification.*

Soit une institution $(Sig, For, Mod, \models)$. Une *spécification* Sp est un couple (Σ, Ax) où Σ est une signature de $|\ Sig\ |$ et $Ax \in For(\Sigma)$ est un ensemble de formules appelées *axiomes*.

Les axiomes décrivant les propriétés attendues du système, il est nécessaire que les modèles associés à la spécification, étant des représentations du système, valident ces axiomes. Un modèle d'une spécification est donc un modèle de la signature de cette spécification qui valide tous les axiomes.

DÉFINITION I.11 — *Modèle d'une spécification.*

Si $Sp = (\Sigma, Ax)$ est une spécification, on appelle *modèle de* Sp un modèle de l'ensemble Ax des axiomes. On note $Mod(Sp)$ la sous-catégorie pleine de $Mod(\Sigma)$ dont les objets sont les modèles de Sp.

Les formules qui sont validées par tous les modèles d'une spécification seront d'une importance particulière dans la suite de cette thèse. Elles représentent en effet toutes les propriétés que le système spécifié valide dès lors qu'il valide les axiomes de la spécification, ce système pouvant être vu comme un modèle de la spécification.

DÉFINITION I.12 — *Conséquences sémantiques d'une spécification.*

Si $Sp = (\Sigma, Ax)$ est une spécification, l'ensemble des *conséquences sémantiques de* Sp, noté $Sp^\bullet$, est l'ensemble des conséquences sémantiques de l'ensemble Ax des axiomes. Si $\varphi \in Sp^\bullet$, on note $Sp \models \varphi$ la relation de conséquence sémantique.

Pour s'assurer qu'il est possible de concevoir un système implantant une spécification, il faut que celle-ci admette au moins un modèle, c'est-à-dire que l'ensemble de ces axiomes soit consistant. On dit alors que la spécification elle-même est consistante.

DÉFINITION I.13 — *Consistance d'une spécification.*

Une spécification $Sp = (\Sigma, Ax)$ est dite *consistante* si seulement si Ax est consistant.

2 Logiques générales

2.1 Définitions élémentaires

Un système d'inférence capture abstraitement les aspects syntaxiques du raisonnement dans une logique. Comme une institution, il est constitué d'une catégorie de signatures et d'un foncteur associant à chaque signature un ensemble de formules, mais la sémantique, au travers du foncteur donnant une catégorie de modèles pour une signature et de la relation de satisfaction, est remplacé ici par le raisonnement déductif, représenté par une relation entre ensembles de formules et formules, appelée relation d'inférence. Intuitivement, cette relation exprime la possibilité de déduire une propriété d'un ensemble d'hypothèses par une certaine méthode de raisonnement exprimée uniquement en termes de manipulations syntaxiques.

DÉFINITION I.14 — *Système d'inférence.*

Un *système d'inférence* est un triplet $(Sig, For, \vdash)$ où :

- Sig est une catégorie dont les objets sont appelés *signatures* ;
- $For : Sig \to \mathcal{S}et$ est un foncteur qui à toute signature $\Sigma \in |Sig|$ associe l'ensemble $For(\Sigma)$ des *formules* sur Σ ;
- $\vdash = (\vdash_\Sigma)_{\Sigma \in |Sig|}$ est la famille de relations $\vdash_\Sigma$ sur $\mathcal{P}(For(\Sigma)) \times For(\Sigma)$ appelées *relations d'inférence*.

Ce raisonnement peut posséder différentes propriétés, listées ci-après.

DÉFINITION I.15 — *Propriétés des systèmes d'inférence.*

Soit $(Sig, For, \vdash)$ un système d'inférence.

La relation $\vdash$ est *réflexive* si pour toute signature Σ de $|\, Sig \,|$, pour toute formule $\varphi \in For(\Sigma)$,

$$\{\varphi\} \vdash_\Sigma \varphi$$

La relation $\vdash$ est *transitive* si pour toute signature Σ de $|\, Sig \,|$, pour tous ensembles [4] de formules $\Gamma, \Gamma' \subseteq For(\Sigma)$, pour toute formule $\varphi \in For(\Sigma)$,

$$\Gamma \vdash_\Sigma \Gamma' \wedge \Gamma \cup \Gamma' \vdash_\Sigma \varphi \Rightarrow \Gamma \vdash_\Sigma \varphi$$

La relation $\vdash$ est *monotone* si pour toute signature Σ de $|\, Sig \,|$, pour tous ensembles de formules $\Gamma, \Gamma' \subseteq For(\Sigma)$, pour toute formule $\varphi \in For(\Sigma)$,

$$\Gamma \vdash_\Sigma \varphi \wedge \Gamma \subseteq \Gamma' \Rightarrow \Gamma' \vdash_\Sigma \varphi$$

La relation $\vdash$ a la propriété de $\vdash$*-translation* si pour toutes signatures Σ et Σ' de $|\, Sig \,|$, pour tout ensemble de formules $\Gamma \subseteq For(\Sigma)$, pour toute formule $\varphi \in For(\Sigma)$, pour tout morphisme [5] de signatures $\sigma : \Sigma \to \Sigma'$,

$$\Gamma \vdash_\Sigma \varphi \Rightarrow For(\sigma)(\Gamma) \vdash_{\Sigma'} For(\sigma)(\varphi)$$

La réflexivité exprime le fait qu'on peut toujours déduire une formule à partir d'elle-même. La transitivité traduit la possibilité de mener des preuves en passant par des lemmes intermédiaires : pour prouver φ à partir de Γ, on commence par prouver un ensemble de lemmes Γ', qu'on utilise pour prouver φ. Si la relation d'inférence est monotone, cela signifie que si on peut prouver φ à partir d'un ensemble d'hypothèses Γ, alors on peut également prouver φ en utilisant plus d'hypothèses. La propriété de $\vdash$-translation est le pendant syntaxique de la condition de satisfaction. Elle impose la préservation de la relation d'inférence par enrichissement de signature : si Γ et φ sont des hypothèses et formules construites sur une certaine signature, s'il est possible de prouver φ à partir de Γ, alors il est possible de traduire Γ et φ dans une signature plus riche et de mener la preuve dans cette signature.

Lorsque les deux aspects du raisonnement, sémantique et syntaxique, sont présents dans une logique, il faut établir une certaine correspondance entre eux, de manière à ce que le raisonnement syntaxique reflète le mieux possible le raisonnement qu'il est possible de mener au niveau sémantique. De façon indispensable, il faut que les propriétés qu'il est possible de déduire par le raisonnement syntaxique soient vérifiées dans le monde mathématique représentatif du phénomène décrit, c'est ce qu'on appelle la correction de la relation d'inférence calcul par rapport à la relation de satisfaction. Si de plus, tout ce qui est vrai dans ce monde mathématique trouve une preuve syntaxique, on dit que la relation d'inférence est complète.

DÉFINITION I.16 — *Correction et complétude du calcul par rapport à la sémantique.*

Soit $(Sig, For, Mod, \models)$ une institution et $(Sig, For, \vdash)$ un système d'inférence. Soit Σ une signature de $|\, Sig \,|$.

La relation d'inférence $\vdash_\Sigma$ est dite *correcte* par rapport à la relation de satisfaction $\models_\Sigma$ si et seulement si

$$\forall \Gamma \subseteq For(\Sigma), \forall \varphi \in For(\Sigma), \Gamma \vdash_\Sigma \varphi \Rightarrow \Gamma \models_\Sigma \varphi$$

[4] Par abus de notation, on note $\Gamma \vdash_\Sigma \Gamma'$ si $\Gamma \vdash_\Sigma \varphi$ pour toute formule φ de Γ'.

[5] On note $For(\sigma)(\Gamma)$ l'ensemble $\{For(\sigma)(\varphi) \mid \varphi \in \Gamma\}$.

Elle est dite *complète* si et seulement si

$$\forall \Gamma \subseteq For(\Sigma), \forall \varphi \in For(\Sigma), \Gamma \models_\Sigma \varphi \Rightarrow \Gamma \vdash_\Sigma \varphi$$

Une logique est alors la combinaison d'une institution et d'un système d'inférence, qui vérifie certaines propriétés, comme la correction de la relation d'inférence par rapport à la relation de satisfaction.

DÉFINITION I.17 — *Logique générale.*

Une logique générale est un quintuplet $(Sig, For, Mod, \models, \vdash)$ où :

- $(Sig, For, Mod, \models)$ est une institution ;
- $(Sig, For, \vdash)$ est un système d'inférence tel que la relation d'inférence $\vdash$ a les propriétés de réflexivité, transitivité, monotonie et $\vdash$-translation,

tel que la relation d'inférence $\vdash$ est correcte par rapport à la relation de satisfaction $\models$.

Les logiques générales et les propriétés qui leur sont associées sont des guides qui permettent de s'abstraire des contingences calculatoires ou de modélisation. En effet, dans ce qui précède, rien n'impose que les signatures soient des ensembles de symboles, ni que les formules soient des suites de symboles, ou que les inférences soient obtenues par des techniques quelconques de raisonnement effectif. L'aspect « théorie de la preuve » n'est pris en compte que de façon très succincte dans les logiques générales. Pour l'incarner par de véritables manipulations symboliques, il est nécessaire d'ajouter des contraintes au cadre des logiques générales : on va imposer que la relation d'inférence pour chaque signature soit générée inductivement à partir d'un ensemble de règles, un système formel.

2.2 Systèmes formels

Les systèmes formels, également appelés calculs, permettent de représenter des mécanismes de production d'énoncés, les formules, ayant une rigueur syntaxique. Ces mécanismes sont représentés de façon abstraite par des règles de manipulation de symboles. Ces manipulations étant symboliques, les productions d'énoncés peuvent alors être contrôlées par ordinateur. Le principe sous-jacent est le principe d'induction. Dans ce contexte, les règles de production sont appelées règles d'inférence et modélisent les étapes autorisées du raisonnement.

DÉFINITION I.18 — *Système formel.*

Un *système formel*, également appelé *calcul*, est un triplet (A, F, R) où :

- A est un ensemble, appelé *alphabet*, dont les éléments sont appelés *symboles* ;
- F est un sous-ensemble de A^* dont [6] les éléments sont appelés *formules bien formées sur* A ;
- R est un ensemble fini de relations n-aires sur F appelées *règles d'inférence*.

L'alphabet A contient ici à la fois les symboles fixes du langage considéré (les connecteurs booléens, les quantificateurs, *etc.*), les symboles de la signature et les symboles dédiés à la construction des formules (les parenthèses, la virgule, *etc.*). Un système formel ne pose aucune contrainte syntaxique sur les formules. Cependant, dans la pratique, l'ensemble F n'est pas quelconque, il est défini inductivement, par une grammaire

[6] A^* désigne l'ensemble des mots finis sur A.

ou tout autre algorithme de reconnaissance de formules bien formées. L'ensemble F contient les énoncés dont la syntaxe est correcte, sans aucune notion de véracité.

Notation. Étant donné un système formel, une règle d'inférence définit un ensemble d'étapes élémentaires de preuve. Soit $r \subseteq For(\Sigma)^n$ une règle d'inférence. Soient $\varphi_1, \ldots, \varphi_n \in For(\Sigma)$. Si $(\varphi_1, \ldots, \varphi_n) \in r$, on dit que le n-uplet $(\varphi_1, \ldots, \varphi_n)$ est une *instance* de r. Les formules $\varphi_1, \ldots, \varphi_{n-1}$ sont appelées les *prémisses* de cette instance et φ_n sa *conclusion*. Une instance de la règle r sera notée

$$\frac{\varphi_1 \ldots \varphi_{n-1}}{\varphi_n} r$$

Les instances des règles telles que $n = 1$ sont appelées *axiomes*.

À partir d'un ensemble de postulats, ou hypothèses, qui sont des formules, les systèmes formels permettent d'accéder à d'autres formules, celles qu'il est possible de déduire de ces hypothèses par application d'instances de règles d'inférence. On appelle déduction la trace des formules intermédiaires utilisées pour prouver une formule.

DÉFINITION I.19 — *Déduction.*

Soit un calcul $C = (A, F, R)$. Soit $\Gamma \subseteq F$ un ensemble de formules. Une *déduction* dans C à partir des hypothèses Γ est une séquence $\varphi_1, \ldots, \varphi_k$ de formules de F telle que pour tout i, $1 \leq i \leq k$:

- soit φ_i est une formule de Γ ;
- soit il existe une instance $\frac{\psi_1 \ldots \psi_n}{\psi} r$ d'une règle de R dont la conclusion est égale à la formule φ_i et telle que chacune des prémisses ψ_m de r, $1 \leq m \leq n$, est égale à l'une des formules ψ_l de la déduction qui précède φ_i, c'est-à-dire telle que $1 \leq l \leq i - 1$.

La dernière formule d'une déduction à partir d'une ensemble Γ est dite inférée par Γ. Il est alors possible d'en déduire une relation binaire entre les ensembles de formules (les hypothèses) et les formules qui en sont déduites, qui est appelée relation d'inférence.

DÉFINITION I.20 — *Relation d'inférence.*

Soit un calcul $C = (A, F, R)$. Soient un ensemble de formules $\Gamma \subseteq F$ et une formule $\varphi \in F$. On dit que Γ *infère* φ, noté $\Gamma \vdash_C \varphi$, s'il existe dans C une déduction $\varphi_1, \ldots, \varphi_k, \varphi$ à partir de Γ.

On retrouve ainsi la notion de relation d'inférence de la définition I.14. En effet, si $C = (A, F, R)$ est un système formel, on remarque que le triplet $(Sig, For, \vdash)$ où :

- Sig ne contient que l'alphabet A ;
- For est l'application qui à l'unique signature A associe l'ensemble F ;
- $\vdash_A = \vdash_C$,

forme un système d'inférence. Un système formel définit alors un système d'inférence dans lequel il n'y a aucun choix possible de signature, un système formel se focalisant sur une démarche systématique de preuve particulière. Il est facile de montrer que le système d'inférence obtenu possède les propriétés de réflexivité,

de transitivité et de monotonie. La $\vdash$-translation est trivialement vérifiée puisque la catégorie Sig est réduite à l'alphabet A.

Une autre façon d'obtenir des énoncés à l'aide des règles d'inférence est le schéma d'induction ci-dessous, qui a pour base l'ensemble d'hypothèses et les axiomes, et dont les règles de production sont les règles d'inférence du système formel. Il permet de construire ce qu'on appelle des théorèmes.

DÉFINITION I.21 — *Théorème.*

Soit un calcul $C = (A, F, R)$. Soit Γ un ensemble de formules de F. L'ensemble des *théorèmes* déduits de Γ à l'aide de C, noté $Th_C(\Gamma)$, est l'ensemble de formules défini inductivement de la façon suivante :

- toute instance d'un axiome et toute formule de Γ est un théorème de $Th_C(\Gamma)$;
- pour toute instance d'une règle d'inférence, si toutes ses prémisses sont des théorèmes de $Th_C(\Gamma)$, alors sa conclusion est un théorème de $Th_C(\Gamma)$.

Lorsque Γ est l'ensemble vide, les formules obtenues sont appelées *tautologies* et forment l'ensemble Th_C.

Les théorèmes sont en fait exactement les formules pour lesquelles il existe une déduction.

PROPOSITION I.3. *Soient $C = (A, F, R)$ un système formel et Γ un ensemble de formules. Pour toute formule $\varphi \in F$, on a*

$$\Gamma \vdash_C \varphi \iff \varphi \in Th_C(\Gamma)$$

Preuve. Soit une formule $\varphi \in F$. On suppose que $\Gamma \vdash_C \varphi$. Montrons que $\varphi \in Th_C(\Gamma)$ par récurrence sur la longueur de la déduction de φ.

Cas de base. Si la déduction de φ est de longueur 1, alors soit φ est dans Γ, soit φ est une instance d'axiome de R. Par définition de l'ensemble $Th_C(\Gamma)$, φ est donc un théorème de $Th_C(\Gamma)$.

Cas général. Si la déduction de φ est de longueur n, alors il existe une instance $\dfrac{\varphi_1 \dots \varphi_n}{\varphi} r$ d'une règle de R telle que pour tout i, $1 \leq i \leq n$, la déduction de φ_i est de longueur strictement inférieure à n. Par hypothèse de récurrence, chaque φ_i est donc un théorème. Par définition de l'ensemble $Th_C(\Gamma)$, φ est donc également un théorème de $Th_C(\Gamma)$.

On suppose maintenant que $\varphi \in Th_C(\Gamma)$. Montrons que $\Gamma \vdash_C \varphi$ par induction sur l'ensemble des théorèmes.

Cas de base. Si φ est une instance d'axiome de R ou une formule de Γ, alors une déduction pour φ est la séquence φ.

Cas général. Si φ est la conclusion d'une instance $\dfrac{\varphi_1 \dots \varphi_n}{\varphi} r$ d'une règle de R telle que chaque φ_i, pour $1 \leq i \leq n$, est un théorème, alors par hypothèse d'induction, chaque φ_i possède une déduction $D_i = \psi_1^i, \dots, \psi_{n_i}^i, \varphi_i$. La séquence $D_1, \dots, D_n, \varphi$ est donc une déduction de φ.

□

Un théorème de $Th_C(\Gamma)$ s'obtient donc à partir des formules de Γ par applications successives des règles d'inférence de C, c'est-à-dire par empilement de ces règles. De ce point de vue, l'obtention d'un théorème de $Th_C(\Gamma)$ peut être mise sous la forme d'un arbre appelé arbre de preuve.

DÉFINITION I.22 — *Arbre de preuve*.

Soit un calcul $C = (A, F, R)$. Un *arbre de preuve* d'un théorème de C est un arbre dont :

- les feuilles sont des instances d'axiomes ;
- les nœuds internes sont des théorèmes intermédiaires tels que le n-uplet constitué des fils d'un nœud et de ce nœud est une instance d'une règle d'inférence ;
- la racine est le théorème.

D'après la proposition I.3, une déduction n'est donc qu'une mise à plat d'un arbre de preuve. À chaque déduction est donc associé un arbre de preuve, qui n'est pas unique en général. Réciproquement, il est possible d'associer à chaque arbre de preuve autant de déductions qu'il y a de parcours en profondeur de l'arbre.

Chapitre II

Logique du premier ordre

Le concept de spécification algébrique a émergé dans le milieu des années 70 avec les travaux de Stephen Zilles [Zil74, LZ74], de John Guttag [Gut75] et du groupe ADJ formé de Joseph Goguen, James Thatcher, Eric Wagner et Jesse Wright [GTWW75]. L'idée fondamentale de cette approche consiste à décrire des structures de données uniquement à l'aide des noms des différents ensembles de données, des noms des fonctionnalités de base et de leurs propriétés caractéristiques. Ces propriétés sont décrites par des formules dont les éléments de base sont des équations. Cette idée, qui avait déjà été mise en avant dans les années 20 dans les algèbres universelles de Garrett Birkhoff [Bir46], provient du constat que les structures algébriques usuelles manipulées en mathématiques, telles que les monoïdes, les groupes, les anneaux, les espaces vectoriels ou les treillis, sont naturellement définies par un ensemble d'axiomes équationnels. L'associativité, la commutativité ou la distributivité par exemple, sont des propriétés qui s'expriment naturellement sous la forme d'équations. Selon le choix des opérations et du type d'axiomes, toutes les structures algébriques mentionnées plus haut peuvent être caractérisées. L'idée de définir n'importe quelle structure algébrique en termes d'opérations et d'équations a été reprise par Stephen Zilles [Zil74] afin de spécifier également les types de données abstraits comme les piles, les files, les arbres ou les listes.

En effet, un type de données, de la même manière que les structures mathématiques mentionnées ci-dessus, est un ensemble de valeurs représentant ces données, muni d'éléments distingués ainsi que d'opérations agissant sur ces valeurs et dédiées à la recherche et à la mise à jour des données. Cependant, les types de données diffèrent de ces structures mathématiques en ce que l'ensemble des valeurs sous-jacent est construit inductivement. C'est pour cette raison qu'il est possible de programmer des fonctions sur ces types de données. Un type de données peut alors être vu comme une structure algébrique, c'est-à-dire un ensemble d'éléments muni de lois internes et externes satisfaisant certaines propriétés. C'est cette idée qui a été développée dans les travaux du groupe ADJ [GTWW75].

Une spécification algébrique d'un type de données abstrait est alors une description de ce type de données, au travers de l'ensemble de ses éléments, l'ensemble des opérations sur ces éléments, ainsi que l'ensemble des équations décrivant les propriétés de ces opérations. Le formalisme utilisé pour cette description est ce qu'on appelle la logique équationnelle. Cette logique est appropriée pour spécifier un très grand nombre des structures algébriques et des structures de données usuelles. Cependant, les structures de données sont

souvent munies de relations qui permettent par exemple de comparer des éléments entre eux. Il est possible de spécifier ces relations par des fonctions booléennes et ainsi de rester dans un cadre équationnel, ou alors de les spécifier directement à l'aide de ce qu'on appelle des prédicats. Il faut pour cela utiliser un formalisme plus riche que la logique équationnelle, la logique des prédicats du premier ordre. La logique équationnelle devient alors une restriction de la logique des prédicats du premier ordre, munie de l'égalité pour seul prédicat. C'est la démarche que nous allons suivre dans ce chapitre : nous présenterons tout d'abord la logique des prédicats du premier ordre, puis la logique équationnelle comme une restriction de la première.

1 Logique des prédicats du premier ordre

La présentation de la logique du premier ordre que nous allons donner suit le schéma donné par la définition I.17 des logiques générales. Vont être présentés successivement la catégorie des signatures du premier ordre, l'ensemble des formules associé à une signature, la catégorie des modèles associée à une signature, la relation de satisfaction et enfin le système d'inférence défini par le calcul des séquents pris comme système formel.

1.1 Syntaxe

Signatures. Une signature du premier ordre comprend les noms des différents types de données nécessaires à la description du type de données visé, un ensemble d'opérations sur ces types de données, et un ensemble de prédicats représentant les propriétés qu'il est possible de connaître sur ces types de données.

DÉFINITION II.1 — *Signature.*

Une *signature* Σ est un triplet (S, F, R) où :

- S est un ensemble dont les éléments sont appelés *sortes* ;
- F est un ensemble dont les éléments sont des noms d'*opérations* où chaque nom f est muni d'une arité dans $S^* \times S$;
- R est un ensemble dont les éléments sont des noms de *prédicats* où chaque nom r est muni d'une arité dans S^+.

On note $f : s_1 \times \ldots \times s_n \to s$ une opération f d'arité $(s_1 \ldots s_n, s)$ et $r : s_1 \times \ldots \times s_n$ un prédicat r d'arité $s_1 \ldots s_n$. Une opération $f : \to s$ est appelée une *constante*.

EXEMPLE II.1 — *Monoïde.*

Un monoïde est une structure algébrique qui consiste en un ensemble muni d'un élément distingué appelé élément neutre et d'une loi de composition interne associative. La signature d'un monoïde est donc composée d'une unique sorte $\acute{E}lem$ représentant le type des éléments de l'ensemble et deux opérations : une constante e de sorte $\acute{E}lem$ désignant l'élément neutre et une opération binaire sur $\acute{E}lem$ notée $*$. On aura également besoin d'un prédicat d'égalité sur les éléments de l'ensemble sous-jacent au monoïde pour exprimer les propriétés de cette structure.

sort $\acute{E}lem$

ops $e :\rightarrow \acute{E}lem$

$\_ * \_ : \acute{E}lem \times \acute{E}lem \rightarrow \acute{E}lem$

pred $= : \acute{E}lem \times \acute{E}lem$

◇

EXEMPLE II.2 — *Groupes.*

Un groupe est un monoïde dont tous les éléments sont inversibles. Un élément x est inversible s'il existe un (unique) élément y tel que $x * y = y * x = e$, où e est l'élément neutre du monoïde. Il y a plusieurs manières d'exprimer cette propriété supplémentaire au monoïde. Il est possible de définir une fonction $\_^{-1}$ qui à chaque élément associe son inverse.

sort $\acute{E}lem$

ops $e :\rightarrow \acute{E}lem$

$\_ * \_ : \acute{E}lem \times \acute{E}lem \rightarrow \acute{E}lem$

$\_^{-1} : \acute{E}lem \rightarrow \acute{E}lem$

pred $= : \acute{E}lem \times \acute{E}lem$

Il est également possible de définir l'inverse comme étant une relation binaire inv symétrique. La signature associée est alors la suivante.

sort $\acute{E}lem$

ops $e :\rightarrow \acute{E}lem$

$\_ * \_ : \acute{E}lem \times \acute{E}lem \rightarrow \acute{E}lem$

preds $= : \acute{E}lem \times \acute{E}lem$

$inv : \acute{E}lem \times \acute{E}lem$

◇

EXEMPLE II.3 — *Graphes.*

Un graphe (non orienté) est une structure formée d'un ensemble de sommets et d'un ensemble d'arêtes reliant ces sommets. Il est possible de le voir comme la représentation d'une relation binaire quelconque entre des éléments représentés par les sommets. On peut spécifier la structure de graphe de différentes manières. On choisit ici de représenter un graphe comme un ensemble de sommets et une relation binaire sur ces sommets. On dispose alors de deux sortes : *Sommet* et *Graphe*. Un graphe vide est représenté par l'ensemble de sommets vide. On construit un graphe à partir d'un sommet grâce à l'opération singleton puis en prenant l'union de deux graphes. Le prédicat $ar\hat{e}te$ indique si deux sommets sont reliés par une arête dans un graphe donné, tandis que le prédicat *chemin* indique si deux sommets sont reliés par un chemin. On dispose du prédicat d'appartenance $\in$ d'un sommet à un graphe et du prédicat d'égalité entre graphes. On a ensuite quelques prédicats permettant de spécifier des propriétés particulières sur la structure des graphes : un arbre peut n'avoir pas de sommet isolé, être complet, connexe, acyclique, être un arbre ou bien encore être un arbre couvrant.

sorts $Sommet, Graphe$

ops $\emptyset :\rightarrow Graphe$

$\{\_\} : Sommet \rightarrow Graphe$

$\_ \cup \_ : Graphe \times Graphe \rightarrow Graphe$

preds $arête\_(\_,\_) : Graphe \times Sommet \times Sommet$

$chemin\_(\_,\_) : Graphe \times Sommet \times Sommet$

$\_ \in \_ : Sommet \times Graphe$

$\_ \equiv \_ : Graphe \times Graphe$

$complet : Graphe$

$connexe : Graphe$

$non\_isolé : Graphe$

$acyclique : Graphe$

$arbre : Graphe$

$couvrant : Graphe \times Graphe$

◇

Les liens entre les signatures sont exprimés à l'aide des morphismes de signatures. Un morphisme entre deux signatures $\Sigma = (S, F, R)$ et $\Sigma' = (S', F', R')$ associe à chaque sorte de S une sorte de S', à chaque opération de F une opération de F' d'arité correspondante et à chaque prédicat de R un prédicat d'arité correspondante.

DÉFINITION II.2 — *Morphisme de signatures.*

Soient $\Sigma = (S, F, R)$ et $\Sigma' = (S', F', R')$ deux signatures. Un *morphisme de signatures* $\eta : \Sigma \rightarrow \Sigma'$ est défini par :

- une application $\eta_S : S \rightarrow S'$;
- une application $\eta_F : F \rightarrow F'$ telle que pour tout $f : s_1 \times \ldots \times s_n \rightarrow s \in F$, $\eta_F(f) : \eta_S(s_1) \times \ldots \times \eta_S(s_n) \rightarrow \eta_S(s)$;
- une application $\eta_R : R \rightarrow R'$ telle que pour tout $r : s_1 \times \ldots \times s_n \in R, \eta_R(r) : \eta_S(s_1) \times \ldots \times \eta_S(s_n)$.

DÉFINITION II.3 — *Catégorie des signatures.*

La *catégorie des signatures*, notée Sig, est la catégorie dont les objets sont les signatures et dont les morphismes sont les morphismes de signatures.

Formules. Une fois fixé l'ensemble des symboles associés à un type de données, il est possible de les composer de manière à former des expressions visant à décrire les propriétés de ce type de données. Il est nécessaire pour cela de disposer d'un ensemble de valeurs génériques, appelées variables, à partir desquelles construire de façon incrémentale ces propriétés. La définition de cet ensemble de variables repose sur la notion d'ensemble multi-sorte.

DÉFINITION II.4 — *S-ensemble.*

Soit un ensemble S. Un *S-ensemble* A est un ensemble muni d'une partition indexée par S :

$$A = \coprod_{s \in S} A_s$$

DÉFINITION II.5 — *Ensemble de variables.*

Soit $\Sigma = (S, F, R)$ une signature. On appelle *ensemble de variables* sur Σ un S-ensemble V tel que pour tout $s \in S$, $V_s \cap (S \cup F \cup R) = \emptyset$.

À partir de cet ensemble de symboles complémentaire à la signature, et des éléments de la signature, il est possible de construire inductivement les briques de base qui vont être utilisées pour exprimer les propriétés du type de données, appelées termes avec variables. Un terme avec variables est soit une variable, soit un terme construit sur une opération de la signature, c'est-à-dire une opération de la signature appliquée à des arguments correspondant, en type et en nombre, à l'arité de cette opération.

DÉFINITION II.6 — *Termes.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. Le S-ensemble des *termes avec variables*, noté $T_\Sigma(V)$, est l'ensemble défini de la façon suivante :

- pour tout $s \in S$, pour tout $x \in V_s$, $x \in T_\Sigma(V)_s$;
- pour tout $f : s_1 \times \ldots \times s_n \to s \in F$, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_s$.

EXEMPLE II.4 — *Monoïdes.*

Sur la signature des monoïdes qui a été donnée à l'exemple II.1, et sur l'ensemble de variables $V = V_{Élem} = \{x, y, z\}$, on peut par exemple construire les termes suivants :

$$e,\ e * x,\ (x * y) * z,\ e * e$$

◇

Les propriétés décrivant un type de données vont pouvoir être exprimées à l'aide des prédicats de la signature appliqués aux termes construits précédemment, des connecteurs booléens et des quantificateurs universel et existentiel. Ces propriétés sont appelées des formules.

DÉFINITION II.7 — *Formules.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. L'ensemble des *formules* sur Σ, noté $For(\Sigma)$, est l'ensemble défini de la façon suivante :

- pour tout $r : s_1 \times \ldots \times s_n \in R$, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, $r(t_1, \ldots, t_n) \in For(\Sigma)$;
- pour tout $\varphi \in For(\Sigma)$, $\neg\varphi \in For(\Sigma)$;
- pour tout $\varphi, \psi \in For(\Sigma)$, $\varphi \wedge \psi, \varphi \vee \psi, \varphi \Rightarrow \psi \in For(\Sigma)$;
- pour tout $x \in V$, pour tout $\varphi \in For(\Sigma)$, $\forall x\varphi, \exists x\varphi \in For(\Sigma)$.

EXEMPLE II.5 — *Monoïdes.*

Pour donner une spécification d'un monoïde, il nous faut exprimer les propriétés caractérisant la neutralité de l'élément distingué e et l'associativité de l'opération $*$. L'élément e est neutre dans le sens où la composition à gauche ou à droite d'un élément quelconque avec e n'a pas d'effet. On a donc les axiomes suivants :

Neutralité de e à gauche $\quad \forall x, e * x = x$

Neutralité de e à droite $\quad \forall x, x * e = x$

Associativité de $*$ $\quad \forall x, \forall y, \forall z, (x * y) * z = x * (y * z)$

On remarque que les axiomes sont ici de simples équations, quantifiées universellement. La structure de monoïde ne nécessite en fait pour être décrite que la logique équationnelle. En logique du premier ordre, il est nécessaire d'ajouter les axiomes suivants spécifiant que l'égalité est une relation de congruence.

$\forall x, x = x$

$\forall x, \forall y, x = y \Rightarrow y = x$

$\forall x, \forall y, \forall z, x = y \wedge y = z \Rightarrow x = z$

$\forall x, \forall y, \forall z, \forall t, x = z \wedge y = t \Rightarrow x * y = z * t$

◇

EXEMPLE II.6 — *Groupes.*

Un groupe est un monoïde dont tous les éléments sont inversibles. Si on utilise l'opération $\_^{-1}$ pour exprimer cette propriété, on ajoute aux axiomes de l'exemple II.5 les axiomes suivants, exprimant le fait que cette opération donne bien l'inverse de son argument. Les opérations étant définies par défaut de façon totale, tout élément possède une image par $\_^{-1}$, donc tout élément est inversible.

$\forall x, x * x^{-1} = e$

$\forall x, x^{-1} * x = e$

De la même manière que pour les monoïdes, la logique équationnelle suffirait à décrire la structure de groupe. Si on veut au contraire exploiter les mécanismes de la logique du premier ordre, on peut spécifier l'inverse comme une relation binaire symétrique sur les éléments et on obtient les axiomes suivants. Le premier axiome exprime le fait que tout élément possède un inverse selon cette relation, le deuxième donne la définition de l'inverse et le troisième précise que la relation est symétrique.

$\forall x, \exists y, inv(x, y)$

$\forall x, \forall y, inv(x, y) \Leftrightarrow x * y = e$

$\forall x, \forall y, inv(x, y) \Rightarrow inv(y, x)$

Il est également possible de spécifier cette propriété sans ajouter d'opération ni de relation à la signature du monoïde, en utilisant l'écriture mathématique habituelle.

$\forall x, \exists y, x * y = e \wedge y * x = e$ ◇

EXEMPLE II.7 — *Graphes.*

L'union de graphes est définie comme étant associative et commutative, et ayant l'ensemble vide pour élément neutre.

- $\forall G, \emptyset \cup G \equiv G$
- $\forall G, \forall G', G \cup G' \equiv G' \cup G$
- $\forall G, \forall G', \forall G'', G \cup (G' \cup G'') \equiv (G \cup G') \cup G''$

Aucun sommet n'appartient au graphe vide. Un sommet appartient à un graphe si on peut écrire celui-ci comme l'union du singleton contenant ce sommet et d'un autre graphe.

- $\forall x, \neg x \in \emptyset$
- $\forall x, \forall G, x \in G \Leftrightarrow \exists G', G \equiv \{x\} \cup G'$

Le graphe vide n'a aucune arête. On considère des graphes non orientés, la relation $arête$ est donc symétrique. Si une arête existe entre deux sommets dans un graphe, alors ces deux sommets appartiennent au graphe. Les arêtes sont préservées par l'union de graphes.

- $\forall x, \forall y, \neg\, arête_{\emptyset}(x, y)$
- $\forall x, \forall y, \forall G, arête_G(x, y) \Rightarrow arête_G(y, x)$
- $\forall x, \forall y, \forall G, arête_G(x, y) \Rightarrow x \in G \wedge y \in G$
- $\forall x, \forall y, \forall G, arête_G(x, y) \Rightarrow \forall G', arête_{G \cup G'}(x, y)$

Il n'y a aucun chemin dans le graphe vide. La relation de chemin est la fermeture symétrique et transitive de la relation d'arête. Si un chemin existe entre deux sommets, alors soit ces sommets sont reliés par une arête, soit il existe un sommet intermédiaire relié à chacun des deux sommets par un chemin.

- $\forall x, \forall y, \neg\, chemin_{\emptyset}(x, y)$
- $\forall x, \forall y, \forall G, arête_G(x, y) \Rightarrow chemin_G(x, y)$
- $\forall x, \forall y, \forall G, chemin_G(x, y) \Rightarrow chemin_G(y, x)$
- $\forall x, \forall y, \forall z, \forall G, chemin_G(x, y) \wedge chemin_G(y, z) \Rightarrow chemin_G(x, z)$
- $\forall x, \forall y, \forall G, chemin_G(x, y) \Rightarrow arête_G(x, y) \vee \exists z, chemin_G(x, z) \wedge chemin_G(z, y)$

Deux graphes sont égaux s'ils contiennent les même sommets et les mêmes arêtes. La relation d'égalité est une relation d'équivalence.

- $\forall G, \forall G', G \equiv G' \Leftrightarrow (\forall x, x \in G \Leftrightarrow x \in G') \wedge (\forall x, \forall y, arête_G(x, y) \Leftrightarrow arête_{G'}(x, y))$
- $\forall G, G \equiv G$
- $\forall G, \forall G', G \equiv G' \Rightarrow G' \equiv G$
- $\forall G, \forall G', \forall G'', G \equiv G' \wedge G' \equiv G'' \Rightarrow G \equiv G''$

On a de plus la propriété de compatibilité de l'égalité avec chaque opération et chaque prédicat. On en donne seulement deux exemples : l'opération d'union et le prédicat $complet$ spécifié ci-après.

- $\forall G, \forall G', \forall G'', G \equiv G' \Rightarrow G \cup G'' \equiv G' \cup G''$
- $\forall G, \forall G', G \equiv G' \Rightarrow (complet(G) \Leftrightarrow complet(G'))$

Un graphe est complet s'il existe une arête entre tout couple de sommets.

- $complet(G) \Leftrightarrow \forall x, \forall y, arête_G(x, y)$

Un graphe est connexe s'il existe un chemin entre tout couple de sommets.

- $connexe(G) \Leftrightarrow \forall x, \forall y, chemin_G(x, y)$

Un sommet est dit isolé dans un graphe s'il n'est relié à aucun autre sommet du graphe. Un graphe ne contient donc pas de sommet isolé si tout sommet est relié à un autre.

- $non\_isolé(G) \Leftrightarrow \forall x, \exists y, arête_G(x, y)$

Un graphe acyclique ne contient pas de chemin d'un sommet vers lui-même.

- $acyclique(G) \Leftrightarrow \forall x, \neg\, chemin_G(x, x)$

Un arbre est un graphe connexe acyclique.

- $arbre(G) \Leftrightarrow connexe(G \wedge acyclique(G)$

Un arbre couvrant un graphe G est un arbre possédant les mêmes sommets que G et un sous-ensemble de ses arêtes.

- $couvrant(A, G) \Leftrightarrow arbre(A) \wedge (\forall x, x \in A \Leftrightarrow x \in G) \wedge (\forall x, \forall y, ar\hat{e}te_A(x, y) \Rightarrow ar\hat{e}te_G(x, y))$

◇

Il nous faut définir maintenant définir le foncteur For et la façon dont il transporte les morphismes de signatures. Un morphisme de signatures $\eta : \Sigma \to \Sigma'$ est prolongé aux termes puis aux formules intuitivement en changeant tous les symboles de Σ apparaissant dans les termes et les formules par les symboles de Σ' correspondants.

DÉFINITION II.8 — *Prolongement des morphismes de signatures aux ensembles de formules.*

Soient Σ et Σ' deux signatures. Soient V un ensemble de variables sur Σ et V' un ensemble de variables sur Σ'. Soit $\eta : \Sigma \to \Sigma'$ un morphisme de signatures. On note $\eta_V : V \to V'$ l'application injective telle que pour tout $s \in S$, pour tout $x \in V_s$, $\eta_V(x) \in V'_{\eta_S(s)}$.

Le *prolongement de η aux termes avec variables*, noté $\overline{\eta} : T_\Sigma(V) \to T_{\Sigma'}(V)$, est défini pour tout terme t inductivement sur la forme de t de la façon suivante :

- si t est une variable $x \in V$, $\overline{\eta}(t) = \eta_V(x)$;
- si t est de la forme $f(t_1, \ldots, t_n)$, alors $\overline{\eta}(f(t_1, \ldots, t_n)) = \eta_F(f)(\overline{\eta}(t_1), \ldots, \overline{\eta}(t_n))$.

Le *prolongement de $\overline{\eta}$ aux formules*, noté $\overline{\eta}^\sharp : For(\Sigma) \to For(\Sigma')$, est défini pour toute formule φ inductivement sur la forme de φ de la façon suivante :

- si φ est de la forme $r(t_1, \ldots, t_n)$, alors $\overline{\eta}^\sharp(\varphi) = \eta_R(r)(\overline{\eta}(t_1), \ldots, \overline{\eta}(t_n))$;
- si φ est de la forme $\neg\psi$, alors $\overline{\eta}^\sharp(\varphi) = \neg\overline{\eta}^\sharp(\psi)$;
- si φ est de la forme $\psi \odot \chi$ avec $\odot \in \{\wedge, \vee, \Rightarrow\}$, alors $\overline{\eta}^\sharp(\varphi) = \overline{\eta}^\sharp(\psi) \odot \overline{\eta}^\sharp(\chi)$;
- si φ est de la forme $Qx\psi$ avec $Q \in \{\forall, \exists\}$, alors $\overline{\eta}^\sharp(\varphi) = Q\eta_V(x)\ \overline{\eta}^\sharp(\psi)$.

Par abus de notation, on notera également $\overline{\eta}$ le prolongement de $\overline{\eta}$ aux formules.

Le foncteur For associe donc à chaque signature son ensemble de formules et à chaque morphisme de signatures son prolongement aux formules.

DÉFINITION II.9 — *Foncteur For.*

On définit $For : Sig \to \mathcal{S}et$ comme étant le foncteur :

- qui associe à chaque signature $\Sigma \in |\, Sig\,|$ l'ensemble $For(\Sigma)$ des formules sur Σ ;
- qui associe à chaque morphisme de signatures $\eta : \Sigma \to \Sigma'$ le prolongement de η aux ensembles de formules $\overline{\eta} : For(\Sigma) \to For(\Sigma')$.

1.2 Sémantique

Modèles. Un type de données est un ensemble muni d'éléments distingués, de lois de composition internes et externes et de relations. On appelle l'interprétation des symboles d'une signature Σ un modèle de Σ ou

un Σ-modèle.

DÉFINITION II.10 — *Modèle d'une signature.*

Soit $\Sigma = (S, F, R)$ une signature. Un *modèle* $\mathcal{M}$ associé à Σ est un S-ensemble M muni pour chaque nom d'opération $f : s_1 \times \ldots \times s_n \to s \in F$ d'une application $f_{\mathcal{M}} : M_{s_1} \times \ldots \times M_{s_n} \to M_s$, et pour chaque nom de prédicat $r : s_1 \times \ldots \times s_n \in R$ d'une relation n-aire $r_{\mathcal{M}} \subseteq M_{s_1} \times \ldots \times M_{s_n}$.

EXEMPLE II.8 — *Groupes.*

Un modèle de la signature donnée à l'exemple II.2 est tout ensemble muni d'un élément distingué et de deux opérations internes unaire et binaire. Par exemple, l'ensemble $\mathbb{N}$ des entiers naturels où e est l'entier 0, l'opération $\_ * \_$ est interprétée par l'addition de deux entiers et l'opération $\_^{-1}$ est interprétée par la fonction qui retire 1 à tout entier non nul et à 0 associe 0 est un modèle de cette signature. Ce n'est bien sûr pas un groupe mais la signature seule ne donne aucune indication sur le comportement des opérations. Ce sont les axiomes qui permettent de distinguer entre les modèles de la signature ceux qui correspondent au comportement attendu.

Un autre exemple de modèle pour cette signature est l'ensemble des entiers relatifs $\mathbb{Z}$ où $e = 0$, l'opération $\_ * \_$ est interprétée par l'addition et $\_^{-1}$ est interprétée par la fonction qui à tout entier associe son opposé. Ce modèle étant un groupe, il sera un modèle de la spécification des groupes donnée à l'exemple II.6.

La distinction entre ces deux modèles sera faite grâce à la notion de satisfaction d'une formule par un modèle en logique du premier ordre définie aux pages suivantes.

◇

Pour définir la catégorie des modèles associée à une signature, il nous faut définir les relations entre les modèles d'une signature : les morphismes de modèles. Deux modèles d'une même signature sont deux ensembles munis d'interprétations différentes des symboles de cette signature. Un morphisme de modèles est donc un moyen de faire correspondre ces deux ensembles et ces deux interprétations des symboles de manière à préserver la satisfaction des formules.

DÉFINITION II.11 — *Morphisme de modèles.*

Soit $\Sigma = (S, F, R)$ une signature. Soient $\mathcal{M}, \mathcal{M}' \in Mod(\Sigma)$ deux modèles de Σ. Un *morphisme de modèles* $\mu : \mathcal{M} \to \mathcal{M}'$ est une famille d'applications $\mu = (\mu_s : M_s \to M'_s)_{s \in S}$ telle que pour toute opération $f : s_1 \times \ldots \times s_n \to s \in F$, pour tout $(a_1, \ldots, a_n) \in M_{s_1} \times \ldots \times M_{s_n}$,

$$\mu_s(f_{\mathcal{M}}(a_1, \ldots, a_n)) = f_{\mathcal{M}'}(\mu_{s_1}(a_1), \ldots, \mu_{s_n}(a_n))$$

et pour tout prédicat $r : s_1 \times \ldots \times s_n \in R$, pour tout $(a_1, \ldots, a_n) \in M_{s_1} \times \ldots \times M_{s_n}$,

$$(a_1, \ldots, a_n) \in r_{\mathcal{M}} \Leftrightarrow (\mu_{s_1}(a_1), \ldots, \mu_{s_n}(a_n)) \in r_{\mathcal{M}'}$$

DÉFINITION II.12 — *Catégorie des modèles associée à une signature.*

Soit $\Sigma \in |\, Sig \,|$. La *catégorie des modèles de* Σ, notée $Mod(\Sigma)$, est la catégorie dont les objets sont les modèles de Σ et dont les morphismes sont les morphismes de modèles.

On peut donc associer à chaque signature la catégorie des modèles de cette signature. Il nous reste à définir de quelle manière les morphismes de signatures sont transportés pour définir le foncteur Mod. Ce foncteur étant contravariant de $Sig \to \mathcal{C}at$, un morphisme de signatures $\eta : \Sigma \to \Sigma'$ est transporté en un foncteur de $Mod(\Sigma') \to Mod(\Sigma)$ qui permet de construire un modèle de Σ à partir d'un modèle de Σ' en « oubliant » les sorte, les opérations et les prédicats qui n'ont pas d'antécédent dans Σ par η.

DÉFINITION II.13 — *Foncteur d'oubli U_σ.*

Soient Σ et Σ' deux signatures. Soit $\eta : \Sigma \to \Sigma'$ un morphisme de signatures. On définit $U_\eta : Mod(\Sigma') \to Mod(\Sigma)$ comme étant le foncteur :

- qui associe à chaque modèle $\mathcal{M}' \in |\, Mod(\Sigma')|$ le modèle $U_\eta(\mathcal{M}')$ dont l'ensemble sous-jacent est M' et qui est muni pour chaque nom d'opération $f \in F$ de l'application $f_{\mathcal{M}'}$ et pour chaque nom de prédicat $r \in R$ de la relation $r_{\mathcal{M}'}$;
- qui associe à chaque morphisme de modèles $\mu' : \mathcal{M}'_1 \to \mathcal{M}'_2$ le morphisme $U_\eta(\mu') : U_\eta(\mathcal{M}'_1) \to U_\eta(\mathcal{M}'_2)$ défini pour tout $s \in S$ par $U_\eta(\mu')_s = \mu'_s$.

Le foncteur Mod associe donc à chaque signature sa catégorie de modèles et à chaque morphisme de signatures le foncteur d'oubli correspondant.

DÉFINITION II.14 — *Foncteur Mod.*

On définit $Mod : Sig \to \mathcal{C}at^{op}$ comme étant le foncteur :

- qui associe à chaque signature $\Sigma \in |\, Sig\,|$ la catégorie $Mod(\Sigma)$ des modèles de Σ ;
- qui associe à chaque morphisme de signatures $\eta : \Sigma \to \Sigma'$ le foncteur d'oubli $U_\eta : Mod(\Sigma') \to Mod(\Sigma)$.

Relation de satisfaction. Une fois donnée l'interprétation des symboles de base, il faut donner un sens à leur combinaison, c'est-à-dire aux termes d'abord et aux formules ensuite. Les termes étant construits inductivement à partir des symboles de la signature et de variables, interpréter un terme implique d'attribuer des valeurs aux variables, des valeurs prises dans l'ensemble sous-jacent au modèle de la signature. On a donc besoin de disposer d'une application donnant à chacune des variables une de ces valeurs, en préservant les sortes, c'est ce qu'on appelle une interprétation des variables. On peut ensuite prolonger cette interprétation des variables aux termes selon la définition inductive de ceux-ci.

DÉFINITION II.15 — *Interprétation des termes.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. Soit $\mathcal{M}$ un Σ-modèle. Une *interprétation des variables* est une application $\nu : V \to M$, telle que pour tout $s \in S$, pour tout $x \in V_s$, $\nu(x) \in M_s$. On prolonge toute interprétation des variables ν en une *interprétation des termes* $\nu^\sharp : T_\Sigma(V) \to M$ de la manière inductive suivante :

- si $x \in V$, alors $\nu^\sharp(x) = \nu(x)$;
- si $f : s_1 \times \ldots \times s_n \to s \in F$ et $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, alors $\nu^\sharp(f(t_1, \ldots, t_n)) = f_{\mathcal{M}}(\nu^\sharp(t_1), \ldots, \nu^\sharp(t_n))$.

Par abus de notation, on notera également ν le prolongement de l'interprétation des variables aux termes.

Si l'ensemble de variables considéré est vide, une interprétation des termes devient une interprétation des termes clos, c'est-à-dire des éléments de T_Σ. Or ces termes sont construits uniquement à partir des symboles de la signature donc, pour un modèle de la signature donné, l'interprétation des termes clos est unique. Lorsque toutes les valeurs du modèle peuvent être dénotées par un terme clos, on dit que ce modèle est finiment engendré.

DÉFINITION II.16 — *Modèle finiment engendré.*

Soit Σ une signature et V un ensemble de variables sur Σ. Soit $\mathcal{M}$ un Σ-modèle. On note $\_^{\mathcal{M}} : T_\Sigma \to M$ l'unique application qui donne son interprétation dans M à chaque terme clos. On dit que $\mathcal{M}$ est *finiment engendré* si et seulement si $\_^{\mathcal{M}}$ est surjective, c'est-à-dire si pour tout $v \in M$, il existe $t \in T_\Sigma$ tel que $t^{\mathcal{M}} = v$.

On note $Gen(\Sigma)$ la sous-catégorie pleine de $Mod(\Sigma)$ dont les objets sont les modèles de Σ finiment engendrés.

Maintenant qu'on a su donner aux éléments syntaxiques du langage un sens mathématique, on peut définir la vérité d'une formule pour une interprétation donnée de la syntaxe. Autrement dit, on est capable de dire si une propriété sous la forme d'une formule est vérifiée ou non par un modèle de la signature pour une certaine interprétation des variables.

DÉFINITION II.17 — *Satisfaction des formules.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. Soit $\mathcal{M}$ un Σ-modèle. Soit $\nu : V \to M$ une interprétation des variables. Soit $\varphi \in For(\Sigma)$. La *satisfaction de φ par le modèle $\mathcal{M}$ pour l'interprétation ν*, notée $\mathcal{M} \models_\nu \varphi$, est définie sur la structure de φ de la manière suivante :

- si φ est de la forme $r(t_1, \ldots, t_n)$, alors $\mathcal{M} \models_\nu \varphi$ ssi $(\nu(t_1), \ldots, \nu(t_n)) \in r_{\mathcal{M}}$;
- si φ est de la forme $\neg\psi$, alors $\mathcal{M} \models_\nu \varphi$ ssi [1] $\mathcal{M} \not\models_\nu \psi$;
- si φ est de la forme $\psi \wedge \chi$, alors $\mathcal{M} \models_\nu \varphi$ ssi $\mathcal{M} \models_\nu \psi$ et $\mathcal{M} \models_\nu \chi$;
- si φ est de la forme $\psi \vee \chi$, alors $\mathcal{M} \models_\nu \varphi$ ssi $\mathcal{M} \models_\nu \psi$ ou $\mathcal{M} \models_\nu \chi$;
- si φ est de la forme $\psi \Rightarrow \chi$, alors $\mathcal{M} \models_\nu \varphi$ ssi, si $\mathcal{M} \models_\nu \psi$, alors $\mathcal{M} \models_\nu \chi$.
- si φ est de la forme $\forall x\psi$, alors $\mathcal{M} \models_\nu \varphi$ ssi pour toute interprétation $\nu' : V \to M$ qui vérifie $\nu'(y) = \nu(y)$ pour tout $y \in V \setminus \{x\}$, $\mathcal{M} \models_{\nu'} \psi$;
- si φ est de la forme $\exists x\psi$, alors $\mathcal{M} \models_\nu \varphi$ ssi il existe une interprétation $\nu' : V \to M$ qui vérifie $\nu'(y) = \nu(y)$ pour tout $y \in V \setminus \{x\}$, telle que $\mathcal{M} \models_{\nu'} \psi$;

On dit que $\mathcal{M}$ *valide* φ, noté $\mathcal{M} \models \varphi$, si et seulement si $\mathcal{M} \models_\nu \varphi$ pour tout $\nu : V \to M$.

THÉORÈME II.1. (Institution de la logique du premier ordre [GB92]) *Le quadruplet* $(Sig, For, Mod, \models)$ *où :*

- Sig *est la catégorie des signatures ;*
- For *est le foncteur donné à la définition II.9 ;*
- Mod *est le foncteur donné à la définition II.14 ;*
- $\models = (\models_\Sigma)_{\Sigma \in |Sig|}$ *où* $\models_\Sigma$ *est la relation de satisfaction de la définition II.17,*

est une institution.

[1] $\mathcal{M} \not\models_\nu \varphi$ est une notation abrégée pour « il n'est pas vrai que $\mathcal{M} \models_\nu \varphi$ ».

1.3 Calcul

Parmi les différents systèmes de preuve qui ont été définis pour la logique du premier ordre, c'est le calcul des séquents que nous avons choisi de présenter ici, car c'est sur ce dernier que sera défini notre algorithme de sélection de tests (voir le chapitre VI). Ce calcul a été introduit par Gerhard Gentzen en 1934 dans le cadre des recherches qu'il menait alors sur la déduction logique et la consistance de l'arithmétique. C'est dans ce calcul qu'il a réussi à montrer le théorème d'élimination des coupures, dans lequel il énonce que toute tautologie de la logique du premier ordre peut être déduite sans l'utilisation de lemmes intermédiaires.

L'élément syntaxique de base de ce calcul est le séquent, qui est une représentation particulière d'une formule du premier ordre sous une forme symétrique.

DÉFINITION II.18 — *Séquent.*

Soit Σ une signature. Un *séquent* est un couple (Γ, Δ) où Γ et Δ sont des ensembles finis de formules sur Σ. Un séquent (Γ, Δ) sera noté $\Gamma \mid\sim \Delta$.

Par la suite, pour tout ensemble $\Gamma \subseteq For(\Sigma)$ et pour toute formule $\varphi \in For(\Sigma)$, on notera Γ, φ l'ensemble $\Gamma \cup \{\varphi\}$ au sein d'un séquent.

Sémantiquement, un séquent $\varphi_1, \ldots, \varphi_m \mid\sim \psi_1, \ldots, \psi_n$ représente la formule

$$\varphi_1 \wedge \ldots \wedge \varphi_m \Rightarrow \psi_1 \vee \ldots \vee \psi_n$$

Remarquons que les séquents considérés ici sont formés d'ensembles de formules et non pas de listes comme il est également possible de les définir. Cela permet de ne prendre en compte ni l'ordre des formules, ni la présence éventuelle d'occurrences multiples d'une formule.

Les règles d'inférence du calcul des séquents font appel à quelques notions qu'il est nécessaire de présenter ici. L'utilisation des quantificateurs en logique du premier ordre amène la notion de variables libres ou liées, selon qu'une variable est sous la portée d'un quantificateur ou non.

DÉFINITION II.19 — *Ensemble de variables d'une formule.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. Soit $t \in T_\Sigma(V)$ un terme sur Σ. L'*ensemble des variables de* t, noté $Var(t)$, est défini de la façon suivante :

- si t est une variable x, alors $Var(t) = \{x\}$;
- si t est de la forme $f(t_1, \ldots, t_n)$, alors $Var(t) = Var(t_1) \cup \ldots \cup Var(t_n)$.

Un terme t tel que $Var(t) = \emptyset$ est appelé *terme clos*. L'ensemble des termes clos sur Σ est noté T_Σ.

On étend la définition de cet ensemble aux formules de la façon suivante. Soit $\varphi \in For(\Sigma)$. L'*ensemble des variables de* φ, noté $Var(\varphi)$, est l'ensemble défini de la façon suivante :

- si φ est de la forme $r(t_1, \ldots, t_n)$, alors $Var(\varphi) = Var(t_1) \cup \ldots \cup Var(t_n)$;
- si φ est de la forme $\neg\psi$, alors $Var(\varphi) = Var(\psi)$;
- si φ est de la forme $\psi \wedge \chi$, $\psi \vee \chi$ ou $\psi \Rightarrow \chi$, alors $Var(\varphi) = Var(\psi) \cup Var(\chi)$;
- si φ est de la forme $\forall x\psi$ ou $\exists x\psi$, alors $Var(\varphi) = \{x\} \cup Var(\psi)$;

Une formule φ telle que $Var(\varphi) = \emptyset$ est appelée *formule sans variables*.

On remarque que, par définition, une formule sans variable n'est pas quantifiée.

DÉFINITION II.20 — *Variables libres et liées.*

Soit Σ une signature et V un ensemble de variables sur Σ. Soit $\varphi \in For(\Sigma)$. On définit[2] l'ensemble $FV(\varphi)$ des *variables libres* de φ et l'ensemble $BV(\varphi)$ de ses *variables liées* de la façon suivante :

- si φ est de la forme $r(t_1, \ldots, t_n)$, alors $FV(\varphi) = Var(\varphi)$ et $BV(\varphi) = \emptyset$;
- si φ est de la forme $\neg\psi$, alors $FV(\varphi) = FV(\psi)$ et $BV(\varphi) = BV(\psi)$;
- si φ est de la forme $\psi \wedge \chi$, $\psi \vee \chi$ ou $\psi \Rightarrow \chi$, alors $FV(\varphi) = FV(\psi) \cup FV(\chi)$ et $BV(\varphi) = BV(\psi) \cup BV(\chi)$;
- si φ est de la forme $\forall x\psi$ ou $\exists x\psi$, alors $FV(\varphi) = FV(\psi) \setminus \{x\}$ et $BV(\varphi) = BV(\psi) \cup \{x\}$.

Une formule φ telle que $FV(\varphi) = \emptyset$ est dite *close*.

Dans la formule $\exists y, x < y$ par exemple, la variable y est liée au quantificateur existentiel, tandis que la variable x est libre. Une variable libre dans une formule peut donc être remplacée par n'importe quelle autre variable qui n'est pas déjà liée dans la formule, ou plus généralement, par tout terme tel qu'aucune de ses variables ne tombe, par ce remplacement, sous la portée d'un quantificateur. Un tel terme est dit libre pour cette variable dans la formule considérée.

DÉFINITION II.21 — *Terme libre pour une variable.*

Soit Σ une signature et V un ensemble de variables sur Σ. Soit $\varphi \in For(\Sigma)$. Soit $x \in Var(\varphi)$ une variable de φ et $t \in T_\Sigma(V)$ un terme. On dit que t est *libre pour x dans φ* si :

- φ est de la forme $r(t_1, \ldots, t_n)$;
- φ est de la forme $\neg\psi$ et si t est libre pour x dans ψ ;
- φ est de la forme $\psi \wedge \chi$, $\psi \vee \chi$ ou $\psi \Rightarrow \chi$, et si t est libre pour x dans ψ et dans χ ;
- φ est de la forme $\forall y\psi$ ou $\exists y\psi$, si
  - x et y sont la même variable,
  - ou bien x et y sont distinctes et $y \notin Var(t)$,

  et si t est libre pour x dans ψ.

Reprenons l'exemple de la formule $\exists y, x < y$. Dans cette formule, tout terme est libre pour x, à condition que y n'apparaisse pas dans t. On peut alors remplacer x par la variable z pour construire la formule $\exists y, 2z < y$, ou bien même par le terme $z + 2x$ afin de construire la formule $\exists y, z + 2x < y$ par exemple, sans changer l'interprétation de la formule. Le remplacement de toutes les occurrences d'une variable libre dans une formule par un même terme est appelé une substitution et est défini de la manière suivante.

DÉFINITION II.22 — *Substitution.*

Soit Σ une signature et V un ensemble de variables sur Σ. Une *substitution des variables* est une famille d'applications indexée par S : $\sigma = (\sigma_s)_{s \in S}$ où pour tout $s \in S$, $\sigma_s : V_s \to T_\Sigma(V)_s$.

On prolonge de façon canonique toute substitution des variables en une *substitution des termes*. Si $\sigma : V \to T_\Sigma(V)$ est une substitution des variables, le prolongement de σ aux termes, noté $\sigma^\natural$, est l'application de $T_\Sigma(V) \to T_\Sigma(V)$ définie de la façon suivante :

[2] On note l'ensemble des variables libres d'une formule FV pour *free variables* et l'ensemble des variables liées BV pour *bound variables*.

- si $x \in V$ alors $\sigma^\sharp(x) = \sigma(x)$;
- si $f : s_1 \times \ldots \times s_n \to s \in F$ et $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$ alors $\sigma^\sharp(f(t_1, \ldots, t_n)) = f(\sigma^\sharp(t_1), \ldots, \sigma^\sharp(t_n))$.

Par abus de notation, on notera également σ le prolongement de la substitution σ aux termes.

On prolonge toute substitution des termes en une *substitution des formules*. Si $\sigma : T_\Sigma(V) \to T_\Sigma(V)$ est une substitution, le prolongement de σ aux formules, noté $\sigma^\sharp$, est l'application de $For(\Sigma) \to For(\Sigma)$ définie pour toute formule φ inductivement de la façon suivante :

- si φ est de la forme $r(t_1, \ldots, t_n)$, alors $\sigma^\sharp(\varphi) = r(\sigma(t_1), \ldots, \sigma(t_n))$;
- si φ est de la forme $\neg\psi$, alors $\sigma^\sharp(\varphi) = \neg\sigma^\sharp(\psi)$;
- si φ est de la forme $\psi \odot \chi$ pour $\odot \in \{\wedge, \vee, \Rightarrow\}$, alors $\sigma^\sharp(\varphi) = \sigma^\sharp(\psi) \odot \sigma^\sharp(\chi)$;
- si φ est de la forme $Qx\psi$ pour $Q \in \{\forall, \exists\}$, alors $\sigma^\sharp(\varphi) = Qy\ \sigma^\sharp \circ \tau(\psi)$ où τ est la substitution qui à x associe y et qui laisse les autres variables inchangées, et $y \notin \{FV(t) \mid \exists z \in V, t = \sigma(z)\} \cup FV(\psi) \setminus \{x\}$.

Par abus de notation, on notera également σ le prolongement de la substitution σ aux formules.

Dans la suite de cette section, on notera $\varphi[t/x]$ la formule φ à laquelle aura été appliquée la substitution σ qui à x associe t et qui laisse les autres variables inchangées. La substitution d'une variable par un terme n'a donc de sens que si ce terme est libre pour cette variable dans la formule considérée.

Les règles d'inférence du calcul des séquents données à la figure II.1 sont de plusieurs sortes. La règle la plus simple consiste en l'introduction d'une tautologie. Un séquent $\Gamma \mathrel{|\!\sim} \Delta$ s'interprétant comme dit précédemment, une tautologie est un séquent dans lequel la même formule apparaît à gauche et à droite du signe $\mathrel{|\!\sim}$, la formule $\varphi \Rightarrow \varphi$ étant toujours trivialement vérifiée. Cette règle n'ayant pas de prémisse, elle est appelée *axiome*. Les règles d'inférence proprement dites caractérisent les propriétés des opérateurs booléens (négation, conjonction, disjonction, implication) et des quantifications universelle et existentielle. Les règles associées aux connecteurs booléens et aux quantificateurs sont appelées introduction lorsqu'elles sont utilisées de haut en bas, et élimination lorsqu'elles sont utilisées de bas en haut. La règle de coupure Cut est une généralisation de la règle de modus ponens, bien connue en logique propositionnelle. Cette règle traduit la partie de la table de vérité de l'implication logique qui stipule que la conclusion d'une implication est vraie si la prémisse l'est :

$$\frac{A \qquad A \Rightarrow B}{B}\,\mathsf{MP}$$

Si on considère dans la règle de coupure Γ, Γ' et Δ comme étant des séquents vides et Δ' contenant une unique formule ψ, on retrouve

$$\frac{\mathrel{|\!\sim} \varphi \qquad \varphi \mathrel{|\!\sim} \psi}{\mathrel{|\!\sim} \psi}\,\mathsf{Cut}$$

Les règles associées aux quantificateurs sont soumises à des restrictions : dans les règles $\forall$-droit et $\exists$-gauche, la variable x ne doit être libre dans aucune des formules de Γ et de Δ.

Ces règles d'inférence ont une lecture relativement intuitive. Considérons par exemple la règle $\neg$-droit. Elle énonce que si Γ et φ suffisent à prouver Δ, alors Γ seul peut prouver Δ ou bien φ doit être faux, c'est-à-dire $\neg\varphi$ est vrai.

$$\frac{}{\Gamma, \varphi \mathrel{|\!\sim} \Delta, \varphi}\,\textsf{Taut}$$

$$\frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi}{\Gamma, \neg\varphi \mathrel{|\!\sim} \Delta}\,\neg\text{-gauche} \qquad \frac{\Gamma, \varphi \mathrel{|\!\sim} \Delta}{\Gamma \mathrel{|\!\sim} \Delta, \neg\varphi}\,\neg\text{-droit}$$

$$\frac{\Gamma, \varphi, \psi \mathrel{|\!\sim} \Delta}{\Gamma, \varphi \wedge \psi \mathrel{|\!\sim} \Delta}\,\wedge\text{-gauche} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi \quad \Gamma \mathrel{|\!\sim} \Delta, \psi}{\Gamma \mathrel{|\!\sim} \Delta, \varphi \wedge \psi}\,\wedge\text{-droit}$$

$$\frac{\Gamma, \varphi \mathrel{|\!\sim} \Delta \quad \Gamma, \psi \mathrel{|\!\sim} \Delta}{\Gamma, \varphi \vee \psi \mathrel{|\!\sim} \Delta}\,\vee\text{-gauche} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi, \psi}{\Gamma \mathrel{|\!\sim} \Delta, \varphi \vee \psi}\,\vee\text{-droit}$$

$$\frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi \quad \Gamma, \psi \mathrel{|\!\sim} \Delta}{\Gamma, \varphi \Rightarrow \psi \mathrel{|\!\sim} \Delta}\,\Rightarrow\text{-gauche} \qquad \frac{\Gamma, \varphi \mathrel{|\!\sim} \Delta, \psi}{\Gamma \mathrel{|\!\sim} \Delta, \varphi \Rightarrow \psi}\,\Rightarrow\text{-droit}$$

$$\frac{\Gamma, \varphi[t/x] \mathrel{|\!\sim} \Delta}{\Gamma, \forall x\varphi \mathrel{|\!\sim} \Delta}\,\forall\text{-gauche} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi}{\Gamma \mathrel{|\!\sim} \Delta, \forall x\varphi}\,\forall\text{-droit}^{a}$$

$$\frac{\Gamma, \varphi \mathrel{|\!\sim} \Delta}{\Gamma, \exists x\varphi \mathrel{|\!\sim} \Delta}\,\exists\text{-gauche}^{a} \qquad \frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi[t/x]}{\Gamma \mathrel{|\!\sim} \Delta, \exists x\varphi}\,\exists\text{-droit}$$

$$\frac{\Gamma \mathrel{|\!\sim} \Delta, \varphi \qquad \Gamma', \varphi \mathrel{|\!\sim} \Delta'}{\Gamma, \Gamma' \mathrel{|\!\sim} \Delta, \Delta'}\,\textsf{Cut}$$

[a] $\forall \psi \in \Gamma \cup \Delta, x \notin FV(\psi)$

FIG. II.1 – *Calcul des séquents.*

Afin de donner l'intuition des règles associées aux quantificateurs, considérons la règle $\forall$-droit. Il n'est bien sûr pas possible de conclure que $\forall x\varphi$ du seul fait que φ est vraie pour une certaine instance de x. Cependant, si cette variable x n'est libre dans aucune des formules de Γ et de Δ, c'est-à-dire qu'elle peut être modifiée sans influencer les autres formules, alors elle est en fait quantifiée universellement de façon implicite, φ est donc vraie pour toute instance de x, d'où $\forall x\varphi$.

EXEMPLE II.9 — *Arbre de preuve.*

On donne ici l'exemple de la dérivation du séquent $\mathrel{|\!\sim} (\forall x f(x) \Rightarrow g(x)) \Rightarrow (\forall x f(x) \Rightarrow \exists x g(x))$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{g(t), f(s) \mathrel{|\!\sim} g(t)}\,\textsf{Taut}}{g(t), f(s) \mathrel{|\!\sim} \exists x g(x)}\,\exists\text{-droit}}{g(t), \forall x f(x) \mathrel{|\!\sim} \exists x g(x)}\,\forall\text{-gauche} \qquad \dfrac{\dfrac{\dfrac{}{f(t) \mathrel{|\!\sim} g(s), f(t)}\,\textsf{Taut}}{f(t) \mathrel{|\!\sim} \exists x g(x), f(t)}\,\exists\text{-droit}}{\forall x f(x) \mathrel{|\!\sim} \exists x g(x), f(t)}\,\forall\text{-gauche}}{f(t) \Rightarrow g(t), \forall x f(x) \mathrel{|\!\sim} \exists x g(x)}\,\Rightarrow\text{-gauche}}{f(t) \Rightarrow g(t) \mathrel{|\!\sim} \forall x f(x) \Rightarrow \exists x g(x)}\,\Rightarrow\text{-droit}}{\forall x f(x) \Rightarrow g(x) \mathrel{|\!\sim} \forall x f(x) \Rightarrow \exists x g(x)}\,\forall\text{-gauche}}{\mathrel{|\!\sim} (\forall x f(x) \Rightarrow g(x)) \Rightarrow (\forall x f(x) \Rightarrow \exists x g(x))}\,\Rightarrow\text{-droit}$$

◇

Au lieu de voir les règles d'inférence de ce système comme une description des dérivations autorisées dans la logique du premier ordre, il est également possible de les considérer comme des instructions pour la construction d'une preuve d'une formule donnée. Il faut pour cela lire les règles de bas en haut. Par exemple, la règle $\wedge$-droit indique que pour prouver $\varphi \wedge \psi$, il suffit de prouver φ et de prouver ψ, dans le même contexte Γ et Δ. Ainsi, d'une preuve de φ et d'une preuve de ψ, il est possible de construire une preuve de $\varphi \wedge \psi$.

Dans la recherche de certaines preuves, la plupart des règles indiquent plus ou moins directement la manière de mener cette recherche. Le cas de règle de coupure est en cela différent. Cette règle énonce que si une formule φ peut être la conclusion d'une formule d'une part, et si elle peut servir de prémisse dans une autre formule d'autre part, alors cette formule peut être coupée, et les dérivations respectives s'unissent. Dans la construction d'une preuve de bas en haut, le problème est de trouver cette formule φ, puisqu'elle n'apparaît pas dans la formule qu'on cherche à prouver. C'est le problème auquel s'intéresse le théorème d'élimination des coupures.

DÉFINITION II.23 — *Système formel associé au calcul des séquents.*

Soient $\Sigma = (S, F, R)$ une signature et V un ensemble de variables sur Σ. Le *système formel associé au calcul des séquents* est le triplet $C_\Sigma = (A_\Sigma, F_\Sigma, R_\Sigma)$ où :

- $A_\Sigma = F \cup R \cup V \cup \{\neg, \wedge, \vee, \Rightarrow, \forall, \exists, \ \sim, (,), ','\}$;
- $F_\Sigma = \{\Gamma \sim \Delta \mid \Gamma, \Delta \subseteq For(\Sigma)\}$;
- R_Σ est l'ensemble des règles d'inférence du calcul des séquents.

DÉFINITION II.24 — *Système d'inférence de la logique du premier ordre.* Le *système d'inférence associé à la logique du premier ordre* est le triplet $(Sig, For, \vdash)$ où Sig est la catégorie des signatures, For est le foncteur donné à la définition II.9 et $\vdash = (\vdash_\Sigma)_{\Sigma \in |Sig|}$ où $\vdash_\Sigma = \vdash_{C_\Sigma}$.

Nous avons vu au chapitre I que tout système d'inférence défini par un système formel est réflexif, transitif et monotone. Il est facile de prouver que système d'inférence associé à la logique du premier ordre possède la propriété de $\vdash$-translation. Enfin, il est connu que le calcul des séquents pour la logique du premier ordre est correct (il est aussi complet) [Bar77]. On a donc le théorème suivant.

THÉORÈME II.2. [Mes89] *Le quintuplet* $(Sig, For, Mod, \models, \vdash)$ *est une logique générale.*

2 Logique équationnelle

La logique équationnelle est une restriction de la logique du premier ordre au seul prédicat d'égalité. Ainsi, pour cette logique, une signature est composée uniquement d'un ensemble de sortes et un ensemble de noms de fonctions, le prédicat d'égalité, noté $=$, devenant un symbole fixe de la logique. Ce prédicat est unique, quelle que soit la sorte des termes sur lesquels il s'applique. Une équation est alors tout élément de $T_\Sigma(V)_s \times T_\Sigma(V)_s$. Pour tous $t, t' \in T_\Sigma(V)_s$, on note $t = t'$ l'équation (t, t'). Les formules de la logique équationnelle peuvent être réduites aux équations, on parle alors de logique équationnelle élémentaire. Sinon, les formules sont construites inductivement à partir des équations, des connecteurs booléens usuels et des quantificateurs universel et existentiel.

Une signature équationnelle est interprétée mathématiquement par une algèbre (universelle), les modèles du premier ordre étant des extensions des algèbres aux prédicats.

DÉFINITION II.25 — *Algèbre.*

Soit $\Sigma = (S, F)$ une signature. Une *algèbre* $\mathcal{A}$ sur Σ est un S-ensemble A muni pour chaque nom d'opération $f : s_1 \times \ldots \times s_n \to s \in F$ d'une application $f_{\mathcal{A}} : A_{s_1} \times \ldots \times A_{s_n} \to A_s$.

On note $Alg(\Sigma)$ la catégorie dont les objets sont les algèbres sur Σ.

Les termes et les formules sont interprétés exactement comme dans le cadre du premier ordre, une équation $t = t'$ étant validée par une algèbre pour une interprétation des variables ν donnée si et seulement si les valeurs $\nu(t)$ et $\nu(t')$ sont égales dans l'algèbre considérée.

Afin de prendre en compte le raisonnement concis et efficace associé à l'égalité connu sous le nom de remplacement d'égal par égal, le calcul des séquents est étendu au raisonnement équationnel. Les règles suivantes, ajoutées aux règles précédemment données, spécialisent le calcul des séquents au prédicat d'égalité [Gal86].

$$\frac{\Gamma, t = t \mid\sim \Delta}{\Gamma \mid\sim \Delta}\,\mathsf{Réf} \qquad \frac{\Gamma, (t = t' \Rightarrow t' = t) \mid\sim \Delta}{\Gamma \mid\sim \Delta}\,\mathsf{Sym}$$

$$\frac{\Gamma, (t = t' \wedge t' = t'' \Rightarrow t = t'') \mid\sim \Delta}{\Gamma \mid\sim \Delta}\,\mathsf{Trans}$$

$$\frac{\Gamma, (t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)) \mid\sim \Delta}{\Gamma \mid\sim \Delta}\,\mathsf{Cong}$$

3 Logique conditionnelle positive

3.1 Définition

Une des restrictions de la logique équationnelle a été particulièrement étudiée pour ses propriétés algébriques, telles que l'initialité, et de décidabilité. La restriction porte sur la forme des formules considérées, appelées formules conditionnelles positives. Ces formules permettent d'exprimer le fait qu'une propriété, sous la forme d'une équation, est conséquence d'un ensemble de conditions, exprimé sous la forme d'une conjonction d'équations. Plus formellement, une formule conditionnelle positive est une formule de la forme :

$$t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t'$$

où $n \geq 0$, l'ensemble de conditions pouvant être vide. Les simples équations sont donc également des formules conditionnelles positives. Ces formules suffisent à spécifier un très grand nombre des structures de données habituelles.

Cette restriction sur la forme des axiomes est liée à la volonté d'utiliser des techniques de preuve pour la sélection d'un jeu de tests. En effet, l'utilisation de telles formules permet de définir un calcul concis et pratique, dont les règles d'inférence sont fondées sur chacune des propriétés de l'égalité et des connecteurs logiques : outre la règle d'introduction d'un axiome de la spécification, ce calcul exprime les propriétés de l'égalité en tant que congruence ainsi que les propriétés logiques de la conjonction et de l'implication et de la quantification universelle (ici implicite).

$$\frac{\varphi \in Ax}{(\Sigma, Ax) \vdash \varphi}\ \mathsf{Ax} \qquad \frac{}{Sp \vdash t = t}\ \mathsf{Réf} \qquad \frac{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t = t'}{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t' = t}\ \mathsf{Sym}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t = t' \qquad Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t' = t''}{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t = t''}\ \mathsf{Trans}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t_1 = t'_1 \quad \ldots \quad Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow t_n = t'_n}{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)}\ \mathsf{Cong}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow \alpha}{Sp \vdash \bigwedge_{1\leq i\leq m} \sigma(\alpha_i) \Rightarrow \sigma(\alpha)}\ \mathsf{Subs} \qquad \frac{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow \alpha}{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \wedge \beta \Rightarrow \alpha}\ \mathsf{Mono}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \wedge \alpha \Rightarrow \beta \qquad Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow \alpha}{Sp \vdash \bigwedge_{1\leq i\leq m} \alpha_i \Rightarrow \beta}\ \mathsf{MP}$$

FIG. II.2 – *Calcul pour des spécifications conditionnelles positives.*

Les règles Réf, Sym, Trans et Cong caractérisent respectivement la réflexivité, la symétrie, la transitivité de l'égalité, ainsi que sa compatibilité avec les opérations de la signature, traduisant ainsi le fait que le prédicat d'égalité est une relation de congruence. La règle de substitution Subs définit le comportement du quantificateur universel traité implicitement dans les formules. Les variables jouant le rôle de valeurs génériques, une formule est valide si elle est valide pour toutes les valeurs possibles que peuvent prendre les variables. Elle est donc toujours valide après l'application d'une substitution changeant une variable par un terme avec variables. La règle Mono de monotonie traduit le fait que l'implication considérée est l'implication logique, dont la valeur de vérité est vraie en particulier lorsque celle de l'antécédent est fausse. La règle de modus ponens MP traduit le reste de la table de vérité de l'implication, c'est-à-dire que la conclusion est valide si les prémisses le sont.

On peut trouver dans plusieurs références les preuves de correction et de complétude de ce calcul [BN98].

3.2 Congruence et initialité

Nous présentons ici quelques résultats fondamentaux de la logique équationnelle en général et de la logique conditionnelle positive en particulier. Ces résultats interviendront dans les preuves d'exhaustivité du jeu de tests formé des conséquences sémantiques observables d'une spécification donnée présentées dans la

première partie de cette thèse.

Lorsque, pour chacune des sortes d'une signature donnée, l'ensemble des termes n'est pas vide, il est possible de construire une algèbre particulière appelée algèbre des termes.

DÉFINITION II.26 — *Algèbre des termes.*

Soit $\Sigma = (S, F)$ une signature telle que pour tout $s \in S$, $T_{\Sigma_s} \neq \emptyset$. L'*algèbre des termes* $\mathcal{T}_\Sigma$ est l'ensemble T_Σ muni pour chaque opération $f : s_1 \times \ldots \times s_n \to s \in F$ de l'application $f_{\mathcal{T}_\Sigma} : T_{\Sigma_{s_1}} \times \ldots \times T_{\Sigma_{s_n}} \to T_{\Sigma_s}$ qui à tout n-uplet $(t_1, \ldots, t_n)$ associe le terme $f(t_1, \ldots, t_n)$.

L'algèbre des termes est donc l'algèbre dont l'ensemble sous-jacent est celui des termes clos. Dans cette algèbre, deux termes syntaxiquement différents sont deux éléments différents. De plus, tout élément de l'algèbre étant dénoté par un terme clos par définition, cette algèbre est finiment engendrée. La propriété la plus importante de cette algèbre est la suivante :

THÉORÈME II.3. *Soit $\Sigma = (S, F)$ une signature telle que pour tout $s \in S$, $T_{\Sigma_s} \neq \emptyset$. L'algèbre des termes $\mathcal{T}_\Sigma$ est initiale dans $Alg(\Sigma)$.*

On rappelle qu'un modèle initial est un objet initial dans la catégorie des modèles (voir l'annexe A). Pour une signature donnée, l'algèbre initiale possède exactement les propriétés qui correspondent à la compréhension intuitive de l'algèbre visée par la description donnée par cette signature. Les autres modèles de cette signature, bien qu'ils interprètent les mêmes opérations, peuvent avoir d'autres caractéristiques qui ne sont pas attendues

On rappelle que pour un ensemble X donné, une relation d'équivalence sur X est une relation binaire $\equiv \subseteq X \times X$ qui est réflexive, symétrique et transitive. La relation $\equiv$ partitionne l'ensemble X en un ensemble disjoint de classes d'équivalence $\lfloor x \rceil = \{y \in X \mid y \equiv x\}$. Le quotient de X par la relation $\equiv$ donne l'ensemble de ces classes : $X/_\equiv = \{\lfloor x \rceil \mid x \in X\}$. La notion de congruence étend la notion d'équivalence aux algèbres.

DÉFINITION II.27 — *Congruence.*

Soient $\Sigma = (S, F)$ une signature et $\mathcal{A} \in Alg(\Sigma)$ une algèbre sur Σ. Soit $\equiv$ une relation d'équivalence sur A, c'est-à-dire une famille indexée par S de relations d'équivalence sur chaque A_s. Alors $\equiv$ est une *congruence* sur $\mathcal{A}$ si et seulement si pour tout $f : s_1 \times \ldots \times s_n \to s \in F$ et pour tous $(a_1, \ldots, a_n), (b_1, \ldots, b_n) \in A_{s_1} \times \ldots \times A_{s_n}$,

$$\forall i, 1 \leq i \leq n, a_i \equiv b_i \Rightarrow f_{\mathcal{A}}(a_1, \ldots, a_n) \equiv f_{\mathcal{A}}(b_1, \ldots, b_n)$$

Il est alors possible de munir un ensemble quotienté par une relation d'équivalence d'une structure d'algèbre, comme l'énonce la proposition ci-dessous.

PROPOSITION II.1. *Soient $\Sigma = (S, F)$ une signature et $\mathcal{A} \in Alg(\Sigma)$ une algèbre sur Σ. Soit $\equiv$ une congruence sur $\mathcal{A}$. L'ensemble $A_\equiv$ est canoniquement muni d'une structure d'algèbre sur Σ de la façon suivante :*

– *$A_\equiv = (A/_\equiv)_s$ où pour tout $s \in S$, $(A/_\equiv)_s$ est l'ensemble $A_s/_{\equiv_s}$;*

– *pour tout* $f : s_1 \times \ldots \times s_n \to s \in F$, *l'interprétation de* f *est l'application* $f_{\mathcal{A}/\equiv} : (A/\equiv)_{s_1} \times \ldots \times (A/\equiv)_{s_n} \to (A/\equiv)_s$ *définie pour tout* n*-uplet* $(\lfloor a_1 \rceil, \ldots, \lfloor a_n \rceil)$ *par* $f_{\mathcal{A}/\equiv}(\lfloor a_1 \rceil, \ldots, \lfloor a_n \rceil) = \lfloor f_{\mathcal{A}}(a_1, \ldots, a_n) \rceil$.

Cette algèbre est appelée algèbre quotient.

Les résultats suivants sont au centre de la théorie des spécifications algébriques. Ils permettent en effet de construire une algèbre initiale pour une spécification donnée, comme quotient de l'algèbre des termes par une congruence induite par les axiomes.

THÉORÈME II.4. *Soient* $\Sigma = (S, F)$ *une signature et* $\mathcal{A} \in Alg(\Sigma)$ *une algèbre sur* Σ. *Soit* $\mathcal{R}$ *une relation binaire sur* A, *c'est-à-dire une famille indexée par* S *de relations binaire sur chaque* A_s. *Il existe une plus petite congruence sur* $\mathcal{A}$ *contenant* $\mathcal{R}$, *appelée* congruence engendrée par $\mathcal{R}$ sur $\mathcal{A}$ *et notée* $\equiv_{\mathcal{R}}$.

Notation. Si $Sp = (\Sigma, Ax)$ est une spécification et $\mathcal{A} \in Alg(\Sigma)$ une algèbre sur Σ, on note $\mathcal{R}_{Ax}$ la relation binaire sur A définie pour tout $(a, b) \in A \times A$ par $a\mathcal{R}_{Ax}b$ si et seulement s'il existe un axiome de Ax $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t'$ et une substitution $\sigma : V \to A$ telle que $\sigma(t_i) = \sigma(t'_i)$ pour tout i, $1 \leq i \leq n, \sigma(t) = a$ et $\sigma(t') = b$.

La relation $\mathcal{R}_{Ax}$ est l'ensemble des égalités qui manquent à $\mathcal{A}$ pour qu'elle valide les axiomes de Ax. En fermant cette relation par réflexivité, symétrie, transitivité et compatibilité avec les opérations de Σ, on obtient une congruence par laquelle il est possible de quotienter l'algèbre des termes. On obtient alors une algèbre dont l'ensemble sous-jacent est l'ensemble des termes clos quotienté par l'égalité entre termes induite des axiomes.

THÉORÈME II.5. *Soit* $Sp = (\Sigma, Ax)$ *une spécification. L'algèbre quotient de* $\mathcal{T}_\Sigma$ *par la congruence engendrée par* $\mathcal{R}_{Ax}$ *est initiale dans* $Alg(Sp)$.

Chapitre III

Logique modale du premier ordre et coalgèbres

Les méthodes algébriques standard décrites au chapitre précédent ont montré leur efficacité à capturer les différents aspects essentiels des structures de données utilisées en informatique. Cependant, il s'avère difficile de décrire de façon algébrique les structures intrinsèquement dynamiques qu'on rencontre naturellement en informatique. Les approches formelles des systèmes dynamiques font généralement usage d'automates ou de systèmes de transitions.

Il existe une multitude de formalismes fondés sur la représentation graphique de la spécification d'un système tels que les systèmes de transitions. L'idée a émergé dans les années 80 qu'une abstraction de ces formalismes pouvait être fournie par des structures mathématiques duales des algèbres appelées coalgèbres. Les coalgèbres se sont alors montré adaptées à la description de la dynamique des systèmes tels que les automates déterministes [AM80, MA86]. Plus tard, dans les années 90, elles ont été utilisées en informatique pour formaliser les notions de classe et d'objet en programmation orientée-objet [Rei95, Jac95b]. Comme pour les algèbres, les coalgèbres étaient, dans ces travaux, spécifiées en logique équationnelle. Les connexions entre les spécifications équationnelles et les coalgèbres ont alors été étudiées, par Ulrich Hensel et Horst Reichel [HR94] et Bart Jacobs [Jac95a]. Dans les années 2000, plusieurs travaux proposèrent une nouvelle approche pour la spécification de structures coalgébriques, à savoir l'utilisation de la logique modale [Rößl01, Kur01, Mos99]. En effet, les coalgèbres sont des généralisations des systèmes de transitions et la logique modale est toujours naturellement utilisée pour décrire les propriétés de tels systèmes. De plus, les coalgèbres décrivent de manière générale des systèmes « à états ». La logique modale permet de spécifier le comportement des opérations qui permettent de faire évoluer l'état du système sans faire référence explicitement à cet état, seul le comportement observable du système est décrit.

La logique modale que nous présentons dans ce chapitre est une logique plus générale que celles précédemment citées car elle permet également de manipuler des données. C'est la logique sous-jacente au langage de spécification CoCasl [MSRR06], défini comme une extension coalgébrique du langage de spécification algébrique Casl et qui permet de combiner la spécification de structures de données algébriques et de processus de nature coalgébrique.

1 Coalgèbres

Nous présentons tout d'abord les coalgèbres en tant que représentations abstraites des systèmes dynamiques, ainsi que les notions et résultats de la théorie des coalgèbres dont nous aurons besoin dans la suite de cette thèse. Pour une introduction pédagogique aux coalgèbres et à la coinduction comme notions duales des algèbres et de l'induction, on pourra consulter l'article de Bart Jacobs et Jan Rutten [JR97]. Pour une présentation complète de la théorie des coalgèbres, on se référera aux articles de référence de Jan Rutten [Rut00] et de Peter Gumm [Gum99] ou au livre en préparation de Bart Jacobs [Jac]. On trouvera dans le chapitre du « Handbook of Modal Logic » écrit par Yde Venema une description des liens entre coalgèbres et logique modale [Ven06].

1.1 Coalgèbres et foncteurs polynomiaux

Une coalgèbre peut être vue comme une abstraction des systèmes à base d'états, consistant en un ensemble E muni d'une certaine opération de transition α, représentée formellement comme une fonction de E dans un ensemble $\mathcal{F}E$. $\mathcal{F}$ désigne un foncteur de la catégorie $\mathcal{S}et$ des ensembles dans elle-même qui représente la signature de la coalgèbre [1]. La fonction de transition munit l'ensemble E d'une certaine structure. Au contraire des opérations algébriques qui permettent de construire des objets complexes à partir des objets de base donnés par la signature, les opérations coalgébriques peuvent être vues comme un moyen de déconstruire ou d'observer les états de E. Plus formellement, on a la définition suivante d'une coalgèbre pour un foncteur donné.

DÉFINITION III.1 — *Coalgèbre.*

Soit $\mathcal{F} : \mathcal{S}et \to \mathcal{S}et$ un foncteur, appelé *foncteur de signature*. Une *coalgèbre du foncteur* $\mathcal{F}$, ou $\mathcal{F}$-coalgèbre, est un couple (E, α) où :

- E est un ensemble dont les éléments sont appelés *états* ;
- $\alpha : E \to \mathcal{F}E$ est une fonction appelée *fonction de transition*.

EXEMPLE III.1 — *Ensemble C-coloré.*

Un des exemples les plus simples que l'on puisse considérer est celui d'un système constitué uniquement d'un ensemble d'états E et dont chaque état est muni d'une couleur $c \in C$, où C est un ensemble quelconque. Quel que soit son état initial, ce système est uniquement capable de donner la couleur de cet état, sans le modifier, puis s'arrête. Ce système peut alors être représenté par la coalgèbre $(E, \alpha : E \to C)$ du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = C$, où E est l'ensemble des états et où α donne à chaque état la couleur qui lui est associée. ◇

EXEMPLE III.2 — *Presse-bouton.*

On considère maintenant une machine en boîte noire possédant un bouton et une lumière. Lorsque le bouton est pressé, une certaine action interne s'effectue, qui fait évoluer l'état interne du système. La lumière s'allume lorsqu'il n'est plus possible d'actionner la machine, c'est-à-dire lorsqu'elle a atteint un état bloquant,

[1] De manière générale, il est possible de définir une coalgèbre sur n'importe quel endofoncteur, c'est-à-dire n'importe quel foncteur d'une catégorie dans elle-même. Nous nous restreignons ici à la catégorie $\mathcal{S}et$ des ensembles, celle-ci étant suffisante pour notre propos.

dit aussi terminal. De l'extérieur, il n'est pas possible de connaître l'état interne de la machine, mais seulement d'observer son comportement au travers du bouton et de la lumière. On est seulement capable d'observer si la lumière est allumée ou non, c'est la seule information dont on dispose sur l'état courant du système. Il est possible de faire évoluer l'état de la machine, et ce jusqu'à temps que la lumière s'allume, si elle s'allume (la machine peut fonctionner éternellement et la lumière ne jamais s'allumer).

Pour représenter cette machine, on considère la coalgèbre $(E, \mathit{bouton} : E \to 1 + E)$, où E est un ensemble représentant l'ensemble de ses états et $1 = \{*\}$ est l'ensemble singleton, l'étoile $*$ dénotant un élément quelconque n'appartenant pas à E. En un état e donné du système, appliquer la fonction bouton peut produire deux sorties : soit un état de E si $\mathit{bouton}(e) \in E$, soit $\mathit{bouton}(e) = *$ signifiant que la lumière s'allume et que la machine s'arrête. Dans le premier cas, l'état de la machine a évolué vers un autre état, dans lequel le bouton peut de nouveau être pressé.

◇

EXEMPLE III.3 — *Presse-bouton 2.*

On considère maintenant une machine en boîte noire qui possède deux boutons : val et suiv. Appuyer sur le premier bouton permet d'afficher une valeur en fonction de l'état de la machine, une couleur d'un ensemble C par exemple, sans affecter cet état, tandis que le deuxième bouton permet de faire évoluer le système vers un autre état. Les opérations val et suiv sont donc définies par

$$\mathit{val} : E \to C \qquad \mathit{suiv} : E \to E$$

On peut écrire ces deux opérations en une seule

$$\langle \mathit{val}, \mathit{suiv} \rangle : E \to C \times E$$

Cette machine peut donc être représentée par la coalgèbre $(E, \langle \mathit{val}, \mathit{suiv} \rangle : E \to C \times E)$ du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = C \times X$.

◇

EXEMPLE III.4 — *Suivant.*

On combine les deux machines précédentes pour obtenir une machine qui possède un bouton bouton et une lumière. Appuyer sur le bouton renvoie une valeur $c \in C$ et fait passer la machine dans un état suivant, ou bien allume la lumière si la machine ne peut plus évoluer. On obtient alors un système correspondant à la coalgèbre $(E, \mathit{bouton} : E \to 1 + C \times E)$ du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = 1 + C \times X$.

◇

EXEMPLE III.5 — *Buffer à une place.*

On considère ici un buffer à une place muni de deux opérations crire et lire. La première permet de stocker un élément d'un ensemble A dans le buffer, tandis que la deuxième permet de lire la valeur stockée s'il y en a une et renvoie un message d'erreur si le buffer est vide. Ces opérations sont définies de la manière suivante où l'ensemble singleton contient le message d'erreur :

$$\mathit{crire} : E \times A \to E \qquad \mathit{lire} : E \to 1 + A \times E$$

qu'on peut écrire sous la forme

$$\langle \mathit{crire}, \mathit{lire} \rangle : E \to E^A \times (1 + A \times E)$$

Une coalgèbre représentant ce système est donc une coalgèbre du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = X^A \times (1 + A \times X)$. ◇

EXEMPLE III.6 — *Automates - LTS*.

Prenons maintenant quelques exemples tirés de la théorie des automates.

Automate non-déterministe. Dans un automate non-déterministe, plusieurs transitions sont possibles à partir d'un état donné. La fonction de transition d'un tel système est donc une fonction qui à chaque état associe un ensemble d'états. Une coalgèbre représentant ce système est alors construite sur le foncteur $\mathcal{P}$ qui à un ensemble associe l'ensemble des parties de cet ensemble :

$$(Q, \alpha : Q \to \mathcal{P}(Q))$$

Automate déterministe reconnaissant un langage. Un automate reconnaissant un langage est un automate construit sur un alphabet A et qui possède un état initial et des états acceptants. Un mot est reconnu par l'automate s'il est possible de trouver, depuis l'état initial, un chemin étiqueté par chacune des lettres du mot dans l'ordre qui mène à un état acceptant. Le langage reconnu par l'automate est l'ensemble des mots reconnus par cet automate. Un tel automate est représenté par un quadruplet $(Q, A, \delta : Q \times A \to Q, F)$ où Q est l'ensemble des états, A est l'alphabet, δ est la fonction de transition prenant un état et donnant l'état suivant selon la lettre lue, et $F \subseteq Q$ est l'ensemble des états acceptants. Il est possible d'écrire δ sous la forme $\delta : Q \to Q^A$. De plus, on peut dénoter les états acceptants par une fonction booléenne $acc : Q \to \{0, 1\}$ telle que $acc(q) = 1$ si et seulement si $q \in F$. On peut alors représenter un tel automate par une coalgèbre du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = X^A \times \{0, 1\}$:

$$(Q, \langle \delta, acc \rangle : Q \to Q^A \times \{0, 1\})$$

Machine de Moore. Une machine de Moore est un automate déterministe à entrées et sorties dont les entrées permettent de faire évoluer l'état et dont les sorties dépendent uniquement de l'état courant. Une machine de Moore est généralement décrite par un quintuplet $(Q, I, O, \gamma : Q \times O, \delta : Q \times I \to Q)$ où Q est l'ensemble des états, I et O sont respectivement les ensembles des entrées et des sorties, γ est la fonction de sortie qui à chaque état associe une sortie, et δ est la fonction de transition qui à tout état associe l'état suivant selon la valeur d'entrée. Si $\delta(q, i) = q'$, on note généralement $q \xrightarrow{i} q'$. On remarque que la fonction δ peut également s'écrire $\delta : Q \to Q^I$, on peut alors écrire les fonctions γ et δ en une seule : $\langle \gamma, \delta \rangle : Q \to O \times Q^I$. Un tel automate correspond alors à une coalgèbre du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = O \times X^I$:

$$(Q, \langle \gamma, \delta \rangle : Q \to O \times Q^I)$$

Système de transitions étiquetées (LTS). Un système de transitions étiquetées est défini par un ensemble d'états Q et une relation δ de transition entre ces états, indexée par un ensemble A de valeurs appelées étiquettes : $\delta \subseteq Q \times A \times Q$. Si $(q, a, q') \in \delta$, on note habituellement $q \xrightarrow{a} q'$. Remarquons qu'une relation $R \subseteq E \times F$ peut être vue comme une fonction de E dans l'ensemble $\mathcal{P}(F)$ des parties de F. On peut alors

écrire la relation δ comme une fonction $\delta : Q \to \mathcal{P}(A \times Q)$. Un LTS est donc une coalgèbre du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = \mathcal{P}(A \times X)$:

$$(Q, \delta : Q \to \mathcal{P}(A \times Q))$$

De tels systèmes de transitions sont bien entendu non-déterministes. Cependant, on a souvent besoin de considérer que ce non-déterminisme est borné, c'est-à-dire que d'un état donné, seul un nombre fini de transitions est possible. Ces systèmes sont dits à branchement fini et peuvent être modélisés à l'aide du foncteur $\mathcal{P}_f$ de l'ensemble des parties finies :

$$(Q, \delta : Q \to \mathcal{P}_f(A \times Q))$$

Une autre classe de systèmes sont les coalgèbres de la forme

$$(Q, \delta : Q \to \mathcal{P}_f(Q)^A)$$

qui sont dites à image finie : pour tout état $q \in Q$ et tout $a \in A$, le nombre d'états accessibles à partir de q par une transition étiquetée par a, $\{q' \mid q \xrightarrow{a} q'\}$, est fini. ◇

EXEMPLE III.7 — *Frames et modèles de Kripke.*

Les modèles de la logique modale propositionnelle sont construits sur des structures appelées frames. Une frame est un couple (W, R) où W est un ensemble non vide d'états (également appelés mondes possibles) et R est une relation binaire sur W appelée relation d'accessibilité. Cette relation pouvant être vue comme une fonction $R : W \to \mathcal{P}(W)$, une frame est en fait une coalgèbre $(W, R : W \to \mathcal{P}(W))$ du foncteur $\mathcal{P}$ de l'ensemble des parties. On remarque qu'un automate non-étiqueté non-déterministe et une frame sont des coalgèbres du même foncteur, une frame pouvant être également vue comme un graphe orienté, donc un automate.

Un modèle de la logique modale propositionnelle est alors une frame à laquelle est ajoutée une valuation des variables propositionnelles. Un modèle est donc un triplet (W, R, v) avec $v : \mathsf{Prop} \to \mathcal{P}(W)$ où Prop est l'ensemble des variables propositionnelles. La fonction v associe à chaque variable propositionnelle l'ensemble des états dans lesquelles elle est validée. On peut également voir cette fonction de valuation comme une fonction qui à chaque état, associe l'ensemble $v^{-1}(w) = \{p \in \mathsf{Prop} \mid w \in v(p)\}$ des variables propositionnelles qui y sont valides. Un modèle est alors une coalgèbre du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = \mathcal{P}(\mathsf{Prop}) \times \mathcal{P}(X)$. ◇

Les foncteurs mentionnés dans les exemples précédents, excepté le foncteur $\mathcal{P}$ de l'ensemble des parties, sont ce qu'on appelle des foncteurs polynomiaux de Kripke, comme par exemple $\mathcal{F}X = A + (B \times X)^C$. Ces foncteurs sont construits inductivement à partir de foncteurs simples de base, en utilisant le produit, la somme, l'exposant et l'ensemble des parties pour construire de nouveaux foncteurs. Cette famille de foncteurs peut paraître restrictive, elle suffit cependant à décrire un grand nombre de cas concrets et a de plus l'avantage de posséder d'intéressantes propriétés, comme on le verra par la suite.

DÉFINITION III.2 — *Foncteurs polynomiaux (de Kripke).*

L'ensemble des *foncteurs polynomiaux* est défini inductivement de la manière suivante [2] :

[2] Voir la section 4 de l'annexe A pour la définition des différents foncteurs.

1. Le foncteur identité $\mathcal{I}d$ est un foncteur polynomial ;
2. Pour tout ensemble C, le foncteur constant $C : \mathcal{S}et \to \mathcal{S}et$ est un foncteur polynomial. Ce foncteur associe à tout ensemble X l'ensemble C et à toute fonction $f : X \to Y$ la fonction identité $\mathrm{Id}_C : C \to C$;
3. Si $\mathcal{T}$ et $\mathcal{S}$ sont deux foncteurs polynomiaux, alors le foncteur produit $\mathcal{T} \times \mathcal{S}$ est aussi polynomial. Ce foncteur associe à tout ensemble X l'ensemble $\mathcal{T}(X) \times \mathcal{S}(X)$ et à toute fonction $f : X \to Y$ la fonction $\mathcal{T}(f) \times \mathcal{S}(f)$ définie par (A.1) ;
4. Si $\mathcal{T}$ et $\mathcal{S}$ sont deux foncteurs polynomiaux, alors le foncteur coproduit $\mathcal{T} + \mathcal{S}$ est aussi polynomial. Ce foncteur associe à tout ensemble X l'ensemble $\mathcal{T}(X) + \mathcal{S}(X)$ et à toute fonction $f : X \to Y$ la fonction $\mathcal{T}(f) + \mathcal{S}(f)$ définie par (A.2) ;
5. Pour tout ensemble E, si $\mathcal{T}$ est un foncteur polynomial, alors l'exposant constant $\mathcal{T}^E$ est également polynomial. Ce foncteur associe à tout ensemble X l'ensemble $\mathcal{T}(X)^E$ et à toute fonction $f : X \to Y$ la fonction $\mathcal{T}(f)^{\mathrm{Id}_E}$ définie pour toute fonction $h : \mathcal{T}(X)^E$ par $\mathcal{T}(f)^{\mathrm{Id}_E}(h) = \mathcal{T}(f) \circ h$, voir (A.3).

L'ensemble des *foncteurs polynomiaux de Kripke* est le sur-ensemble des foncteurs polynomiaux défini par les 5 conditions précédentes auxquelles sont ajoutées les deux règles suivantes :

6. Si $\mathcal{T}$ est un foncteur polynomial de Kripke, alors le foncteur de l'ensemble des parties $\mathcal{P}(\mathcal{T})$ est aussi polynomial de Kripke. Ce foncteur associe à tout ensemble X l'ensemble $\mathcal{P}(\mathcal{T}(X))$ et à toute fonction $f : X \to Y$ la fonction $\mathcal{P}(\mathcal{T}(f)) : \mathcal{P}(\mathcal{T}(X)) \to \mathcal{P}(\mathcal{T}(Y))$ définie par (A.4) ;
7. Si $\mathcal{T}$ est un foncteur polynomial de Kripke, alors le foncteur $\mathcal{T}^*$ est aussi polynomial de Kripke. Ce foncteur associe à tout ensemble X l'ensemble X^* et à toute fonction $f : X \to Y$ la fonction $\mathcal{T}(f)^* : \mathcal{T}(X)^* \to \mathcal{T}(Y)^*$ définie par (A.5).

On appelle *foncteur polynomial de Kripke fini* un foncteur polynomial de Kripke dans lequel le foncteur $\mathcal{P}$ est remplacé par le foncteur $\mathcal{P}_f$ de l'ensemble des parties finies.

1.2 Morphismes et bisimulation

Un moyen effectif de comparer des coalgèbres peut se faire au travers des morphismes de coalgèbres.

DÉFINITION III.3 — *Morphisme de coalgèbres.*

Soient un foncteur $\mathcal{F} : \mathcal{S}et \to \mathcal{S}et$ et deux $\mathcal{F}$-coalgèbres (E, α) et (E', α'). Un *morphisme de coalgèbres* est une fonction $f : E \to E'$ telle que $\alpha' \circ f = \mathcal{F}(f) \circ \alpha$, c'est-à-dire telle que le diagramme suivant commute :

$$\begin{array}{ccc} E & \xrightarrow{\ f\ } & E' \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha'} \\ \mathcal{F}(E) & \xrightarrow[\ \mathcal{F}(f)\ ]{} & \mathcal{F}(E') \end{array}$$

Intuitivement, les morphismes de coalgèbres préservent les structures de transition. La fonction identité d'une $\mathcal{F}$-coalgèbre est un morphisme de coalgèbres et la composition de deux morphismes est un morphisme,

donc la collection des $\mathcal{F}$-coalgèbres munie des morphismes de $\mathcal{F}$-coalgèbres est une catégorie, notée $\mathsf{Coalg}(\mathcal{F})$.

EXEMPLE III.8 — *Systèmes de transitions étiquetés.*

On donne l'intuition du transport d'une coalgèbre au travers d'un morphisme de coalgèbres sur les systèmes de transitions étiquetés. On a vu précédemment (exemple III.6) qu'un système de transitions étiquetés est une coalgèbre du foncteur $\mathcal{F}X = \mathcal{P}(A \times X)$ où A est un ensemble d'étiquettes. On considère deux systèmes de transitions $(S, A, \longrightarrow_S)$ et $(T, A, \longrightarrow_T)$ sur le même ensemble d'étiquettes A, c'est-à-dire deux coalgèbres (S, α_S) et (T, α_T) du foncteur $\mathcal{F}$. Par définition, un morphisme de coalgèbres $f : (S, \alpha_S) \to (T, \alpha_T)$ est une fonction $f : S \to T$ telle que $\mathcal{F}(f) \circ \alpha_S = \alpha_T \circ f$ où la fonction $\mathcal{F}(f) : \mathcal{P}(A \times S) \to \mathcal{P}(A \times T)$ est définie pour tout $V \subseteq A \times S$ par :

$$\mathcal{F}(f)(V) = \mathcal{P}(A \times f)(V) = \{(a, f(s)) \in A \times T \mid (a, s) \in V\}$$

L'égalité $\mathcal{F}(f) \circ \alpha_S = \alpha_T \circ f$ est en fait équivalente aux deux conditions suivantes :

1. Pour toute transition $s \xrightarrow{a}_S s'$, on a la transition $f(s) \xrightarrow{a}_T f(s')$.
2. Pour toute transition $f(s) \xrightarrow{a}_T t$, il existe $s' \in S$ tel que $s \xrightarrow{a}_S s'$ et $f(s') = t$.

Un morphisme de coalgèbres est donc une fonction préservant les transitions. ◇

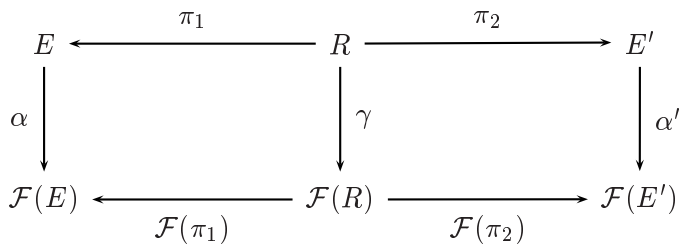
On a le résultat fondamental suivant sur la décomposition d'un morphisme de coalgèbres en un épimorphisme suivi d'un monomorphisme appelée factorisation.

THÉORÈME III.1. [Gum99] *Soient $\mathcal{F}$ un foncteur, (E, α) et (E', α') deux $\mathcal{F}$-coalgèbres. Tout morphisme de coalgèbre $f : E \to E'$ admet une unique factorisation $f = i \circ q$ où i est un monomorphisme et q est un épimorphisme.*

Les coalgèbres généralisant les systèmes de transitions, la notion de bisimulation entre systèmes trouve sa généralisation dans ce cadre. Une bisimulation entre deux coalgèbres est intuitivement une structure de transition préservant la relation entre les ensembles d'états.

DÉFINITION III.4 — *Bisimulation.*

Soient $\mathcal{F}$ un foncteur, (E, α) et (E', α') deux $\mathcal{F}$-coalgèbres. Une *bisimulation* entre E et E' est une relation $R \subseteq E \times E'$ pour laquelle il existe une structure de $\mathcal{F}$-coalgèbre $\gamma : R \to \mathcal{F}(R)$ telle que les deux fonctions de projection $\pi_1 : R \to E$ et $\pi_2 : R \to E'$ sont des morphismes de coalgèbres, c'est-à-dire tels que le diagramme suivant commute :



Deux états e et e' sont dits *bisimilaires* s'il existe une bisimulation R telle que $(e, e') \in R$.

EXEMPLE III.9 — *Systèmes de transitions étiquetés.*

On peut rapprocher cette définition de la définition habituelle de la bisimulation entre deux systèmes de transitions étiquetés, issue de la théorie de la concurrence. On considère deux systèmes de transitions étiquetés, c'est-à-dire deux coalgèbres (S, α_S) et (T, α_T) sur le foncteur $\mathcal{F}X = \mathcal{P}(A \times X)$ où A est un ensemble d'étiquettes. Une bisimulation entre S et T, au sens usuel, est une relation $R \subseteq S \times T$ satisfaisant, pour tout couple d'états $(s, t) \in R$, les deux conditions suivantes :

1. Pour tout $s' \in S$ tel qu'il existe une transition $s \xrightarrow{a}_S s'$, il existe $t' \in T$ tel que $t \xrightarrow{a}_T t'$ et $(s', t') \in R$.
2. Pour tout $t' \in T$ tel qu'il existe une transition $t \xrightarrow{a}_T t'$, il existe $s' \in S$ tel que $s \xrightarrow{a}_S s'$ et $(s', t') \in R$.

On considère R une bisimulation entre S et T munie de la fonction de transition $\alpha_R : R \to \mathcal{F}R$. La structure α_R de R induit une relation $\longrightarrow_R : R \times A \times R$. Soit $(s, t) \in R$ et $s \xrightarrow{a}_S s'$. Comme $s = \pi_1(s, t)$, on a $\pi_1(s, t) \xrightarrow{a}_S s'$. Puisque π_1 est un morphisme de coalgèbres, il existe une transition dans R, correspondant au « transport » de la transition $\pi_1(s, t) \xrightarrow{a}_S s'$ de S par π_1, de la forme $(s, t) \xrightarrow{a}_R (s'', t')$ tel que $\pi_1(s'', t') = s'$. D'où $(s', t') \in R$. Comme π_2 est un morphisme de coalgèbres, $\pi_2(s, t) \xrightarrow{a}_T \pi_2(s', t')$, c'est-à-dire $t \xrightarrow{a}_T t'$. La relation R vérifie donc la condition 1. La condition 2 se prouve de la même manière. Réciproquement, on suppose que R satisfait les conditions 1 et 2. On définit alors $\alpha_R : R \to \mathcal{P}(A \times R)$ pour tout $(s, t) \in R$ par $\alpha_R(s, t) = \{(a, (s', t')) \in A \times R \mid s \xrightarrow{a}_S s' \text{ et } t \xrightarrow{a}_T t'\}$. Des conditions 1 et 2 on déduit que les projections sont des morphismes de coalgèbres de (R, α_R) dans (S, α_S) et (T, α_T). $\diamond$

Notation. Pour une fonction $f : A \to B$, on note $G(f)$ son graphe, c'est-à-dire l'ensemble $\{(x, f(x)) \mid x \in A\}$.

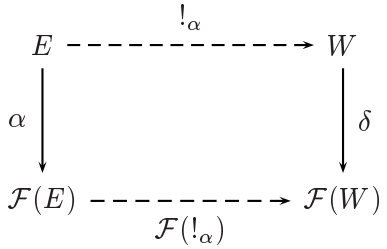
On a la relation fondamentale suivante entre les morphismes de coalgèbres et la notion de bisimulation.

THÉORÈME III.2. [Rut00] *Soient $\mathcal{F}$ un foncteur, (E, α) et (E', α') deux $\mathcal{F}$-coalgèbres. Une fonction $f : E \to E'$ est un morphisme de coalgèbre si et seulement si son graphe $G(f)$ est une bisimulation entre E et E'.*

1.3 Coalgèbre terminale

DÉFINITION III.5 — *Coalgèbre terminale.*

Une *coalgèbre terminale* (W, δ) d'un foncteur $\mathcal{F}$ est une coalgèbre telle que pour toute $\mathcal{F}$-coalgèbre (E, α), il existe un unique morphisme de coalgèbres $!_\alpha : (E, \alpha) \to (W, \delta)$:



Une coalgèbre terminale, si elle existe, est unique à isomorphisme près. Une coalgèbre terminale peut être vue comme une représentation minimale d'un système, c'est-à-dire qui contient de la façon la plus simple tous les comportements de ce système. Une fois construite une coalgèbre terminale pour un foncteur donné, on peut associer à tout état d'une coalgèbre de ce foncteur son comportement, c'est-à-dire un état de la coalgèbre terminale qui lui soit bisimilaire. C'est ce que dit le théorème suivant :

THÉORÈME III.3. [Gum99] *Soit $\mathcal{F}$ un foncteur. Si $\mathcal{F}$ admet une coalgèbre terminale (W, δ), alors pour toute $\mathcal{F}$-coalgèbre (E, α), pour tout état $e \in E$, il existe un unique état $w = !_\alpha(e) \in W$ tel que e et w sont bisimilaires.*

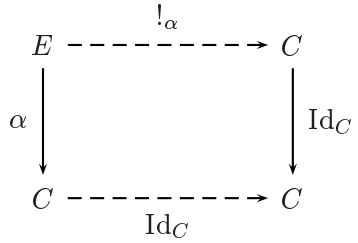
On a de plus le résultat suivant dû à Joachim Lambek (voir [SP82]), concernant la structure d'une coalgèbre terminale lorsqu'elle existe.

THÉORÈME III.4. [SP82] *Soit $\mathcal{F}$ un foncteur. Si $\mathcal{F}$ admet une coalgèbre terminale (W, δ), alors $\delta : W \to \mathcal{F}W$ est un isomorphisme.*

On donne maintenant quelques exemples de coalgèbres terminales pour les systèmes qui ont été décrits à la sous-section précédente.

EXEMPLE III.10 — *Ensemble C-coloré.*

Dans l'exemple III.1, le système décrit consiste à donner une couleur pour chaque état. Le comportement observable d'un état est alors réduit à la couleur qui lui est associée. La coalgèbre terminale pour le foncteur $\mathcal{F}$ défini par $\mathcal{F}X = C$ est donc construite sur l'ensemble C lui-même, muni de la fonction identité : $(C, \mathrm{Id}_C : C \to C)$. L'unique morphisme de coalgèbres entre une coalgèbre quelconque (E, α) de ce foncteur et la coalgèbre terminale est alors la fonction α elle-même, puisque ce diagramme doit commuter :



◇

EXEMPLE III.11 — *Presse-bouton.*

On considère la machine munie d'un bouton et d'une lumière décrite à l'exemple III.2. Appuyer sur le bouton permet de faire évoluer la machine dans un autre état, ou bien l'arrête. La seule observation disponible étant liée à la lumière, tout ce qu'un utilisateur de cette machine peut faire est appuyer sur le bouton et regarder si la lumière s'allume. Si elle ne s'allume pas, il peut réitérer l'opération, et noter combien de fois il a fallu appuyer sur le bouton pour que la lumière s'allume. Il peut ainsi obtenir un entier entre 0 et l'infini : 0 si la lumière est déjà allumée, un entier $n \in \mathbb{N}$ s'il parvient à allumer la lumière après n pressions, ou l'infini si la lumière ne s'allume jamais.

Le comportement observable de cette machine est alors un élément de $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$, décrivant le nombre de fois que le bouton a dû être pressé pour que la lumière s'allume. La coalgèbre terminale de ce

système est donc $(\overline{\mathbb{N}}, pred : \overline{\mathbb{N}} \to 1 + \overline{\mathbb{N}})$, où $pred$ est la fonction de transition suivante :

$$pred(n) = \begin{cases} \kappa_1(*) & \text{si } n = 0 \\ \kappa_2(n-1) & \text{si } 0 < n < \infty \\ \kappa_2(\infty) & \text{si } n = \infty \end{cases}$$

où κ_1 et κ_2 sont les coprojections associées au coproduit $1 + \overline{\mathbb{N}}$. ◇

EXEMPLE III.12 — *Presse-bouton 2.*

Considérons de nouveau la machine à deux boutons, val et $suiv$. Avec ces opérations, deux actions sont possibles étant donné un état $e \in E$: on peut produire un élément de C, $val(e)$, ou bien produire un état suivant de E, $suiv(e)$. Il est possible de répéter ces deux opérations et ainsi de former un autre élément de C, $val(suiv(e))$. En continuant de la sorte, il est possible de produire, pour chaque état $e \in E$ une suite infinie d'éléments de C $(c_0, c_1, c_2, \ldots) \in C^{\mathbb{N}}$, avec $c_i = val(suiv^n(e))$ pour tout $i \in \mathbb{N}$. Cette suite d'éléments issue de e est ce qu'il est possible d'observer à propos de e. La coalgèbre terminale de ce système est donc $(C^{\mathbb{N}}, \langle tête, queue \rangle : C^{\mathbb{N}} \to C \times C^{\mathbb{N}})$, où $tête$ et $queue$ sont les fonctions suivantes :

$$tête((c_0, c_1, c_2, \ldots)) = c_0 \qquad queue((c_0, c_1, c_2, \ldots)) = (c_1, c_2, c_3, \ldots)$$

qu'on peut également exprimer de la façon suivante, où γ est une fonction de $\mathbb{N}$ dans C :

$$tête(\gamma) = \gamma(0) \qquad queue(\gamma) = \gamma' \text{ t.q. } \forall n \in \mathbb{N}, \gamma'(n) = \gamma(n+1)$$

Pour tout coalgèbre $(E, \langle val, suiv \rangle : E \to C \times E)$ du foncteur $\mathcal{F}X = C \times X$, il existe donc un unique morphisme de coalgèbres $f : E \to C^{\mathbb{N}}$. Ce morphisme est celui qui associe à tout état de E son comportement observable, c'est-à-dire la suite infinie d'éléments de C donnés par les observations $val(suiv^n(e))$ pour tout n. Le morphisme f est donc défini pour tout $e \in E$, pour tout $n \in \mathbb{N}$, par :

$$f(e)(n) = val(suiv^n(e))$$

Il est facile de vérifier que $tête \circ f = val$ et $queue \circ f = f \circ suiv$, f est donc bien un morphisme de coalgèbres.

$$\begin{array}{ccc} E & \xrightarrow{\;f\;} & C^{\mathbb{N}} \\ {\scriptstyle \langle val, suiv \rangle}\downarrow & & \downarrow{\scriptstyle \langle tête, queue \rangle} \\ C \times E & \xrightarrow[\mathrm{Id}_A \times f]{} & C \times C^{\mathbb{N}} \end{array}$$

◇

EXEMPLE III.13 — *Suivant.*

Reprenons l'exemple de la machine munie d'un bouton et d'une lumière, et dont le bouton permet à la fois d'afficher une valeur dépendant de l'état et de faire évoluer l'état interne de la machine. Pour un état $e \in E$ de la machine, deux observations sont possibles : soit $bouton(e) = *$, ce qui signifie que la machine est arrêtée, qu'elle a atteint un état dans lequel il est impossible de continuer ; soit $bouton(e) = (c, e')$, donnant ainsi

un nouvel état à partir duquel il est possible d'appuyer de nouveau sur le bouton. En repétant l'opération, on obtient, pour tout état $e \in E$, une suite finie $(c_1, c_2, \ldots, c_n) \in C^*$ ou une suite infinie $(c_1, c_2, \ldots) \in C^{\mathbb{N}}$. Les observations associées aux états de E sont alors des éléments de l'ensemble $C^\infty = C^* + C^{\mathbb{N}}$ des listes finies et infinies d'éléments de C. La coalgèbre terminale est donc $(C^\infty, sp : C^\infty \to 1 + C \times C^\infty)$ où sp (pour « suivant possible ») est la fonction :

$$sp(e) = \begin{cases} \kappa_1(*) & \text{si } e = () \\ \kappa_2((a, e')) & \text{si } e = a \cdot e \end{cases}$$

où $()$ est la liste vide et $\_ \cdot \_$ est l'ajout d'un élément en tête de liste, κ_1 et κ_2 étant les coprojections associées au coproduit $1 + C \times C^\infty$. ◇

EXEMPLE III.14 — *Automate déterministe reconnaissant un langage.*

Un automate déterministe reconnaissant un langage est une coalgèbre $(Q, \langle \delta, acc \rangle : Q \to Q^A \times \{0,1\})$ du foncteur $\mathcal{F}X = X^A \times \{0,1\}$, comme on l'a vu à l'exemple III.6. Par définition de ces automates, à un état donné, fixé comme étant initial, il est possible d'associer un langage, la langage reconnu à partir de cet état. Le comportement observable d'un état est donc le langage qu'il permet de reconnaître. Le support de la coalgèbre terminale est donc l'ensemble des langages qu'il est possible de reconnaître à l'aide de cet automate, c'est-à-dire $\mathcal{P}(A^*)$, un langage reconnu par l'automate étant un ensemble de mots sur A. Cette coalgèbre est munie de la fonction de transition $\langle \lambda_1, \lambda_2 \rangle : \mathcal{P}(A^*) \to \mathcal{P}(A^*)^A \times \{0,1\}$ définie pour tout langage L par :

$$\lambda_1(L)(a) = \{w \in A^* \mid a.w \in L\}$$
$$\lambda_2(L) = 1 \Leftrightarrow \epsilon \in L$$

où ϵ dénote la séquence vide et $\_.\_$ l'ajout d'un élément en tête de séquence. ◇

EXEMPLE III.15 — *Machine de Moore.*

Comme il a été vu à l'exemple III.6, une machine de Moore est une coalgèbre $(Q, \langle \gamma, \delta \rangle : Q \to O \times Q^I)$ du foncteur $\mathcal{F}X = O \times X^I$. Les opérations possibles en un état $q \in Q$ sont les suivantes : on peut observer la sortie associée à l'état courant $\gamma(q) = o$, ou, pour une entrée $i \in I$, on peut faire évoluer la machine dans un nouvel état $\delta(q, i) = q'$. À partir d'un nouvel état, on peut réitérer, et ainsi obtenir une suite de sorties $(o_0, o_1, \ldots)$, où chacune des sorties est associée à une suite d'entrées : $\gamma(q) = o_0$ correspond à une suite d'entrées vide, $\gamma(\delta(q, i_1)) = o_1$ correspond à la suite d'entrée i_1, $\gamma(\delta(q, i_2)) = o_2$ correspond à la suite d'entrée i_2, $\gamma(\delta(\delta(q, i_1), i_3)) = o_3$ correspond à la suite d'entrées $i_1 i_3$, *etc.* Le comportement observable d'un état est donc une fonction de I^* dans O qui à chaque suite d'entrées finies $(i_1, i_2, \ldots, i_n)$ associe la sortie $\gamma(\delta(\delta(\ldots \delta(\delta(q, i_1), i_2), \ldots, i_n)))$.

La coalgèbre terminale du foncteur $\mathcal{F}$ est alors $(O^{I^*}, \Pi : O^{I^*} \to O \times (O^{I^*})^I)$ où Π est la fonction définie pour tout $\phi : I^* \to O$ par :

$$\Pi(\phi) = \langle \phi(\epsilon), \psi \rangle$$

où ψ est définie pour tout $i \in I$, pour tout $v \in I^*$, par

$$\psi(i)(v) = \phi(i.v)$$

(Ici, ϵ dénote la séquence vide et _._ l'ajout d'un élément en tête de séquence.)

La fonction Π décompose en fait les éléments de O^{I^*} en couples de $O \times (O^{I^*})^I$, de la façon suivante. Si ϕ est une fonction de I^* dans O :

- $\phi(\epsilon)$ est la sortie associée à l'état courant ;
- après une entrée i, on arrive dans un état décrit par une fonction $\psi(i) : I^* \to O$, qui à toute suite d'entrées v associe la même sortie que la fonction ϕ appliquée à la suite d'entrées $b.v$.

◇

Comme il a déjà été dit plus haut, tout foncteur n'admet pas nécessairement une coalgèbre terminale. Considérons par exemple le foncteur $\mathcal{P}$ de l'ensemble des parties, correspondant aux systèmes non-déterministes. Si (W, δ) est une coalgèbre terminale pour ce foncteur, alors d'après le théorème III.4, δ est un isomorphisme, c'est-à-dire que W et $\mathcal{P}(W)$ sont isomorphes. Or la cardinalité de $\mathcal{P}(W)$ est strictement supérieure à celle de W, une coalgèbre terminale ne peut donc pas exister pour un tel foncteur. En revanche, le foncteur de l'ensemble des parties est restreint aux parties finies, il admet une coalgèbre terminale, de la même manière que les foncteurs polynomiaux, comme l'établit le théorème suivant.

THÉORÈME III.5. [Rut00] *Tout foncteur polynomial de Kripke fini admet une coalgèbre terminale.*

1.4 Quasi-covariétés

Nous aurons besoin, plus loin dans cette thèse, d'un résultat concernant l'existence d'une coalgèbre terminale pour une certaine classe de coalgèbres d'un foncteur donné. Cette classe, appelée quasi-covariété, est caractérisée par le fait d'être fermée sous certaines opérations, à savoir le quotient et le coproduit (voir l'annexe A pour la définition de cette notion). Elle admet alors une coalgèbre terminale qui est une sous-coalgèbre de la coalgèbre terminale pour ce foncteur, au sens défini comme suit :

DÉFINITION III.6 — *Sous-coalgèbre.*

Soient (E, α) une coalgèbre et S un sous-ensemble de E muni de l'injection canonique $i : S \to E$. S'il existe une structure de transition β sur S telle que $i : (S, \beta) \to (E, \alpha)$ est un morphisme de coalgèbre, alors (S, β) est appelée *sous-coalgèbre* de (E, α).

DÉFINITION III.7 — *(Quasi-)covariété.*

Une *quasi-covariété* est une classe de coalgèbres fermée par coproduit et par quotient. Une *covariété* est une quasi-covariété fermée par sous-coalgèbre.

THÉORÈME III.6. [Kur01] *Soit $\mathcal{F}$ un foncteur admettant une coalgèbre terminale. Alors toute quasi-covariété de $\mathsf{Coalg}(\mathcal{F})$ admet une coalgèbre terminale pleinement abstraite, c'est-à-dire qui est une sous-coalgèbre de la coalgèbre terminale de $\mathcal{F}$.*

Ce résultat est le dual d'un résultat d'algèbre universelle établi par Garrett Birkhoff [Bir35], qui montre l'existence d'une algèbre initiale pour toute classe d'algèbres fermée par produit, sous-algèbre et quotient. Dans ce résultat, l'algèbre initiale est un quotient de l'algèbre initiale de la catégorie des algèbres d'un foncteur donné.

2 Logique modale du premier ordre

De même que pour la logique du premier ordre, la présentation que nous allons donner de la logique modale du premier ordre suit le schéma de la définition I.17 des logiques générales.

2.1 Syntaxe

Le principe est ici d'étendre la logique du premier ordre en introduisant la notion de modalité. Une modalité permet de modifier la nature d'une proposition en produisant une autre proposition dont la valeur de vérité ne dépend pas de façon purement fonctionnelle de celle de la proposition initiale. Cela permet d'exprimer des notions telles que la possibilité, l'obligation, le futur, le devoir, la connaissance, *etc.* En informatique, les modalités sont le plus souvent utilisées pour modéliser la dépendance de la valeur de vérité d'une proposition au temps ou à l'exécution d'un programme. On va vouloir exprimer le fait qu'il existe un moment à partir duquel une certaine propriété est vraie, par exemple, ou bien qu'après l'exécution d'une fonction, une certaine variable est affectée d'une valeur donnée.

La logique modale présentée ici est construite au-dessus de la logique du premier ordre, de la même manière que la logique modale habituelle est construite comme une extension de la logique propositionnelle. Les formules de la logique modale propositionnelle sont construites inductivement à partir des éléments de base de la logique propositionnelle, c'est-à-dire des variables propositionnelles et des connecteurs booléens (négation, conjonction, disjonction et implication), auxquels est ajouté un nouvel élément de construction appelé modalité, généralement noté $\Box$. Elles sont construites inductivement selon les règles suivantes :

$$\varphi ::= p \mid \mathit{true} \mid \mathit{false} \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid \Box\varphi$$

où p est une variable propositionnelle. Une modalité est un opérateur qui a pour principale caractéristique de ne pas être vérifonctionnel. Tandis que la valeur de vérité d'une formule $p \wedge q$, où p et q sont des variables propositionnelles, dépend uniquement des valeurs de vérité de p et de q, il n'est pas possible de déduire la valeur de vérité de la formule $\Box p$ à partir de la valeur de vérité de p.

Signatures. Les formules de la logique modale du premier ordre sont construites à partir des mêmes briques de base que la logique du premier ordre. Une signature de cette logique est donc un triplet constitué d'un ensemble de sortes, d'un ensemble d'opérations et d'un ensemble de prédicats. Cependant, une différence est introduite ici entre une partie dite « fixe » des données et une partie dite « dynamique ». Certaines composantes de l'ensemble des données spécifiées sont considérées de la même manière que dans le cadre de la logique du premier ordre, c'est-à-dire sont pourvues d'une interprétation fixe. Par exemple, les entiers naturels munis des opérations et des prédicats qui leur sont associés, sont interprétés de la même manière dans tous les états. Les autres composantes vont être affectées d'une interprétation qui va évoluer selon une certaine dynamique, qui va changer selon l'état du système décrit. L'ensemble des sortes est alors partitionné en deux sous-ensembles : l'un dont les éléments sont des sortes dites « observables » représentant la partie fixe des données ; l'autre composé de sortes dites « non observables » qui représente la partie dynamique. Les opérations manipulant les données sont différenciées selon qu'elles manipulent des éléments de sorte observable ou non. Les opérations sur la partie fixe des données ne manipulent que des éléments de sorte observable et ont, de ce fait, une interprétation fixe. Les opérations liées à la partie dynamique font intervenir

des éléments de sorte non observable, caractéristiques de l'état. Leur interprétation évolue alors selon l'état, c'est-à-dire selon l'interprétation des éléments de sorte non observable qu'elles manipulent. De la même manière, les prédicats sont partagés en plusieurs catégories. Les prédicats sur la partie fixe des données ne prennent en argument que des éléments de sorte observable, tandis que les prédicats sur la partie dynamique ont parmi leurs arguments des éléments de sorte non observable, leur interprétation dépendant de l'état. On a alors la définition suivante d'une signature de la logique modale du premier ordre.

DÉFINITION III.8 — *Signature.*

Une *signature* Σ est un triplet (S, F, R) où :

- S est un ensemble dont les éléments sont appelés *sortes*, muni d'une partition S_{obs} et T où S_{obs} désigne le sous-ensemble des sortes dites *observables*, et T celui des *sortes non observables* ou *cotypes* ;
- F est un ensemble de noms d'opérations où chaque nom f est muni d'une arité dans $S^* \times S$. F est muni d'une partition F_{obs} et $(F_s)_{s \in T}$ où
  - F_{obs} est le sous-ensemble des noms d'*opérations sur les données* : pour tout $f : s_1 \times \ldots \times s_n \to s \in F_{obs}, s_1, \ldots, s_n, s \in S_{obs}$,
  - pour tout $s \in T$, F_s est le sous-ensemble des noms d'*observateurs de la sorte* s : toute opération $f \in F_s$ a le profil $s_1 \times \ldots \times s_n \times s \to s'$ avec $s_1, \ldots, s_n \in S_{obs}$. Si $s' \in S_{obs}$, f est appelé *observateur de sorte observable* ou *attribut*, si $s' \in T$, il est appelé *observateur de sorte non observable* ou *méthode* ;
- R est un ensemble de noms de prédicats où chaque nom r est muni d'une arité dans S^+. R est muni d'une partition $R_{obs} \amalg (R_s)_{s \in T}$ où
  - R_{obs} est le sous-ensemble des noms de *prédicats sur les données* : pour tout $r : s_1 \times \ldots \times s_n \in R_{obs}$, $s_1, \ldots, s_n \in S_{obs}$,
  - pour tout $s \in T$, R_s est le sous-ensemble des noms de *prédicats sur la sorte* s : tout prédicat $r \in R_s$ a le profil $s_1 \times \ldots \times s_n \times s$ avec $s_1, \ldots, s_n \in S_{obs}$.

Les opérations et les prédicats dépendant de l'état ne font en fait intervenir qu'un seul argument de sorte non observable. Ceci est dû au type de modèles qui seront utilisés pour interpréter la signature dans un monde mathématique. En effet, l'approche coalgébrique ne permet pas de considérer les observateurs n-aires où $n \neq 1$.

Les sortes observables sont celles provenant de l'environnement local. L'ensemble des sortes non observables forme un espace d'états multi-sorte, les observateurs permettant soit de produire directement une valeur observable, soit de faire évoluer l'état.

EXEMPLE III.16 — *Listes infinies.*

On donne ici la signature d'une liste de longueur infinie. On dispose d'une sorte observable qui représente les éléments de la liste. La sorte non observable est la sorte $Liste$. On observe une liste au travers de son élément de tête et du reste de la liste. L'opération $tête$ donnant la tête de la liste est un observateur de la sorte $Liste$ de sorte observable $Élem$, il donne une information sur la liste sans la modifier, tandis que l'opération $queue$ est un observateur de la sorte $Liste$ de sorte non observable, il fait évoluer la liste dans un autre état.

sort $\acute{E}lem$

cotype $Liste$

obs $t\hat{e}te : Liste \to \acute{E}lem$

$queue : Liste \to Liste$

◇

EXEMPLE III.17 — *Machines de Moore.*

Une machine de Moore est un automate à états finis dans lequel les sorties sont déterminées uniquement par l'état courant et ne dépendent pas directement des entrées. Une machine de Moore comprend une sortie pour chaque état et évolue d'état en état grâce aux entrées reçues. On a donc deux sortes observables, les entrées et les sorties, et une sorte non observable, l'état. Chaque état peut produire une sortie par l'observateur $observe$. Enfin, un état évolue en un autre grâce à un paramètre de sorte $Entr\acute{e}e$ par l'observateur $suiv$ de sorte non observable.

sorts $Entr\acute{e}e, Sortie$

cotype $\acute{E}tat$

obs $suiv : \acute{E}tat \times Entr\acute{e}e \to \acute{E}tat$

$observe : \acute{E}tat \to Sortie$

◇

EXEMPLE III.18 — *Arbres de profondeur infinie à branchement infini.*

Un arbre est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément est appelé nœud, le nœud initial étant appelé racine. Un arbre de profondeur infinie possède au moins une branche, c'est-à-dire une suite de nœuds, de longueur infinie. De plus, dans un arbre à branchement infini, le nombre de fils d'un nœud n'est pas borné.Chaque fils étant lui-même la racine d'un arbre, l'ensemble des fils d'un nœud est en fait un ensemble d'arbres, qu'on appelle également forêt.

On dispose d'une sorte observable représentant les éléments contenus dans les nœuds de l'arbre et de deux sortes non observables $Arbre$ et $For\hat{e}t$. On peut connaître l'élément contenu dans la racine d'un arbre par l'observateur $\acute{e}tiq$ de sorte observable $\acute{E}lem$, ainsi que l'ensemble de ses fils par l'observateur *fils* de sorte non observable $For\hat{e}t$. Dans une forêt, on accède au premier arbre par l'observateur *premier* de sorte non observable $Arbre$ et au reste de la forêt par l'observateur de sorte non observable $For\hat{e}t$.

sort $\acute{E}lem$

cotypes $Arbre, For\hat{e}t$

obs $\acute{e}tiq : Arbre \to \acute{E}lem$

$fils : Arbre \to For\hat{e}t$

$premier : For\hat{e}t \to Arbre$

$reste : For\hat{e}t \to For\hat{e}t$

◇

Les morphismes de signatures sont définis de façon similaire aux morphismes entre signatures du premier ordre, à la précision près qu'ils font correspondre les sortes observables entre elles et les sortes non

observables entre elles.

DÉFINITION III.9 — *Morphisme de signatures.*

Soient $\Sigma = (S, F, R)$ et $\Sigma' = (S', F', R')$ deux signatures. Un *morphisme de signatures* $\eta : \Sigma \to \Sigma'$ est défini par :

- une application $\eta_S : S \to S'$ telle que $\eta_S(S_{obs}) \subseteq S'_{obs}$ et $\eta_S(T) \subseteq T'$;
- une application $\eta_F : F \to F'$ telle que pour tout $f : s_1 \times \ldots \times s_n \to s \in F$, $\eta_F(f) : \eta_S(s_1) \times \ldots \times \eta_S(s_n) \to \eta_S(s)$;
- une application $\eta_R : R \to R'$ telle que pour tout $r : s_1 \times \ldots \times s_n \in R$, $\eta_R(r) : \eta_S(s_1) \times \ldots \times \eta_S(s_n)$.

DÉFINITION III.10 — *Catégorie des signatures.*

La *catégorie des signatures*, notée Sig, est la catégorie dont les objets sont les signatures et dont les morphismes sont les morphismes de signatures.

Formules. On construit les termes inductivement à partir des éléments de la signature, de la même manière qu'en logique du premier ordre, à une exception près : un terme construit sur un observateur de la sorte s ne fait pas intervenir cette sorte. Cela permet de considérer l'état comme implicite, comme il est habituel de le faire dans le cadre des logiques modales.

DÉFINITION III.11 — *Termes.*

Soit une signature $\Sigma = (S, F, R)$. Soit V un ensemble de variables sur Σ. L'ensemble des *termes avec variables*, noté $T_\Sigma(V)$, est défini de la façon suivante :

- pour tout $x \in V_s$, $x \in T_\Sigma(V)_s$;
- pour tout $f : s_1 \times \ldots \times s_n \to s \in F_{obs}$ opération sur les données, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_s$;
- pour tout $f : s_1 \times \ldots \times s_n \times s \to s' \in F_s$ observateur, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, $f(t_1, \ldots, t_n) \in T_\Sigma(V)_{s'}$.

Les formules sont construites sur le même schéma que les formules du premier ordre pour ce qui est des formules construites à partir des prédicats sur les données, des connecteurs booléens et des quantificateurs. Les formules spécifiques à la logique modale sont celles construites à partir des prédicats sur une sorte non observable, ainsi que celles faisant intervenir la notion de modalité.

Une modalité dans la logique modale du premier ordre est indexée par un terme du premier ordre de sorte non observable. Ainsi, au lieu de disposer d'une unique modalité $\Box$, on a autant de modalités $\Box_t$, notées également $[t]$, que de termes de sorte non observable construits sur la signature. Une formule $[t]\varphi$ signifie intuitivement « tout état suivant, accessible par t, valide φ ». De la même manière qu'il existe en logique propositionnelle une modalité duale à $\Box$, notée $\Diamond$ et définie par $\Diamond\varphi = \neg\Box\neg\varphi$, il existe dans la logique modale présentée ici une modalité duale $\langle t \rangle$ pour chaque terme t de sorte non observable. La signification intuitive d'une formule $\langle t \rangle\varphi$ est la suivante : « il existe un état suivant, accessible par t, qui valide φ ».

Dans le cadre des logiques temporelles, on dispose d'une modalité notée G exprimant le fait qu'une propriété « est toujours vérifiée », c'est-à-dire « est vérifiée dans tous les états futurs », et d'une modalité notée F exprimant le fait qu'une propriété « sera vérifiée dans un état futur ». Lorsque les modalités sont indexées

par des termes, cela revient à dire que dans tous les états accessibles par un nombre fini d'applications de t, φ est validée, ou qu'il existe un état accessible par un nombre fini d'applications de t dans lequel φ sera validée. La première propriété s'exprime également de la manière suivante : φ est validée dans l'état courant, puis elle l'est dans tous les états suivants accessibles par t ($[t]\varphi$ est validée), puis elle l'est également dans tous les états accessibles par deux applications de t ($[t][t]\varphi$ est validée), *etc.* On exprime alors intuitivement cette propriété par la formule infinie $\varphi \wedge [t]\varphi \wedge [t][t]\varphi \ldots$, abrégée grâce à la modalité $[t*]$ en $[t*]\varphi$. La deuxième propriété peut être vue comme la disjonction suivante : soit φ est validée dans l'état courant, soit elle l'est dans un des états suivants accessibles par t ($\langle t\rangle\varphi$ est validée dans l'état courant), soit elle est validée dans un des états accessibles par deux applications de t ($\langle t\rangle \langle t\rangle\varphi$ est validée), *etc.* La formule infinie $\varphi \vee \langle t\rangle\varphi \vee \langle t\rangle \langle t\rangle\varphi \ldots$, abrégée en $\langle t*\rangle\varphi$, donne intuitivement le sens de cette propriété. On retrouve les modalités G (« toujours ») et F (« finalement ») de la logique temporelle.

Les modalités comportant une étoile permettent ainsi de répéter une seule observation un nombre arbitrairement grand de fois. Cependant, on peut parfois vouloir exprimer le fait qu'un groupe d'observateurs peut être répété. Les observations exprimées par des termes de sorte non observable peuvent alors être groupées en n-uplets $\{t_1, \ldots, t_n\}$ de façon à pouvoir écrire les modalités $[\{t_1, \ldots, t_n\}]$ et $\langle\{t_1, \ldots, t_n\}\rangle$ dénotant respectivement la conjonction et la disjonction des formules obtenues pour les modalités individuelles correspondantes. On remarque que dans le cas où ces modalités ne comportent pas d'étoile, les formules $[\{t_1, \ldots, t_n\}]\varphi$ et $\langle\{t_1, \ldots, t_n\}\rangle\varphi$ peuvent être exprimées explicitement comme une conjonction ou une disjonction, ce qui est impossible dans le cas où elles comportent une étoile.

DÉFINITION III.12 — *Formules.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. L'ensemble des *formules* sur Σ, noté $For(\Sigma)$, est l'ensemble défini de la façon suivante :

- $true, false \in For(\Sigma)$;
- pour tout $r : s_1 \times \ldots \times s_n \in R_{obs}$, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, $r(t_1, \ldots, t_n) \in For(\Sigma)$;
- pour tout $r : s_1 \times \ldots \times s_n \times s \in R_s$, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, $r(t_1, \ldots, t_n) \in For(\Sigma)$;
- pour tout $\varphi \in For(\Sigma)$, $\neg\varphi \in For(\Sigma)$;
- pour tout $\varphi, \psi \in For(\Sigma)$, $\varphi \wedge \psi, \varphi \vee \psi, \varphi \Rightarrow \psi \in For(\Sigma)$;
- pour tout $x \in V$, pour tout $\varphi \in For(\Sigma)$, $\forall x\varphi, \exists x\varphi \in For(\Sigma)$;
- pour tout $s \in T$, pour tout $t \in T_\Sigma(V)_s$, pour tout $\varphi \in For(\Sigma)$, $[t]\varphi, \langle t\rangle\varphi, [t*]\varphi, \langle t*\rangle\varphi \in For(\Sigma)$;
- pour tous $s_1, \ldots, s_n \in T$, pour tout $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, pour tout $\varphi \in For(\Sigma)$, $[\{t_1, \ldots, t_n\}]\varphi, \langle\{t_1, \ldots, t_n\}\rangle\varphi, [\{t_1, \ldots, t_n\}*]\varphi, \langle\{t_1, \ldots, t_n\}*\rangle\varphi \in For(\Sigma)$.

On note $M_\Sigma(V)$ l'ensemble des modalités

$$\{[t], \langle t\rangle, [t*], \langle t*\rangle, [\{t_1, \ldots, t_n\}], \langle\{t_1, \ldots, t_n\}\rangle, [\{t_1, \ldots, t_n\}*], \langle\{t_1, \ldots, t_n\}*\rangle \mid t, t_1, \ldots, t_n \in T_\Sigma(V)\}$$

EXEMPLE III.19 — *Listes infinies.*

On donne ici une spécification du cotype $Liste$ dont la signature a été donnée à l'exemple III.16, et à laquelle on ajoute les deux observateurs de sorte non observable suivants :

$pair : Liste \to Liste$

$impair : Liste \to Liste$

L'observateur *pair* renvoie une liste contenant tous les éléments d'indice pair de la liste entrée en argument, tandis que l'observateur *impair* renvoie une liste composée des éléments d'indice impair. Les axiomes suivants spécifient le comportement de ces observateurs :

- $tête = n \Leftrightarrow [impair]\ tête = n$
- $[impair][queue]\varphi \Leftrightarrow [queue][queue][impair]\varphi$
- $[pair]\varphi \Leftrightarrow [queue][impair]\varphi$

Les deuxième et troisième axiomes sont en fait des schémas d'axiomes, c'est-à-dire qu'ils représentent l'ensemble de toutes leurs instances où φ a été remplacée par une formule quelconque.

◇

EXEMPLE III.20 — *Spécification d'un ruban infini de rationnels triés.*

On prend ici l'exemple d'un ruban de longueur infinie consistant en une suite de cases dans lesquelles sont stockés des nombres rationnels et munie d'une tête de lecture. La tête de lecture permet de lire le rationnel qui est stocké dans la case sur laquelle elle est pointée. Elle peut également se déplacer en avant et en arrière le long du ruban. Les rationnels sont triés sur le ruban. Une opération d'insertion permet de stocker un nouveau rationnel de façon à ce que le ruban reste trié. On dispose également d'un prédicat d'appartenance d'un rationnel au ruban.

spec RUBANINF =

types $Nat ::= 0 \mid s(Nat)$

$Rat ::= \_/\_(Nat, Nat)$

cotype $Ruban$

obs $tête : Ruban \to Rat$

$avant : Ruban \to Ruban$

$arrière : Ruban \to Ruban$

$insérer : Rat \times Ruban \to Ruban$

pred $estdans : Rat \times Ruban$

La spécification des entiers et des rationnels est celle donnée à l'exemple VI.1 du chapitre VI. On donne uniquement ici les axiomes permettant de spécifier le comportement des observateurs et du prédicat *estdans* sur la sorte *Ruban*. Les observateurs *avant* et *arrière* permettent de déplacer la tête de lecture en avant et en arrière sur le ruban, les rationnels étant triés de l'arrière vers l'avant. L'opération d'insertion est toujours possible. Insérer un élément déjà présent sur le ruban ne modifie pas celui-ci. Si l'élément à insérer est supérieur à l'élément pointé par la tête de lecture, mais inférieur à l'élément suivant, il doit être inséré entre ces deux éléments : l'élément inséré devient l'élément suivant, tandis que la suite du ruban est décalée d'une case vers l'avant. Lorsque l'élément à insérer est supérieur à l'élément suivant, il doit être inséré dans la suite du ruban. Si l'élément à insérer est inférieur à l'élément courant mais supérieur à l'élément précédent, l'insertion se déroule de la même façon en décalant le ruban vers l'arrière. S'il est inférieur à l'élément précédent, il doit être inséré à l'arrière du ruban. Après qu'un élément a été inséré, il appartient

au ruban. Lorsqu'un élément est stocké sur le ruban, soit il est égal à l'élément pointé par la tête de lecture, soit il est supérieur à cet élément et il existe une façon de positionner la tête de lecture de façon à la faire pointer sur cet élément en la déplaçant uniquement vers l'avant, soit il est inférieur à l'élément courant, et il faut déplacer la tête vers l'arrière un certain nombre de fois pour positionner la tête sur l'élément cherché.

vars $x, y : Nat ; n : Rat$

- $\langle\, avant \rangle\, true$
- $\langle\, arrière \rangle\, true$
- $\varphi \Leftrightarrow [avant][arrière]\varphi$
- $\varphi \Leftrightarrow [arrière][avant]\varphi$
- $egr(tête, n) \Rightarrow [arrière]\, infr(tête, n) \wedge [avant]\, infr(n, tête)$
- $\langle\, insérer(x/s(y)) \rangle\, true$
- $egr(tête, x/s(y)) \Rightarrow [insérer(x/s(y))]\, egr(tête, x/s(y))$
- $infr(tête, x/s(y)) \wedge [avant]\, infr(x/s(y), tête) \Rightarrow ([insérer(x/s(y))][avant]\, egr(tête, x/s(y)) \wedge$
  $([insérer(x/s(y))][avant][avant]\varphi \Leftrightarrow [avant][insérer(x/s(y))]\varphi))$
- $infr(x/s(y), tête) \wedge [arrière]\, infr(tête, x/s(y)) \Rightarrow$
  $([insérer(x/s(y))][arrière]\, egr(tête, x/s(y)) \wedge$
  $([insérer(x/s(y))][arrière][arrière]\varphi \Leftrightarrow [arrière][insérer(x/s(y))]\varphi))$
- $infr(tête, x/s(y)) \wedge [avant]\, infr(tête, x/s(y)) \Rightarrow$
  $([insérer(x/s(y))][avant]\varphi \Leftrightarrow [avant][insérer(x/s(y))]\varphi)$
- $infr(x/s(y), tête) \wedge [arrière]\, infr(x/s(y), tête) \Rightarrow$
  $([insérer(x/s(y))][arrière]\varphi \Leftrightarrow [arrière][insérer(x/s(y))]\varphi)$
- $[insérer(x/s(y))]\, estdans(x/s(y))$
- $estdans(n) \Leftrightarrow egr(tête, n) \vee (infr(tête, n) \wedge \langle\, avant* \rangle\, egr(tête, n))$
  $\vee\, (infr(n, tête) \wedge \langle\, arrière* \rangle\, egr(tête, n))$

$\diamond$

Il nous faut définir maintenant définir le foncteur For et la façon dont il transporte les morphismes de signatures. Un morphisme de signatures $\eta : \Sigma \to \Sigma'$ est prolongé aux termes de la même manière qu'un morphisme de signatures du premier ordre, puis aux formules de la façon suivante.

DÉFINITION III.13 — *Prolongement des morphismes de signatures aux ensembles de formules.*

Soient Σ et Σ' deux signatures. Soient V un ensemble de variables sur Σ et V' un ensemble de variables sur Σ'. Soit $\eta : \Sigma \to \Sigma'$ un morphisme de signatures. On note $\eta_V : V \to V'$ l'application injective telle que pour tout $s \in S$, pour tout $x \in V_s$, $\eta_V(x) \in V'_{\eta_S(s)}$.

Le prolongement de η aux variables et aux termes, noté $\overline{\eta}$ est défini de la même manière que dans le cadre du premier ordre II.8. Le *prolongement de* $\overline{\eta}$ *aux formules*, noté $\overline{\eta}^\sharp : For(\Sigma) \to For(\Sigma')$, est défini pour toute formule φ inductivement sur la forme de φ de la façon suivante :

– $\overline{\eta}^\sharp(true) = true$ et $\overline{\eta}^\sharp(false) = false$;

- si φ est de la forme $r(t_1, \ldots, t_n)$, alors $\overline{\eta}^\sharp(\varphi) = \eta_R(r)(\overline{\eta}(t_1), \ldots, \overline{\eta}(t_n))$;
- si φ est de la forme $\neg\psi$, alors $\overline{\eta}^\sharp(\varphi) = \neg\overline{\eta}^\sharp(\psi)$;
- si φ est de la forme $\psi \odot \chi$ avec $\odot \in \{\wedge, \vee, \Rightarrow\}$, alors $\overline{\eta}^\sharp(\varphi) = \overline{\eta}^\sharp(\psi) \odot \overline{\eta}^\sharp(\chi)$;
- si φ est de la forme $Qx\psi$ avec $Q \in \{\forall, \exists\}$, alors $\overline{\eta}^\sharp(\varphi) = Q\eta_V(x)\ \overline{\eta}^\sharp(\psi)$;
- si φ est de la forme ¡t! ψ avec ¡ ! $\in \{[\,], \langle\,\rangle, [\,*], \langle\,*\rangle\}$, alors $\overline{\eta}^\sharp(\varphi) =$ ¡$\overline{\eta}(t)$! $\overline{\eta}^\sharp(\psi)$;
- si φ est de la forme ¡$\{t_1, \ldots, t_n\}$! ψ avec ¡ ! $\in \{[\ ], \langle\ \rangle, [\ *], \langle\ *\rangle\}$, alors $\overline{\eta}^\sharp(\varphi) =$ ¡$\{\overline{\eta}(t_1), \ldots, \overline{\eta}(t_n)\}$! $\overline{\eta}^\sharp(\psi)$.

Par abus de notation, on notera également $\overline{\eta}$ le prolongement de $\overline{\eta}$ aux formules.

Le foncteur For associe donc à chaque signature son ensemble de formules et à chaque morphisme de signatures son prolongement aux formules.

DÉFINITION III.14 — *Foncteur* For.

Le foncteur $For : Sig \to \mathcal{S}et$ est le foncteur :
- qui associe à chaque signature $\Sigma \in |\ Sig\ |$ l'ensemble $For(\Sigma)$ des formules sur Σ ;
- qui associe à chaque morphisme de signatures $\eta : \Sigma \to \Sigma'$ le prolongement de η aux ensembles de formules $\overline{\eta} : For(\Sigma) \to For(\Sigma')$.

2.2 Sémantique

Modèles. Nous avons vu précédemment que les modèles de la logique modale propositionnelle sont des structures qu'il est possible de définir de façon plus générale en termes de coalgèbres. La logique modale présentée ici étant une extension de la logique modale propositionnelle, les signatures seront donc naturellement interprétées par des coalgèbres. Ces signatures comportent une partie fixe, dont l'interprétation ne dépend pas de l'état, et une partie dynamique. La partie fixe va alors être interprétée par un modèle du premier ordre, identique dans tous les états, tandis que la partie dynamique va être interprétée par une coalgèbre sur un foncteur donnant une interprétation aux observateurs et aux prédicats d'une sorte non observable. Les états de cette coalgèbre sont multi-sorte : un état est un n-uplet dont chaque composante est un élément de sorte s pour chaque sorte s non observable. On a alors la définition suivante d'un modèle d'une signature de la logique modale du premier ordre :

DÉFINITION III.15 — *Modèle*.

Soit $\Sigma = (S, F, R)$ une signature. On note $\Sigma_{obs} = (S_{obs}, F_{obs}, R_{obs})$ la sous-signature « observable » de Σ. Un *modèle* de Σ est un couple $(\mathcal{M}, (E, \alpha))$ où :
- $\mathcal{M}$ est un modèle de Σ_{obs} (voir la Définition II.10) ;
- (E, α), où $E = \prod_{s \in T} E_s$ et $\alpha : E \to \mathcal{F}E$, est une coalgèbre du foncteur $\mathcal{F}$ tel que $\mathcal{F}E = \prod_{s \in T} \mathcal{F}E_s$ et qui, pour tout $s \in T$, associe à E_s l'ensemble $\mathcal{F}E_s$ défini par :

$$\mathcal{F}E_s = \prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_s \\ s' \in S_{obs}}} M_{s'}^{M_{s_1} \times \ldots \times M_{s_n}} \times \prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_s \\ s' \in T}} E_{s'}^{M_{s_1} \times \ldots \times M_{s_n}} \times \prod_{r : s_1 \times \ldots \times s_n \times s \in R_s} 2^{M_{s_1} \times \ldots \times M_{s_n}}$$

Le modèle du premier ordre sous-jacent $\mathcal{M}$ ne sera précisé que lorsque ce sera nécessaire. Un modèle de Σ sera donc noté simplement (E, α).

Un observateur de sorte observable $f : s_1 \times \ldots \times s_n \times s \to s'$ est interprété par une application sur les données de $M_{s_1} \times \ldots \times M_{s_n}$ dans $M_{s'}$. Un observateur de sorte non observable $f : s_1 \times \ldots \times s_n \times s \to s'$ est interprété par une application modifiant la composante de sorte s' de l'état, $E_{s'}$, en fonction d'un n-uplet de paramètres dans $M_{s_1} \times \ldots \times M_{s_n}$. Un prédicat $r : s_1 \times \ldots \times s_n \times s$ avec $s \in T$ est interprété comme une application booléenne de $M_{s_1} \times \ldots \times M_{s_n}$ dans un ensemble à deux valeurs. On remarque que le foncteur donnant une interprétation aux éléments dynamiques de la signature est polynomial.

EXEMPLE III.21 — *Listes infinies.*

Un modèle associé à la signature des listes infinies donnée à l'exemple III.16 est constitué comme suit. Soit M un ensemble quelconque interprétant les éléments de sorte $\acute{E}lem$, $M = M_{\acute{E}lem}$. Soit $E = E_{Liste} = M^{\mathbb{N}}$ l'ensemble d'états, qui est l'ensemble des suites infinies d'éléments de M. Les observateurs $t\hat{e}te$ et $queue$ sont interprétés par les fonctions

$$\begin{array}{rcl} t\hat{e}te : M^{\mathbb{N}} & \to & M \\ \gamma & \mapsto & \gamma(0) \end{array} \qquad \begin{array}{rcl} queue : M^{\mathbb{N}} & \to & M^{\mathbb{N}} \\ \gamma & \mapsto & \gamma' \end{array}$$

où pour tout $n \in \mathbb{N}$, $\gamma'(n) = \gamma(n+1)$. Un modèle associé à cette signature est donc une coalgèbre $(M^{\mathbb{N}}, \langle\, t\hat{e}te, queue\rangle : M^{\mathbb{N}} \to M \times M^{\mathbb{N}})$ du foncteur $\mathcal{F}$ défini par $\mathcal{F}X = M \times X$.

Cette signature est complétée par les observateurs $pair$ et $impair$ donnés à l'exemple III.19, qui sont interprétés par les fonctions :

$$\begin{array}{rcl} pair : M^{\mathbb{N}} & \to & M^{\mathbb{N}} \\ \gamma & \mapsto & \gamma_0 \end{array} \qquad \begin{array}{rcl} impair : M^{\mathbb{N}} & \to & M^{\mathbb{N}} \\ \gamma & \mapsto & \gamma_1 \end{array}$$

où pour tout $n \in \mathbb{N}$, $\gamma_0(n) = \gamma(2n)$ et $\gamma_1(n) = \gamma(2n+1)$. On a alors la coalgèbre suivante :

$$(M^{\mathbb{N}}, \langle\, t\hat{e}te, queue, pair, impair\rangle : M^{\mathbb{N}} \to M \times M^{\mathbb{N}} \times M^{\mathbb{N}} \times M^{\mathbb{N}})$$

◇

Un modèle d'une signature est un couple formé d'un modèle du premier ordre et d'une coalgèbre. Un morphisme entre deux modèles d'une même signature se décompose alors en un morphisme entre modèles du premier ordre et un morphisme de coalgèbres.

DÉFINITION III.16 — *Morphisme de modèles.*

Soit $\Sigma = (S, F, R)$ une signature. Soient $(\mathcal{M}, (E, \alpha)), (\mathcal{M}', (E', \alpha')) \in Mod(\Sigma)$ deux modèles de Σ. Un *morphisme de modèles* $\mu : (\mathcal{M}, (E, \alpha)) \to (\mathcal{M}', (E', \alpha'))$ est défini par :

- un morphisme de modèles du premier ordre $\mu_M : \mathcal{M} \to \mathcal{M}'$ (voir la définition II.11) ;
- un morphisme de coalgèbres $\mu_C : (E, \alpha) \to (E', \alpha')$ (voir la définition III.3).

DÉFINITION III.17 — *Catégorie des modèles associée à une signature.*

Soit $\Sigma \in |\, Sig\,|$. La *catégorie des modèles de* Σ, notée $Mod(\Sigma)$, est la catégorie dont les objets sont les modèles de Σ et dont les morphismes sont les morphismes de modèles.

DÉFINITION III.18 — *Foncteur d'oubli U_σ.*

Soient Σ et Σ' deux signatures. Soit $\eta : \Sigma \to \Sigma'$ un morphisme de signatures. On note Σ_{obs} et Σ'_{obs} les sous-signatures « observables » de Σ et Σ' respectivement, et $\eta_{obs} : \Sigma_{obs} \to \Sigma'_{obs}$ la restriction de η aux sous-signatures observables. On définit $U_\eta : Mod(\Sigma') \to Mod(\Sigma)$ comme étant le foncteur :

- qui associe à chaque modèle $(\mathcal{M}', (E', \alpha')) \in |\, Mod(\Sigma')|$ le modèle $U_\eta(\mathcal{M}') = (\mathcal{M}, (E, \alpha))$ tel que :
  - $\mathcal{M} = U_{\eta_{obs}}(\mathcal{M}')$ (voir la définition II.13),
  - (E, α) où $E = \coprod_{s \in T} E'_s$ est la coalgèbre du foncteur $\mathcal{F}$ tel que $\mathcal{F}E = \prod_{s \in T} \mathcal{F}E_s$ et qui, pour tout $s \in T$, associe à E_s l'ensemble $\mathcal{F}E_s$ défini par :

$$\mathcal{F}E_s = \prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_s \\ s' \in S_{obs}}} {M'_{s'}}^{M'_{s_1} \times \ldots \times M'_{s_n}} \times \prod_{\substack{f : s_1 \times \ldots \times s_n \times s \to s' \in F_s \\ s' \in T}} {E_{s'}}^{M'_{s_1} \times \ldots \times M'_{s_n}} \times \prod_{r : s_1 \times \ldots \times s_n \times s \in R_s} 2^{M'_{s_1} \times \ldots \times M'_{s_n}}$$

- qui associe à chaque morphisme de modèles $\mu' : (\mathcal{M}'_1, (E'_1, \alpha'_1)) \to (\mathcal{M}'_2, (E'_2, \alpha'_2))$ où $\mu' = (\mu'_M, \mu'_C)$ un morphisme $U_\eta(\mu') : U_\eta((\mathcal{M}'_1, (E'_1, \alpha'_1))) \to U_\eta((\mathcal{M}'_2, (E'_2, \alpha'_2)))$ tel que :
  - $U_\eta(\mu')_M = U_{\eta_{obs}}(\mu'_M)$ (voir la définition II.13),
  - $U_\eta(\mu')_C = \mu_C$ où $\mu_{C_s} = \mu'_{C_s}$ pour tout $s \in S$.

Le foncteur Mod associe donc à chaque signature sa catégorie de modèles et à chaque morphisme de signatures le foncteur d'oubli correspondant.

DÉFINITION III.19 — *Foncteur Mod.*

Le foncteur $Mod : Sig \to \mathcal{C}at^{op}$ est le foncteur :

- qui associe à chaque signature $\Sigma \in |\, Sig \,|$ la catégorie $Mod(\Sigma)$ des modèles de Σ ;
- qui associe à chaque morphisme de signatures $\eta : \Sigma \to \Sigma'$ le foncteur d'oubli $U_\eta : Mod(\Sigma') \to Mod(\Sigma)$.

Relation de satisfaction. Les termes sont évalués ici à partir d'une interprétation des variables et d'un état, les observateurs étant munis d'une interprétation variant selon l'état. Les termes construits sur les opérations sur les données sont évalués de manière usuelle dans le modèle du premier ordre sous-jacent au modèle de la signature. Ils sont évalués de la même façon quel que soit l'état. Les termes construits sur les observateurs sont évalués suivant leur interprétation, donnée par le foncteur du modèle de la signature, et l'état courant. Lorsqu'il s'agit d'un observateur de sorte observable, un terme correspond à une valeur de l'ensemble des données, tandis que dans le cas d'un observateur de sorte non observable s', l'évaluation d'un terme est un état, dans lequel la composante de sorte s' a changé selon l'interprétation de cet observateur.

DÉFINITION III.20 — *Évaluation des termes.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. Soit $(\mathcal{M}, (E, \alpha))$ un modèle de Σ. Une *interprétation des variables* est une application $\nu : V \to M$, telle que pour tout $s \in S_{obs}$, pour tout $x \in V_s$, $\nu(x) \in M_s$.

À partir d'une interprétation des variables ν et d'un état $e = (e_s)_{s \in T} \in E$, on construit une *interprétation des termes* $\nu^\sharp_e : T_\Sigma(V) \to M$ de la façon suivante :

- si $x \in V_s$, alors $\nu_e^\natural(x) = \nu(x)$;
- si $f : s_1 \times \ldots \times s_n \to s \in F_{obs}$ est une opération sur les données et $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, alors $\nu_e^\natural(f(t_1, \ldots, t_n)) = f^{\mathcal{M}}(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$;
- si $f : s_1 \times \ldots \times s_n \times s \to s' \in F_s$ est un observateur de sorte observable, et $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, alors $\nu_e^\natural(f(t_1, \ldots, t_n)) = (\pi_f \circ \pi_s \circ \alpha)(e)(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$, où :
  - $\pi_s : E \to E_s$ est la projection canonique d'un état sur sa composante de sorte s,
  - π_f est la projection canonique de la composante de sorte s de l'état sur l'interprétation de f ;
- si $f : s_1 \times \ldots \times s_n \times s \to s' \in F_s$ est un observateur de sorte non observable et $(t_1, \ldots, t_n) \in T_\Sigma(V)_{s_1} \times \ldots \times T_\Sigma(V)_{s_n}$, alors $\nu_e^\natural(f(t_1, \ldots, t_n)) = e'$ tel que $e' = (e'_s)_{s \in T} \in E$ avec $e'_{s''} = e_{s''}$ pour tout $s'' \neq s'$, et $e_{s'} = (\pi_f \circ \pi_s \circ \alpha)(e)(\nu_e^\natural(t_1), \ldots, \nu_e^\natural(t_n))$.

Par abus de notation, on notera ν_e le prolongement de ν aux termes.

Nous avons donné plus haut une intuition de la signification des formules. Leur satisfaction par un modèle est bien sûre dépendante de l'état courant, ainsi que de l'interprétation des variables. Une formule construite sur un prédicat sur les données est évaluée dans le modèle du premier ordre sous-jacent, tandis que l'évaluation d'une formule construite sur un prédicat d'une sorte non observable dépend de l'interprétation de ce prédicat en l'état donné. Une formule de la forme $[t]\psi$ est validée en un état e si pour tout état « accessible par t », c'est-à-dire pour toute évaluation du terme t en l'état e, qui est un état e', ψ est validée. La satisfaction des autres formules découle de celle-là de manière naturelle.

DÉFINITION III.21 — *Satisfaction des formules.*

Soit $\Sigma = (S, F, R)$ une signature. Soit V un ensemble de variables sur Σ. Soit $(\mathcal{M}, (E, \alpha))$ un modèle de Σ. Soit $\nu : V \to M$ une interprétation des variables. Soit un état $e \in E$. Soit $\varphi \in For(\Sigma)$. La *satisfaction de φ par le modèle $(\mathcal{M}, (E, \alpha))$ pour l'interprétation ν et l'état e*, notée $(\mathcal{M}, (E, \alpha)) \models_{\nu,e} \varphi$, est définie de la façon suivante :

- si $\varphi = true$, alors $(\mathcal{M}, (E, \alpha)) \models_{\nu,e} \varphi$;
- si φ est de la forme $r(t_1, \ldots, t_n)$ avec $r \in R_{obs}$, alors $(\mathcal{M}, (E, \alpha)) \models_{\nu,e} \varphi$ si et seulement si $(\nu_e(t_1), \ldots, \nu_e(t_n)) \in r^{\mathcal{M}}$;
- si φ est de la forme $r(t_1, \ldots, t_n)$ avec $r \in R_s$, alors $(\mathcal{M}, (E, \alpha)) \models_{\nu,e} \varphi$ si et seulement si $(\nu_e(t_1), \ldots, \nu_e(t_n)) \in \pi_r \circ \pi_s(e)$, où π_r est la projection canonique de la composante de sorte s de l'état sur l'interprétation de r ;
- si φ est de la forme $[t]\psi$, alors $(\mathcal{M}, (E, \alpha)) \models_{\nu,e} \varphi$ si et seulement si, s'il existe $e' \in E$ tel que $\nu_e(t) = e'$, alors $(\mathcal{M}, (E, \alpha)) \models_{\nu,e'} \psi$.

Les autres modalités sont définies par les équivalences élémentaires suivantes :

- $\langle t \rangle \varphi \equiv \neg[t]\neg\varphi$;
- $[t*]\varphi \equiv \varphi \wedge [t][t*]\varphi$;
- $[\{t_1, \ldots, t_n\}]\varphi \equiv [t_1]\varphi \wedge \ldots \wedge [t_n]\varphi$.

Les connecteurs booléens sont définis de la façon habituelle.

On dit que $(\mathcal{M}, (E, \alpha))$ *satisfait* φ, noté $(\mathcal{M}, (E, \alpha)) \models \varphi$, si et seulement si $(\mathcal{M}, (E, \alpha)) \models_{\nu,e} \varphi$ pour tout $\nu : V \to M$ et pour tout $e \in E$.

THÉORÈME III.7. (Institution de la logique modale du premier ordre [MSRR06]) *Le quadruplet* $(Sig, For, Mod, \models)$ *où :*

- Sig *est la catégorie des signatures ;*
- For *est le foncteur donné à la définition III.14 ;*
- Mod *est le foncteur donné à la définition III.19 ;*
- $\models$ *est la relation de satisfaction de la définition III.21,*

est une institution.

2.3 Calcul

On associe à cette logique un calcul des séquents, très proche du calcul des séquents pour la logique du premier ordre. Les formules étant construites de la même manière sur les connecteurs booléens et les quantificateurs, toutes les règles du calcul des séquents présenté à la figure II.1 sont valables pour la logique modale du premier ordre. Il est cependant nécessaire de considérer également des règles associées aux différentes modalités. Les règles habituelles liée à la modalité $\Box$ dans le cadre de la logique modale propositionnelle sont les suivantes :

$$\frac{}{\Box(p \Rightarrow q) \Rightarrow (\Box p \Rightarrow \Box q)}\,\mathsf{K} \qquad \frac{p}{\Box p}\,\mathsf{Nec}$$

la première étant appelée axiome de distributivité de Kripke, et la seconde règle de nécessitation. L'axiome de distributivité, comme son nom l'indique, exprime la distributivité de la modalité $\Box$ sur l'implication : s'il est nécessaire que $p \Rightarrow q$, alors si on a nécessairement p, on a nécessairement q. La règle de nécessitation permet l'introduction de la modalité $\Box$: si une propriété p est valide, alors elle est nécessairement valide. C'est cette règle qui fait échouer le théorème de la déduction en logique modale. Ce théorème énonce que s'il est possible de prouver une proposition q à partir d'une hypothèse p, alors il est possible de prouver $p \Rightarrow q$. Or, par la règle de nécessitation, on prouve à partir de p que $\Box p$ est valide, alors qu'il est faux en général que $p \Rightarrow \Box p$.

Ces règles, adaptées à la logique modale du premier ordre et écrites selon le calcul des séquents, deviennent les suivantes :

$$\frac{}{\mid\!\sim [t](\varphi \Rightarrow \psi) \Rightarrow ([t]\varphi \Rightarrow [t]\psi)}\,\mathsf{K} \qquad \frac{\Gamma \mid\!\sim \varphi}{[t]\Gamma \mid\!\sim [t]\varphi}\,\mathsf{Nec}$$

où $[t]\Gamma = \{[t]\gamma \mid \gamma \in \Gamma\}$. Nous ne considérerons pas la règle K de distributivité dans la suite de cette thèse, cette règle n'étant pas utile dans le cadre de la sélection de tests par dépliage. À partir de la règle de nécessitation Nec, il est possible de dériver les règles d'inférence suivantes, qui nous seront utiles par la suite :

$$\frac{\Gamma \mid\!\sim \varphi}{[t*]\Gamma \mid\!\sim [t*]\varphi}\,\mathsf{Nec}* \qquad \frac{\Gamma \mid\!\sim \varphi}{[\{t_1,\ldots,t_n\}]\Gamma \mid\!\sim [\{t_1,\ldots,t_n\}]\varphi}\,\mathsf{Nec}^n$$

$$\frac{\Gamma \mid\!\sim \varphi, \Delta}{[t]\Gamma \mid\!\sim [t]\varphi, \langle t\rangle\Delta} \qquad \frac{\Gamma, \varphi \mid\!\sim \Delta}{[t]\Gamma, \langle t\rangle\varphi \mid\!\sim \langle t\rangle\Delta} \qquad \frac{\Gamma \mid\!\sim \Delta}{[t]\Gamma \mid\!\sim \langle t\rangle\Delta}$$

DÉFINITION III.22 — *Système formel associé au calcul des séquents.*

Soient $\Sigma = (S, F, R)$ une signature et V un ensemble de variables sur Σ. Le *système formel associé au calcul des séquents* est le triplet $C_\Sigma = (A_\Sigma, F_\Sigma, R_\Sigma)$ où :

- $A_\Sigma = F \cup R \cup V \cup \{\neg, \wedge, \vee, \Rightarrow, \forall, \exists, [,], \langle, \rangle, \{, \}, *, \mathrel{|\!\sim}, (,), `,'\}$;
- $F_\Sigma = \{\Gamma \mathrel{|\!\sim} \Delta \mid \Gamma, \Delta \subseteq For(\Sigma)\}$;
- R_Σ est l'ensemble des règles d'inférence du calcul des séquents étendu la logique modale du premier ordre.

DÉFINITION III.23 — *Système d'inférence de la logique modale du premier ordre.*

Le *système d'inférence associé à la logique modale du premier ordre* est le triplet $(Sig, For, \vdash)$ où Sig est la catégorie des signatures, For est le foncteur donné à la définition III.14 et $\vdash = (\vdash_\Sigma)_{\Sigma \in |Sig|}$ où $\vdash_\Sigma = \vdash_{C_\Sigma}$.

Nous avons vu au chapitre I que tout système d'inférence défini par un système formel est réflexif, transitif et monotone. Comme dans le cas de la logique du premier ordre, il est facile de prouver que système d'inférence associé à la logique modale du premier ordre possède la propriété de $\vdash$-translation. La correction du calcul des séquents pour la logique modale du premier ordre découle de la correction du calcul de la logique modale propositionnelle. La logique modale du premier ordre telle qu'elle a été présentée est donc une logique générale. On ne s'est pas intéressé ici à la complétude du calcul.

THÉORÈME III.8. *Le quintuplet* $(Sig, For, Mod, \models, \vdash)$ *est une logique générale.*

3 Logique modale conditionnelle positive

Nous présentons ici une restriction de la logique modale du premier ordre présentée à la section précédente. Cette logique a été la première logique modale dans laquelle nous avons étendu les techniques de dépliage, avant de les étudier dans le cadre plus général de la logique modale sous-jacente au langage de spécification COCASL. Cette logique modale, dite conditionnelle positive, est munie de l'égalité pour seul prédicat. Une signature de cette logique est donc uniquement composée d'un ensemble de sortes, partitionné en deux sous-ensembles de sortes observables et non observables, et d'un ensemble de noms d'opérations, partitionné en un ensemble d'opérations sur les données et un ensemble d'observateurs, eux-mêmes de sorte observable ou non observable. Les formules sont construites à partir des équations entre des termes de sorte observable et de la modalité $\Box$ indexée par les termes de sorte non observable : on appellera formule modale une formule de la forme $[\alpha_1] \ldots [\alpha_n] t = t'$, où $\alpha_1, \ldots, \alpha_n$ sont des termes de sorte non observable et t et t' sont des termes de sorte observable. Comme $n \geq 0$, une équation est également une formule modale. Une formule conditionnelle positive est alors une formule de la forme $\varphi_1 \wedge \ldots \wedge \varphi_{n-1} \Rightarrow \varphi_n$ où pour tout i, $1 \leq i \leq n$, φ_i est une formule modale.

Un modèle de la logique modale conditionnelle positive est un modèle de la logique modale du premier ordre auquel a été retirée l'interprétation des prédicats. Les termes et les formules sont interprétés de la même manière dans ce cadre restreint que dans le cadre général.

Le calcul associé à cette logique est celui présenté à la figure III.1. Les règles sont sensiblement les mêmes que pour la logique conditionnelle positive, excepté la règle de nécessitation, propre à la logique modale.

$$\frac{Ax \in Sp}{Sp \vdash Ax}\,\mathsf{Ax} \qquad \frac{}{Sp \vdash t = t}\,\mathsf{Ref} \qquad \frac{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t = t'}{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t' = t}\,\mathsf{Sym}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t = t' \quad Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t' = t''}{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t = t''}\,\mathsf{Trans}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t_1 = t'_1 \quad \ldots \quad Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ t_n = t'_n}{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow [\alpha_1]\ldots[\alpha_k]\ f(t_1,\ldots,t_n) = f(t'_1,\ldots,t'_n)}\,\mathsf{Cong}$$

$$\frac{Sp \vdash \bigwedge_{i\leq n} \varphi_i \Rightarrow \varphi}{Sp \vdash \bigwedge_{1\leq i\leq n} \sigma(\varphi_i) \Rightarrow \sigma(\varphi)}\,\mathsf{Subs} \qquad \frac{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow \varphi}{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \wedge \psi \Rightarrow \varphi}\,\mathsf{Mono}$$

$$\frac{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow \varphi}{Sp \vdash \bigwedge_{1\leq i\leq n} [\alpha]\varphi_i \Rightarrow [\alpha]\varphi}\,\mathsf{Nec} \qquad \frac{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \wedge \psi \Rightarrow \varphi \quad Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow \psi}{Sp \vdash \bigwedge_{1\leq i\leq n} \varphi_i \Rightarrow \varphi}\,\mathsf{MP}$$

FIG. III.1 – *Calcul pour des spécifications modales conditionnelles positives.*

CHAPITRE IV

Théorie du test à partir de spécifications axiomatiques

Le test est une méthode de validation très utilisée dans le développement de logiciels. L'objectif du test est de déceler les erreurs d'un système informatique. Son ambition n'est pas de prouver la correction d'un système mais seulement d'assurer un certain degré de confiance en ce système. En effet, il est possible que le système passe un certain nombre de tests en succès mais échoue à un seul test supplémentaire. Le principe est de confronter le système à un objet de référence qui décrit plus ou moins formellement le fonctionnement attendu du système, par exemple en termes de fonctionnalités ou de communications avec l'environnement. C'est la recherche de l'inadéquation entre le système et cet objet de référence qui va guider le processus de test. Celui-ci consiste à exécuter le système sur un sous-ensemble fini de ses entrées possibles. Trois phases interviennent dans le test d'un système :

1. La sélection d'un sous-ensemble pertinent des entrées possibles du système, appelé jeu de tests.
2. L'exécution du système sur le jeu de tests choisi.
3. La décision du succès ou de l'échec du jeu de tests, connu sous le nom de « problème de l'oracle ».

C'est la phase de sélection qui nous intéressera plus particulièrement. Elle représente le point crucial du processus de test. Différentes stratégies peuvent être adoptées pour la sélection d'un jeu de tests, définissant ainsi plusieurs approches du test.

Le test aléatoire Les données sont choisies aléatoirement parmi l'ensemble des entrées possibles [CH00, BBL97]. Elles peuvent également être sélectionnées relativement à une distribution probabiliste des entrées, représentative de l'utilisation réelle du système. Cette stratégie est très largement utilisée car elle est facile à mettre en place. Elle n'assure cependant pas toujours une bonne couverture de l'ensemble des entrées possibles, les données associées à une faible probabilité d'utilisation étant de fait négligées. Or c'est souvent sur ces entrées que des erreurs peuvent apparaître.

Le test structurel (ou test « boîte blanche ») Les tests sont sélectionnés à partir du code source du système qui est alors vu comme à travers une boîte transparente, dite aussi blanche. L'analyse du code, représenté par un graphe de contrôle ou un flot de données par exemple, permet de sélectionner les données, grâce à des critères de couverture relatifs à la structure du code [BCDM03, GBR00]. De nombreux outils ont été développés pour ce type de sélection, très utilisée pour tester de petites entités.

L'inconvénient de cette méthode est son incapacité à détecter des oublis par rapport à la spécification. En outre, n'ayant pas de référence de correction, cette méthode de sélection fournit peu de moyens de décider du succès ou de l'échec d'un test. C'est également le cas de la sélection aléatoire.

Le test fonctionnel (ou test « boîte noire ») Les tests sont sélectionnés à partir de la spécification du système, qu'elle soit formelle ou informelle, sans connaissance de l'implantation (le système est une boîte noire) [BGM91, DF93, BW04, KATP02]. Un jeu de tests peut alors être utilisé plusieurs fois à différentes étapes du développement du système, si la spécification reste la même au cours du développement. De plus, la spécification comme objet de référence permet d'aider à la décision du verdict de succès ou d'échec du jeu de tests. En effet, on peut généralement déduire de la spécification le résultat attendu pour un test donné, le problème de l'oracle est donc facilité, à condition d'avoir un moyen de comparer le résultat donné par la spécification et celui rendu par l'exécution du test.

Les stratégies de test structurel et de test fonctionnel reposent toutes deux sur le découpage selon certains critères du domaine d'entrée du système. Ces critères sont liés à la structure du code source du système sous test dans le cas du test structurel, à la spécification du système dans le cas du test fonctionnel. La formation d'un jeu de tests nécessite alors une dernière phase de sélection, appelée génération, qui consiste, après la division des entrées en sous-domaines, à choisir des tests dans chacun des sous-domaines.

Les travaux présentés dans cette thèse se situent dans le cadre du test fonctionnel et concernent plus particulièrement la définition de critères de sélection de jeux de tests à partir de spécifications axiomatiques [Ber91, BGM91, Gau95, LA96]. L'utilisation de langages formels pour la spécification de systèmes informatiques présente de grands avantages pour l'activité de test. Elle permet en particulier la formalisation de l'activité de test, ainsi que l'automatisation de certaines phases importantes du processus de test, comme le calcul des critères de sélection ou l'évaluation du verdict des tests. Les critères de sélection sont en général choisis par des experts, qui les établissent sur la base de leur connaissance du domaine d'application, ou en fonction de la nature plus ou moins critique du système. Utiliser des spécifications axiomatiques permet de dériver ces critères directement de la spécification, la plupart du temps en effectuant une couverture des axiomes de la spécification [MS02, Mar91, BW05]. L'évaluation des tests, donc le problème de l'oracle, est également facilitée par l'utilisation d'un langage formel pour la description du système. La calcul du verdict de succès ou d'échec provient de la comparaison du résultat fourni par l'exécution du test par le système et du résultat attendu obtenu à partir de la spécification. À la fois le calcul du résultat attendu et la comparaison entre les deux résultats sont automatisables si la spécification est décidable.

Il est évident que cette automatisation n'est possible qu'à condition de poser certaines hypothèses, à la fois sur le système à tester et sur les tests eux-mêmes. Se placer dans le cadre des spécifications formelles permet alors de formaliser ces hypothèses de test.

Dans la suite de ce chapitre, nous supposons donnée une logique générale

$$\mathcal{L} = (Sig, For, Mod, \models, \vdash)$$

1 Hypothèses de test et conditions d'observabilité

Le point de départ dans la définition d'un cadre de test consiste à poser les hypothèses sous lesquelles il est possible de raisonner sur le processus de test. Ces hypothèses concernent à la fois le système sous test et

les tests en eux-mêmes. Il s'agit d'établir à quelles conditions et en quels termes on peut définir le succès ou l'échec de la soumission d'un test au système, et ainsi définir une notion de correction du système.

Pour pouvoir tester un système contre sa spécification, il faut disposer d'un cadre sémantique général dans lequel la spécification et le système puissent être exprimés. Le cadre formel pour les spécifications étant posé par la notion de logique générale définie au chapitre I, il faut alors étudier de quelle manière il est possible de parler du système dans ce même cadre. Un système $\mathcal{S}$ définit un certain nombre d'opérations, une signature $\Sigma_{\mathcal{S}}$, et peut ainsi être considéré comme un moyen d'exécuter ces opérations. La première hypothèse est alors de considérer que son comportement peut être assimilé à un modèle de cette signature, c'est-à-dire $\mathcal{S} \in Mod(\Sigma_{\mathcal{S}})$. Il est de plus raisonnable de considérer que la signature $\Sigma_{\mathcal{S}}$ du système coïncide avec la signature Σ de sa spécification. On part en effet du principe que le système à tester n'est pas trop éloigné de sa spécification, qu'il donne effectivement une implantation des opérations de la spécification. La première hypothèse de test est alors la suivante :

HYPOTHÈSE 1. *Le système sous test est assimilé à un Σ-modèle.*

C'est l'hypothèse fondamentale sur laquelle repose le cadre formel de test qui va être présenté par la suite. Par abus de langage, on désignera par $\mathcal{S}$ à la fois le système sous test et son comportement en tant que Σ-modèle.

Le système en tant que modèle d'une signature donne d'une part une interprétation aux éléments de la signature et vérifie d'autre part un certain nombre de propriétés. La satisfaction de ces propriétés par le système peut alors être formalisée comme la validation des formules qui expriment ces propriétés par le système en tant que modèle. On peut alors écrire $\mathcal{S} \models_{\Sigma} \varphi$ si le système $\mathcal{S}$ valide la propriété exprimée par la formule φ. Cela amène la deuxième hypothèse nécessaire à la définition formelle du processus de test :

HYPOTHÈSE 2. *Les tests à soumettre au système sont des formules, construites sur Σ.*

Cependant, il n'est pas possible de soumettre au système n'importe quelle formule. Il faut en effet que le système soit capable d'évaluer le succès ou l'échec du test lorsqu'il lui est soumis. Soumettre un test dont il est impossible de conclure au succès ou à l'échec serait inutile. Les tests sont alors soumis à des restrictions d'observabilité. Dans le cadre des logiques du premier ordre, afin d'être évalué, un test ne doit par exemple pas contenir de variables non-instanciées. On dit alors que la formule est sans variables. De même, si le système contient des types de données ou des comportements non-observables, les tests ne pourront pas prendre en compte ces éléments. Un type de données qui ne serait pas muni d'un prédicat d'égalité dans le langage de programmation utilisé pour implanter le système, par exemple, ou de tout autre moyen de comparaison entre ses éléments, ne pourrait donner lieu à un test faisant intervenir ce type dans son résultat. Celui-ci ne pourrait en effet pas être évalué par le système comme réussi ou échoué, si ce dernier ne dispose pas d'un moyen de comparer le résultat du test au résultat attendu, déterminé par la spécification. De même, un comportement non-observable, c'est-à-dire interne au système, ne faisant pas intervenir d'entrées-sorties ou modifiant une composante de l'état du système pour laquelle on ne dispose pas d'observateur, ne pourra pas être pris en compte dans l'élaboration des tests. Les formules qui peuvent effectivement être évaluées par le système, c'est-à-dire interprétées comme « vraies » ou « fausses », sont appelées formules observables. On note Obs cet ensemble de formules. Les tests soumis au système, également appelés cas de test, feront

donc nécessairement partie de l'ensemble Obs.

Il faut remarquer que, si le système est un modèle de Σ, il ne satisfait pour autant pas nécessairement tous les axiomes de la spécification. Il est possible que ce ne soit pas un modèle de la spécification. Le système est donc bien un élément de $Mod(\Sigma)$, et c'est l'objectif du test de tenter de montrer que ce n'est pas un élément de $Mod(Sp)$.

2 Correction et exhaustivité

Ces hypothèses de test sur le système et ces restrictions sur l'observabilité des formules étant posées, il est maintenant possible de définir formellement la notion de correction d'un système par rapport à sa spécification. Cette notion est fortement liée à l'interprétation de la soumission des cas de test. Il est nécessaire de définir ce qu'est le succès (et donc également l'échec) de la soumission d'un test au système pour déterminer à quelles conditions le système peut être dit correct par rapport à sa spécification.

DÉFINITION IV.1 — *Succès et échec d'un test.*

Soit $\mathcal{S}$ un système sous test. Soit φ un cas de test. La soumission de φ au système $\mathcal{S}$ est un *succès* si et seulement si $\mathcal{S} \models_\Sigma \varphi$.

Un *jeu de tests* est un ensemble de cas de test. La soumission d'un jeu de tests T au système $\mathcal{S}$ est alors un *succès* si et seulement si pour tout cas de test φ de T, $\mathcal{S} \models_\Sigma \varphi$.

Un système doit pouvoir être considéré comme une implantation correcte de sa spécification si, en tant que Σ-modèle, il est indifférenciable d'un modèle de la spécification. La validité des formules pour un système sous test ne pouvant être vérifiée que sur les formules observables, on ne peut comparer ce système à un modèle de la spécification qu'au travers des formules observables qu'ils valident. La notion d'équivalence élémentaire vue au chapitre I permet justement cette comparaison. On dira alors que deux modèles sont observationnellement équivalents s'ils sont élémentairement équivalents par rapport à Obs. La correction d'un système par rapport à sa spécification est alors définie en suivant une approche observationnelle semblable à celles de [HWB97, ONS93].

DÉFINITION IV.2 — *Correction.*

Un système $\mathcal{S}$ est *correct* par rapport à sa spécification Sp via un ensemble de formules observables Obs, noté $Correct_{Obs}(\mathcal{S}, Sp)$, si et seulement s'il existe un modèle $\mathcal{M}$ de $Mod(Sp)$ tel que $\mathcal{M} \equiv_{Obs} \mathcal{S}$.

Il est maintenant possible de lier la correction d'un système avec le succès ou l'échec de la soumission d'un jeu de tests à ce système. Le but du test est de mettre le système en défaut, c'est-à-dire de détecter les erreurs d'un système qui ne serait pas correct. Un jeu de tests ne doit alors être passé en succès que par des systèmes corrects : on dit alors que le jeu de tests est valide. D'autre part, le processus de test ne doit pas rejeter de systèmes corrects, c'est-à-dire que tout système correct doit passer en succès le jeu de tests choisi. Celui-ci est alors dit non-biaisé.

DÉFINITION IV.3 — *Validité et non-biais.*

Soient une spécification Sp d'un système $\mathcal{S}$ et un ensemble de formules observables Obs.

Un jeu de tests T est dit *valide* pour $\mathcal{S}$ si et seulement si

$$\mathcal{S} \models_\Sigma T \Rightarrow Correct_{Obs}(\mathcal{S}, Sp)$$

Il est dit *non-biaisé* pour $\mathcal{S}$ si et seulement si

$$Correct_{Obs}(\mathcal{S}, Sp) \Rightarrow \mathcal{S} \models_\Sigma T$$

Un jeu de tests valide rejettera alors tout système incorrect, tandis qu'un jeu de tests non-biaisé ne rejettera pas de programmes corrects. Un jeu de tests idéal devra alors posséder conjointement les propriétés de validité et de non-biais. Le succès de la soumission d'un tel jeu de tests serait équivalent à la correction du système. Cela amène la notion de jeu de tests exhaustif.

Le point de départ de la recherche d'un jeu de tests à soumettre au système est la définition d'un jeu de tests idéal qui permettrait de démontrer la correction du système. L'existence d'un tel jeu de tests, dit exhaustif, assure qu'il est possible de prouver que le système est correct par rapport à sa spécification ou, de façon duale, qu'il existe pour tout système incorrect un cas de test qui le mettra en échec. Par conséquent, il est pertinent de tester ce système par rapport à sa spécification puisque sa correction peut être approchée de manière asymptotique, en soumettant au système un jeu de tests potentiellement infini. En tant que référence de correction, le jeu de tests exhaustif est donc approprié pour commencer le processus de sélection d'un jeu de tests fini de taille raisonnable.

DÉFINITION IV.4 — *Jeu de tests exhaustif.*

Soient $Sp = (\Sigma, Ax)$ une spécification et $Obs \subseteq For(\Sigma)$ un ensemble de formules observables. Soit $\mathcal{K} \subseteq Mod(\Sigma)$ une classe de systèmes. Un jeu de tests T est *exhaustif* pour $\mathcal{K}$ par rapport à Sp et Obs si et seulement si

$$\forall \mathcal{S} \in \mathcal{K}, \mathcal{S} \models_\Sigma T \Leftrightarrow Correct_{Obs}(\mathcal{S}, Sp)$$

Un jeu de tests exhaustif est donc, par définition, valide et non-biaisé pour les systèmes de $\mathcal{K}$.

Suivant la nature de la spécification, l'ensemble de formules observables et la classe de systèmes $\mathcal{K}$, un tel jeu de tests n'existe pas nécessairement. Comme il sera montré au chapitre V, dans le cas d'une spécification algébrique conditionnelle positive, un jeu de tests exhaustif n'existe que sous une condition forte sur les programmes appelée initialité. Cette condition étant presque aussi difficile à vérifier sur le système sous test que sa propre correction, elle restreint d'autant sa testabilité.

Parmi tous les jeux de tests possibles, le plus grand est l'ensemble des conséquences sémantiques observables de la spécification. En effet, pour être correct, le système doit être observationnellement équivalent à un modèle de la spécification, il doit donc valider exactement les mêmes formules observables que ce modèle. Or les formules validées par tous les modèles de la spécification sont, par définition, les conséquences sémantiques de cette spécification, dont l'ensemble est noté $Sp^\bullet$. Le système doit alors valider exactement les conséquences sémantiques de la spécification, qui sont de plus observables, c'est-à-dire les formules de $Sp^\bullet \cap Obs$.

On voit alors que la définition d'un jeu de tests exhaustif est très fortement liée à la sémantique choisie pour interpréter la spécification du système. Cette dépendance est naturelle, la correction d'un système étant définie en termes d'équivalence de modèles par rapport à un ensemble de formules d'une part, et le jeu de

tests exhaustif étant une référence de correction d'autre part. Prouver l'existence d'un jeu de tests exhaustif revient à montrer l'existence d'un modèle particulier qui soit équivalent à un modèle de la spécification. Nous développerons plus loin dans cette thèse la preuve d'exhaustivité du jeu de tests $Sp^{\bullet} \cap Obs$ pour Sp une spécification dans différents formalismes axiomatiques.

3 Critères de sélection

Lorsqu'il existe, le jeu de tests exhaustif est à la base de la sélection d'un jeu de tests praticable. En pratique, les experts appliquent des critères de sélection sur un jeu de tests de référence, afin d'en extraire un jeu de tests de taille raisonnable à soumettre au système. L'idée sous-jacente consiste à supposer que tous les tests répondant à un certain critère de sélection permettent de détecter la même classe de systèmes incorrects. Un critère de sélection est alors vu comme capturant un modèle de faute. Un critère de sélection couramment utilisé, appelé « hypothèse d'uniformité », postule que tous les cas de test d'un jeu de tests ont le même pouvoir de faire échouer le système [BGM91]. La validation d'un nombre fini quelconque de cas de test dans un jeu de tests est alors suffisante pour valider le jeu de tests entier. La méthode de sélection qui sera présentée par la suite tirera profit de cette hypothèse.

DÉFINITION IV.5 — *Hypothèse d'uniformité.*

Soit T un jeu de tests. Pour tout entier naturel $k > 0$, on note $\mathbb{T}_k \subseteq \mathcal{P}(T)$ l'ensemble des sous-ensembles de T de cardinal k. L'*hypothèse d'uniformité* est la suivante :

$$\forall k > 0, \forall T' \in \mathbb{T}_k, \mathcal{S} \models_{\Sigma} T' \Longrightarrow \mathcal{S} \models_{\Sigma} T$$

Une méthode classique pour sélectionner des jeux de tests à l'aide d'un critère de sélection consiste à diviser un jeu de tests de référence en une famille de sous-jeux de tests, de manière à conserver tous les cas de test. Cette méthode est appelée test par partition, bien que la famille de jeux de tests obtenue ne soit pas nécessairement une partition, au sens mathématique, du jeu de tests initial. Un critère de sélection définit de quelle manière un jeu de tests donné est subdivisé en une famille de jeux de tests.

DÉFINITION IV.6 — *Critère de sélection.*

Soit Exh un jeu de tests exhaustif. Un *critère de sélection* C est une application de $\mathcal{P}(Exh) \to \mathcal{P}(\mathcal{P}(Exh))$.

Pour tout $T \subseteq Exh$, $C(T)$ étant une famille de jeux de tests $\{T_i\}_{i \in I_{C(T)}}$ où $I_{C(T)}$ est l'ensemble des indices associé à l'application du critère C à T, on note $|C(T)| = \bigcup_{i \in I_{C(T)}} T_i$.

Appliquer un critère de sélection consiste à associer à un jeu de tests donné une famille de jeux de tests. Tous les cas de test d'un jeu de tests T_i de $C(T)$ sont supposés équivalents pour détecter des systèmes incorrects par rapport au modèle de faute capturé par T_i. L'application d'un critère de sélection à un jeu de tests T donné permet de raffiner ce jeu de tests. Les jeux de tests obtenus sont en effet plus « spécialisés » que le jeu de tests initial, ils correspondent à des modèles de fautes plus précis que celui associé à T. La construction d'un jeu de tests approprié associé à un critère de sélection nécessite alors de profiter du découpage obtenu par l'application de ce critère. Il s'agit de choisir les tests de façon à ne perdre aucun cas capturé par le critère.

DÉFINITION IV.7 — *Satisfaction d'un critère de sélection.*

Soient $T \subseteq Exh$ un jeu de tests et C un critère de sélection. Un jeu de tests T' *satisfait le critère* C *appliqué à* T si et seulement si

$$T' \subseteq |C(T)| \wedge \forall i \in I_{C(T)}, T_i \neq \emptyset \Rightarrow T' \cap T_i \neq \emptyset$$

Un jeu de tests qui satisfait un critère de sélection contient alors au moins un cas de test de chaque partie T_i du jeu de tests initial, lorsque celle-ci n'est pas vide. Un critère de sélection peut donc être considéré comme un critère de couverture, selon la manière dont il divise le jeu de tests initial. Il peut être utilisé pour couvrir un aspect particulier de la spécification. Dans le cadre de cette thèse, la définition des critères de sélection sera fondée sur une couverture des axiomes de la spécification.

La pertinence d'un critère de sélection est déterminée par le lien entre le jeu de tests initial et la famille de jeux de tests obtenue par l'application de ce critère.

DÉFINITION IV.8 — *Correction et complétude d'un critère de sélection.*

Soient T un jeu de tests et C un critère de sélection.

- C est *correct pour* T si et seulement si $|C(T)| \subseteq T$;
- C est *complet pour* T si et seulement si $|C(T)| \supseteq T$.

Ces propriétés sont essentielles pour la définition d'un critère de sélection approprié. La correction du critère assure que les cas de test sont bien sélectionnés dans le jeu de tests initial. L'application du critère n'ajoute alors aucun nouveau cas de test. Des cas de test supplémentaires risqueraient en effet de biaiser le jeu de tests, c'est-à-dire de faire échouer un système correct. Réciproquement, si le critère de sélection est complet, aucun test du jeu de tests initial n'est perdu. En effet, si des cas de test manquent, un système peut passer le jeu de tests en succès tout en étant incorrect, les tests manquant étant ceux qui auraient dû le faire échouer. Un critère correct et complet a donc la propriété de conserver exactement tous les cas de test du jeu de tests qu'il divise, c'est-à-dire de préserver le non-biais et la validité du jeu de tests initial.

Première partie

Test à partir de spécifications du premier ordre

Bien que le cadre général de test ait été défini pour des spécifications exprimées dans un formalisme axiomatique quelconque, la méthode de sélection par dépliage des axiomes a été principalement étudiée pour des spécifications conditionnelles positives. Nous présentons cette méthode en tant qu'état de l'art, toutes les procédures de dépliage proposées dans cette thèse en étant directement inspirées. La méthode de dépliage pour les spécifications du premier ordre sans quantificateurs est la généralisation de cette première procédure. Nous prouvons la correction et la complétude de cette procédure généralisée grâce à une démarche similaire à celle suivie dans le cas conditionnel positif. La procédure de dépliage définit en fait une stratégie de recherche d'arbres de preuve. Il faut alors montrer que la recherche selon cette stratégie n'est pas restrictive, c'est-à-dire que tout preuve peut être menée selon cette stratégie. Ceci fait appel à des résultats de normalisation d'arbres de preuve présentés dans l'annexe B.

Une des étapes préliminaires vers la sélection d'un jeu de tests effectif à soumettre au système est la définition d'un jeu de tests exhaustif à partir duquel effectuer cette sélection. Les conditions sous lesquelles l'existence d'un tel jeu de tests pouvait être assurée ont pourtant été peu étudiées dans les travaux sur la sélection de tests à partir de spécifications algébriques. Il s'avère que dans des cas simples, comme celui des spécifications purement équationnelles lorsque toutes les sortes sont observables, le jeu de tests qu'il est le plus naturel de définir est exhaustif sans conditions. Cependant, lorsqu'il existe des sortes pour lesquelles il n'est pas possible de décider de l'égalité entre deux éléments ou lorsque les axiomes des spécifications manipulées sont plus complexes que de simples équations, l'existence d'un jeu de tests exhaustif n'est assurée qu'à condition de poser des hypothèses parfois très fortes sur les spécifications ou sur le système sous test. Nous montrerons même, dans le cas des spécifications du premier ordre les plus générales, que prouver l'existence d'un jeu de tests exhaustif est équivalent à prouver la correction du système lui-même, ce qui interdit tout simplement l'utilisation de certaines formules dans les spécifications, à savoir les formules quantifiées existentiellement. Nous présentons alors des résultats d'exhaustivité aussi bien dans le cadre conditionnel positif que dans le cadre du premier ordre sans quantificateurs.

CHAPITRE V

Sélection à partir de spécifications conditionnelles positives

En fait de spécifications axiomatiques, les méthodes de test étudiées et développées jusqu'à présent ont principalement eu pour objet des spécifications dites algébriques, et en particulier des spécifications conditionnelles positives. Comme on l'a vu à la section 3 du chapitre II, l'aspect du système qu'elles permettent de décrire est essentiellement son comportement au travers des structures de données qu'il manipule. Ces spécifications possèdent également de bonnes propriétés algébriques pour la structuration [GTW78, GH78] de spécifications ainsi que leur prototypage défini à partir de techniques de réécriture [BN98]. La première procédure de dépliage, implantée dans l'outil LOFT [BGM91, Mar91, Mar95], est construite pour des spécifications équationnelles conditionnelles positives dont les axiomes peuvent être transformés en règles de réécriture [AABLM05]. C'est sous cette hypothèse que la correction et la complétude de la méthode ont été assurées. Depuis, cette méthode de sélection par dépliage a été généralisée [AABLM05] en ne supposant plus cette hypothèse sur les axiomes. C'est cette généralisation que nous présentons dans ce chapitre, comme état de l'art des méthodes de sélection fondées sur le dépliage des axiomes.

1 Étude des conditions d'exhaustivité

La première étape dans le processus de sélection d'un jeu de tests est la définition d'un jeu de tests de référence, exhaustif, d'où on pourra commencer la sélection. Il s'agit de définir un ensemble de formules observables, puis de trouver le plus grand ensemble de formules que le système doit vérifier (et qu'il suffit qu'il vérifie) pour assurer sa correction. Comme il a été dit au chapitre IV, une fois fixée la notion d'observabilité, le plus grand jeu de tests qu'un système doit vérifier est l'ensemble $Sp^{\bullet} \cap Obs$ des conséquences sémantiques observables de la spécification. C'est cet ensemble qui va servir de jeu de tests de référence pour la sélection d'un jeu de tests effectif à soumettre au système sous test.

En pratique, la plupart des méthodes de sélection de cas de test à partir de spécifications restreignent naturellement les tests à des équations sans variable sur des sortes particulières appelées sortes observables. Ces sortes sont celles pour lesquelles il existe un prédicat d'égalité dans le langage de programmation du système sous test. Par exemple, les sortes de base comme les caractères, les booléens ou les entiers naturels sont observables, car tous les langages de programmation possèdent un moyen effectif de comparer des

valeurs appartenant aux ensembles représentés par ces sortes. Au contraire, les structures de données plus complexes comme les listes, les tableaux, ou les ensembles ne sont pas toujours pourvues de moyens de comparaison, par exemple pour décider de l'égalité entre deux éléments de ces sortes. Ces sortes seront alors considérées comme non observables. Une équation entre deux termes de sorte non observable ne pourra donc pas être un cas de test qui puisse être soumis au programme sous test, le problème de l'oracle ne pouvant être résolu par ce programme. On dispose alors d'un sous-ensemble de l'ensemble S des sortes, noté S_{obs}, de sortes observables. L'ensemble des formules observables est le suivant :

$$Obs(S_{obs}) = \{t = t' \mid \exists s \in S_{obs}, t, t' \in T_{\Sigma_s}\}$$

Avec cette restriction sur l'ensemble des tests possibles, le jeu de tests $Sp^\bullet \cap Obs(S_{obs})$ n'est en général pas exhaustif pour tout programme. Considérons la spécification suivante de l'inversion d'une liste :

spec LISTES =
  sorts *Elem*, *Liste*
  ops $[\,] :\rightarrow Liste$
    $\_ :: \_ : Élem \times Liste \rightarrow Liste$
    $inverser : Liste \rightarrow Liste$
  var $L : Liste$
  • $inverser([\,]) = [\,]$
  • $inverser(inverser(L)) = L$
end

Si les deux sortes $Élem$ et $Liste$ sont observables, le jeu de tests $Sp^\bullet \cap Obs$ ne contient que les tautologies. En effet, l'ensemble des termes sans variable de sorte $Élem$ est vide, ce qui implique que l'ensemble des termes sans variable de sorte $Liste$ est également vide. Une première condition à imposer est alors celle selon laquelle pour toute sorte $s \in S$ de la signature l'ensemble T_{Σ_s} n'est pas vide. Ceci s'obtient aisément en considérant des signatures dites *raisonnables* [LeG93], c'est-à-dire qui contiennent au moins une constante pour chaque sorte de la signature. On remplace alors dans la spécification précédente la sorte $Élem$ par la sorte Nat dont les éléments sont construits à partir des opérations 0 et $s : Nat \rightarrow Nat$.

Cependant, cette hypothèse ne suffit pas pour résoudre le problème lié à l'observabilité des sortes non observables. En effet, si on restreint l'observabilité à la seule sorte Nat, alors les axiomes de l'opération $inverser$ ne peuvent pas être soumis au programme comme cas de test, étant des équations entre des éléments de sorte non observable $Liste$. Quelle que soit l'implantation de cette opération, même incorrecte, le programme passera en succès le jeu de tests $Sp^\bullet \cap Obs(S_{obs})$, qui n'est en fait composé que des tautologies $t = t$ sur les éléments de sorte Nat.

Il est connu que les équations non observables peuvent être observées au travers de ce qu'on appelle des contextes observables. Un contexte observable est une sorte de filtre, qui permet d'observer une certaine caractéristique d'un élément de sorte non observable dénoté par un terme. En observant cet élément au travers de différents filtres, de différents contextes observables, il est possible d'acquérir suffisamment de connaissances pour pouvoir le comparer avec un autre élément de cette même sorte. Considérons par exemple

la sorte $Liste$, munie des deux opérations $tête$ et $queue$ permettant respectivement de connaître le premier élément de la liste et la suite de la liste. Admettons que la sorte $Liste$ ne soit pas observable mais que la sorte de ses éléments le soit. Il est impossible *a priori* de comparer deux listes L et L' entre elles. Cependant, grâce à l'opération $tête$, on peut connaître le premier élément de chacune des deux listes, $tête(L)$ et $tête(L')$, et les comparer, puisque la sorte des éléments d'une liste est observable. Le terme $tête(\Box)$, où $\Box$ désigne un « trou », est un contexte observable. Ce filtre permet de connaître une caractéristique observable d'une liste, ici, son premier élément. De plus, si on applique l'opération $queue$ à chacune des deux listes qu'on veut comparer, on accède à la suite de chacune de ces listes, dont il est de nouveau possible de connaître le premier élément : $tête(queue(\Box))$ est alors également un contexte observable, qui permet de connaître le deuxième élément d'une liste. Il est possible de continuer la construction de contextes observables pour les listes en prenant à chaque fois la suite de la liste précédente. On obtient l'ensemble de contextes $\{tête(queue^n(\Box)) \mid n \geq 0\}$. En observant les deux listes L et L' au travers de tous les contextes de cet ensemble et puisque ces listes sont finies, on est capable de décider de l'égalité de ces deux listes. Cette notion de contexte observable capture la pratique du test qui consiste à appliquer aux deux membres d'une équation la même fonction, appliquée à des valeurs concrètes excepté à la position du $\Box$.

DÉFINITION V.1 — *Contexte observable.*

Soit une signature $\Sigma = (S, F, R)$ munie d'un ensemble de sortes observables $S_{obs} \subseteq S$. On pose $\Box = \{\Box_s : \rightarrow s \mid s \in S \setminus S_{obs}\}$ et $\Sigma_\Box = (S, F \cup \Box, R)$.

Un Σ-*contexte observable* c est un terme de $T_{\Sigma_\Box s}$ pour $s \in S_{obs}$, possédant exactement une occurrence du symbole $\Box_{s'}$ de l'ensemble $\Box$, tel que ce soit l'unique occurrence d'un symbole de $\Box$ dans c. Le contexte c est alors dit de sorte s' et noté $c : s'$. L'application d'un contexte $c : s'$ à un terme $t \in T_\Sigma(V)_{s'}$, noté $c[t]$, est le terme obtenu en remplaçant le symbole $\Box_{s'}$ par le terme t dans c. On note Ctx l'ensemble des contextes observables.

Un contexte observable est dit minimal si aucun de ses sous-termes stricts n'est un contexte observable. En effet, si un contexte c contient un contexte observable c' comme sous-terme strict, alors $c[t]$ peut être décomposé en $c''[c'[t]]$. Pour tous termes t et t', on a alors que $c[t] = c[t']$ si et seulement si $c'[t] = c'[t']$. Ces deux équations étant observables, la plus simple, $c'[t] = c'[t']$, suffit pour déduire que $c[t] = c[t']$. Dans la suite, tous les contextes seront considérés minimaux par défaut. On note $\mu\, Ctx$ l'ensemble des contextes observables minimaux.

PROPOSITION V.1. *Soit $Sp = (\Sigma, Ax)$ une spécification dont les axiomes sont des équations et telle que pour chaque sorte $s \in S$, $T_{\Sigma_s} \neq \emptyset$. Le jeu de tests suivant*

$$Exh = \{c[\sigma(t)] = c[\sigma(t')] \mid t = t' \in Ax, \sigma : V \rightarrow T_\Sigma, c \in \mu\, Ctx\}$$

est exhaustif pour tout programme dans $Mod(\Sigma)$ pour Sp et $Obs(S_{obs})$.

Preuve. Soit un programme $P \in Mod(\Sigma)$. Supposons que $P \models Exh$. Montrons que $Correct_{Obs(S_{obs})}(P, Sp)$.

Soit $T_\Sigma/_{\sim_P}$ le quotient de T_Σ où $\sim_P$ est la congruence sur T_Σ définie pour tout $t, t' \in T_{\Sigma_s}$ par :

$$t \sim_P t' \iff \begin{cases} P \models t = t' & \text{si } s \in S_{obs} \\ \forall c : s \in \mu\, Ctx, P \models c[t] = c[t'] & \text{sinon} \end{cases}$$

Par définition, $P \equiv_{Obs(S_{obs})} T_\Sigma/_{\sim_P}$. Montrons que $T_\Sigma/_{\sim_P} \in Mod(Sp)$. Soit $t = t'$ un axiome de Ax où $t, t' \in T_\Sigma(V)_s$. Soit $\iota : V \to T_\Sigma/_{\sim_P}$ une interprétation des variables. Par induction structurelle sur les termes de T_Σ, on peut montrer qu'il existe une substitution close $\sigma : V \to T_\Sigma$ telle que $\iota = q_{\sim_P} \circ \sigma$, où $q_{\sim_P} : T_\Sigma \to T_\Sigma/_{\sim_P}$ est le morphisme quotient. Il suffit de choisir σ telle que pour tout $x \in V$, si $\iota(x) = t$, alors $\sigma(x) = t'$ avec $t \sim_P t'$. Deux cas doivent être considérés :

- soit $s \in S_{obs}$. Dans ce cas, par hypothèse, $P \models c[\sigma(t)] = c[\sigma(t')]$ pour tout contexte $c : s \in \mu\, Ctx$. Or $\{c : s \mid c \in \mu\, Ctx, s \in S_{obs}\} = \{\Box_s\}$, on a donc $P \models \sigma(t) = \sigma(t')$. Comme $P \equiv_{Obs(S_{obs})} T_\Sigma/_{\sim_P}$, alors $T_\Sigma/_{\sim_P} \models \sigma(t) = \sigma(t')$, c'est-à-dire $\sigma(t) \sim_P \sigma(t')$. On a alors $q_{\sim_P} \circ \sigma(t) \sim_P q_{\sim_P} \circ \sigma(t')$, donc $T_\Sigma/_{\sim_P} \models_\iota t = t'$;
- soit $s \notin S_{obs}$. Dans ce cas, par hypothèse, $P \models c[\sigma(t)] = c[\sigma(t')]$ pour tout contexte $c : s \in \mu\, Ctx$. Par définition, on a donc $\sigma(t) \sim_P \sigma(t')$, et $T_\Sigma/_{\sim_P} \models_\iota t = t'$.

On suppose qu'il existe $\mathcal{M} \in Mod(Sp)$ tel que $\mathcal{M} \equiv_{Obs(S_{obs})} P$. Soit $t = t' \in Exh$. Par hypothèse, $\mathcal{M} \models t = t'$, donc $P \models t = t'$ également. □

COROLLAIRE . *Sous les mêmes hypothèses que la proposition V.1, le jeu de tests* $Sp^\bullet \cap Obs(S_{obs})$ *est exhaustif pour tout programme dans* $Mod(\Sigma)$ *pour* Sp *et* $Obs(S_{obs})$.

Preuve. Il est facile de montrer que $Exh = Sp^\bullet \cap Obs(S_{obs})$. □

Lorsque les axiomes ne sont pas de simples équations mais des formules conditionnelles positives, les tests étant des équations sans variable, l'exhaustivité du jeu de tests $Sp^\bullet \cap Obs(S_{obs})$ n'est pas immédiate. Établir ce résultat nécessite une hypothèse de test supplémentaire sur le système. Intuitivement, cette condition impose que le système se comporte comme l'algèbre initiale (et donc comme la spécification) sur les prémisses des instances sans variables des axiomes. Pour illustrer la nécessité de cette hypothèse, considérons la spécification suivante :

spec BASIQUE =
  types $Nat ::= 0 \mid s(Nat)$
    $Bool ::= true \mid false$
  vars $x : Nat$
  • $s(s(x)) = x \Rightarrow true = false$
end

Les conséquences sémantiques observables de cette spécification sont seulement les équations triviales $t = t$ pour tout terme clos t. En effet, les seules équations susceptibles d'êtres des conséquences sémantiques observables sont, exceptées les tautologies, des instances de $s(s(x)) = x$ ou $true = false$. Un modèle de cette spécification doit valider l'axiome $s(s(x)) = x \Rightarrow true = false$. Il le valide s'il ne valide pas

$s(s(x)) = x$, ou s'il valide $s(s(x)) = x$ et $true = false$. Par conséquent, l'équation $s(s(x)) = x$ n'est pas une conséquence sémantique observable de BASIQUE. Si le modèle ne valide pas $s(s(x)) = x$, il peut aussi bien valider $true = false$ que sa négation, cette équation n'appartient donc pas non plus à BASIQUE$^\bullet$. De ce fait, aucune des deux équations n'est validée par tous les modèles de la spécification et les conséquences sémantiques se réduisent aux tautologies. Considérons un système qui implante la sorte Nat par $\mathbb{Z}/_{2\mathbb{Z}}$, c'est-à-dire l'ensemble $\{0, 1\}$ dans lequel l'opération successeur est interprétée par l'application qui à 0 associe 1 et à 1 associe 0 (i.e. à tout élément n associe $(n+1) \mod 2$), au lieu de l'implanter par l'ensemble des entiers naturels, muni de l'opération successeur qui donne pour un entier n l'entier suivant $n+1$. On considère que les booléens sont interprétés par ce système de façon usuelle. Ce système valide tous les cas de test de $Sp^\bullet \cap Obs$ puisque ce sont des tautologies. Cependant, il n'est pas correct puisqu'il valide $s(s(x)) = x$ sans valider $true = false$. Il n'est donc pas équivalent à un modèle de la spécification.

Ce problème est en partie dû au fait que l'ensemble des formules qui va être soumis au système ne contient pas les axiomes de la spécification. Il faut donc imposer une condition supplémentaire pour s'assurer qu'un système qui valide les conséquences sémantiques observables de la spécification valide également les axiomes. Cette condition est une propriété appelée initialité, qu'on va imposer au système sous test pour certaines équations.

DÉFINITION V.2 — *Initialité.*

Soit une spécification $Sp = (\Sigma, Ax)$ d'un système $\mathcal{S}$. Soit $t = t'$ une Σ-équation. Le système $\mathcal{S}$ est dit *initial sur* $t = t'$ *pour* Sp si et seulement si

$$\mathcal{S} \models t = t' \Leftrightarrow Sp \models t = t'$$

Lorsqu'on considère $Sp^\bullet \cap Obs$ comme jeu de tests exhaustif, la partie de cette propriété exprimant le fait que l'équation $t = t'$ doit être validée par le système si elle est une conséquence de la spécification est redondante. C'est surtout l'implication inverse, ou plus exactement sa contraposée, qui est importante : le système ne doit pas valider l'équation si elle n'est pas une conséquence sémantique de la spécification. Pour établir l'exhaustivité de $Sp^\bullet \cap Obs$, cette condition va être imposée au système pour toutes les équations apparaissant dans les prémisses des instances sans variables des axiomes. L'initialité d'un système par rapport à sa spécification sur ces équations particulières assure alors que s'il valide $Sp^\bullet \cap Obs$, il valide les axiomes. En effet, pour un axiome donné,

- soit les prémisses sont des conséquences sémantiques de la spécification, et alors la conclusion aussi : comme le système valide $Sp^\bullet \cap Obs$, il valide en particulier cet axiome ;
- soit au moins une des prémisses $t = t'$ n'est pas une conséquence sémantique : le système étant initial sur ces prémisses, en particulier il ne valide pas $t = t'$, il valide donc l'axiome.

Il est donc impossible que le système valide toutes les prémisses d'un axiome sans valider sa conclusion.

Cependant, l'hypothèse d'initialité sur le système n'est pas suffisante pour établir l'exhaustivité de $Sp^\bullet \cap Obs(S_{obs})$. Une condition particulière sur les spécifications est également requise.

Nous avons vu qu'une égalité non observable peut être testée au travers de contextes observables. Cependant, ce n'est pas le cas des formules conditionnelles positives. Le problème réside dans le fait que, pour une équation $t = t'$, la validation de $c[t] = c[t']$ par un modèle pour tout contexte $c \in \mu\, Ctx$ n'implique pas la validation de $t = t'$. En effet, le nombre de contextes observables pour la sorte de t et t' peut ne pas

être suffisant. On peut considérer l'exemple de listes sur lesquelles on ne disposerait que de l'opération $t\hat{e}te$. Il est évident que la validation de $t\hat{e}te(L) = t\hat{e}te(L')$ pour deux listes L et L' n'implique pas l'égalité de ces listes. Considérons une spécification Sp contenant un axiome de la forme $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t'$, tel que pour tout i, $1 \leq i \leq n$, $t_i = t'_i$ soit une conséquence sémantique de la spécification. L'équation $t = t'$ est donc également une conséquence sémantique. Le jeu de tests $Sp^\bullet \cap Obs$ contient alors toutes les équations $c[\sigma(t)] = c[\sigma(t')]$ telles que $\sigma : V \to T_\Sigma$ et $c \in \mu\, Ctx$. On impose que le système sous test soit initial sur toutes les instances sans variables des prémisses des axiomes. Le système valide donc toutes les formules observables $c_i[\sigma(t_i)] = c_i[\sigma(t'_i)]$ où $\sigma : V \to T_\Sigma$ et $c_i \in \mu\, Ctx$. Pour autant, il peut ne pas valider une des prémisses $t_i = t'_i$, puisque la validation d'une formule au travers de tous ses contextes observables n'implique pas la validation de cette formule. Il est alors possible que le système sous test ne valide pas l'équation $t = t'$, alors même qu'il est correct, puisqu'il valide l'axiome $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t'$. Le jeu de tests $Sp^\bullet \cap Obs$ est biaisé, puisqu'il permet de rejeter un programme correct.

La propriété d'initialité seule ne suffit donc pas à garantir l'exhaustivité de $Sp^\bullet \cap Obs$. On impose alors que les spécifications soient positives, c'est-à-dire que les prémisses des axiomes ne fassent pas intervenir de sorte non observable.

DÉFINITION V.3 — *Spécification positive.*

Soit $Sp = (\Sigma, Ax)$ une spécification munie d'un ensemble de sortes observables $S_{obs} \subseteq S$. Sp est dite *positive* si et seulement si pour tout axiome $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t' \in Ax$, pour tout i, $1 \leq i \leq n$, $t_i = t'_i$ est une équation entre termes de sorte observable : $t_i, t'_i \in T_\Sigma(V)_s$ avec $s \in S_{obs}$.

Sous l'hypothèse d'initialité du système sur les prémisses des instances sans variables des axiomes et les conditions imposant aux spécifications d'être positives et de posséder une constante de chaque sorte, il est maintenant possible de prouver l'exhaustivité de $Sp^\bullet \cap Obs(S_{obs})$.

THÉORÈME V.1. *Soit $Sp = (\Sigma, Ax)$ une spécification positive munie d'un ensemble de sortes observables $S_{obs} \subseteq S$ et telle que pour chaque sorte $s \in S$, $T_{\Sigma_s} \neq \emptyset$. L'ensemble $Sp^\bullet \cap Obs(S_{obs})$ est un jeu de tests exhaustif pour tout système initial sur les instances sans variables des équations apparaissant dans les prémisses des axiomes de Sp.*

Preuve. Soit un programme $P \in Mod(\Sigma)$ initial sur les instances sans variables des prémisses de Ax. Supposons que $P \models Sp^\bullet \cap Obs(S_{obs})$. Montrons que $Correct_{Obs(S_{obs})}(P, Sp)$.

Soit $T_\Sigma/_{\sim_P}$ le quotient de T_Σ où $\sim_P$ est la congruence sur T_Σ définie dans la preuve de la proposition V.1. Soit $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \Rightarrow t = t'$ un axiome de Ax. Soit $\iota : V \to T_\Sigma/_{\sim_P}$ une interprétation telle que $T_\Sigma/_{\sim_P} \models_\iota t_i = t'_i$ pour tout i, $1 \leq i \leq n$. Par induction structurelle sur les termes de T_Σ, on peut montrer qu'il existe une substitution $\sigma : V \to T_\Sigma$ telle que $\iota = q_{\sim_P} \circ \sigma$ où $q_{\sim_P} : T_\Sigma \to T_\Sigma/_{\sim_P}$ est le morphisme quotient. Comme Sp est positive et comme $P \equiv_{Obs(S_{obs})} T_\Sigma/_{\sim_P}, T_\Sigma/_{\sim_P} \models \sigma(t_i) = \sigma(t'_i)$ implique que $P \models \sigma(t_i) = \sigma(t'_i)$. Comme P est initial sur les instances sans variables des prémisses des axiomes de Ax, $Sp \models \sigma(t_i) = \sigma(t'_i)$, donc $Sp \models \sigma(t) = \sigma(t')$. Par conséquent, $\sigma(t) = \sigma(t') \in Sp^\bullet \cap Obs(S_{obs})$, d'où $P \models \sigma(t) = \sigma(t')$. Ainsi $T_\Sigma/_{\sim_P} \models \sigma(t) = \sigma(t')$, et donc $T_\Sigma/_{\sim_P} \models_\iota t = t'$.

On suppose qu'il existe $\mathcal{M} \in Mod(Sp)$ tel que $\mathcal{M} \equiv_{Obs(S_{obs})} P$. Soit $t = t' \in Sp^\bullet \cap Obs(S_{obs})$. Par hypothèse, $\mathcal{M} \models t = t'$, donc $P \models t = t'$ également.

□

2 Sélection par dépliage des axiomes

Une fois posé un jeu de tests exhaustif, il est possible de commencer la phase de sélection d'un jeu de tests d'une taille raisonnable à soumettre au système. Parmi les méthodes qui ont été définies, la plus étudiée et certainement la plus efficace est la méthode appelée dépliage des axiomes. L'idée fondamentale est de profiter du cadre axiomatique dans lequel le processus de test a été formalisé pour utiliser les techniques de preuve fournies par ce formalisme, ces techniques étant bien connues, outillées et ayant montré leur efficacité.

Composer un jeu de tests à soumettre au système consiste à définir une méthode dans le but de découper le jeu de tests initial en sous-jeux de tests, puis en supposant l'hypothèse d'uniformité, à choisir un nombre fini de cas de test dans chacun des jeux de tests obtenus. Cette phase de choix des tests qui vont effectivement être soumis au système, appelée génération de tests, ne sera pas traitée dans cette thèse. Il s'agit plutôt ici de définir des critères de sélection pertinents dans le but de guider le choix final des tests, la génération nécessitant en soi une étude à part entière. L'application des critères de sélection choisis va permettre de raffiner l'ensemble initial des tests en caractérisant des sous-jeux de tests respectant certaines contraintes sur les données d'entrée. La définition de ces critères par dépliage des axiomes est fondée sur une analyse par cas de la spécification. Tester une opération donnée de la signature revient à tester son comportement sur chacun des cas décrits par la spécification. Il est alors possible de découper l'ensemble des données d'entrée de cette opération en autant de sous-ensembles qu'il y a de cas spécifiés.

Le jeu de tests initial $Sp^\bullet \cap Obs$ dont l'exhaustivité a été montrée à la section précédente est l'ensemble suivant :

$$T(Sp) = \{f(t_1, \ldots, t_n) = t \mid f \in F, t_1, \ldots, t_n, t \in T_\Sigma, Sp \models f(t_1, \ldots, t_n) = t\}$$

auquel sont ajoutées les tautologies entre les variables de V. On ne considérera pas ces équations triviales par la suite, celles-ci n'étant d'aucune utilité pour le test. Sans perte de généralité, on considère ici que toutes les sortes sont observables. Dans le cas général, il suffit de ne considérer dans $Sp^\bullet \cap Obs$ que les opérations de sorte observable, c'est-à-dire les opérations $f : s_1 \times \ldots \times s_n \to s$ de F telles que s soit observable. Diviser l'ensemble $T(Sp)$ revient alors à diviser chacun des jeux de tests associés à une opération donnée f de F. Un tel jeu de tests est appelé domaine de l'opération f.

DÉFINITION V.4 — *Domaine d'une opération.*

Soit $Sp = (\Sigma, Ax)$ une spécification conditionnelle positive. Soit $f : s_1 \times \ldots \times s_n \to s$ une opération de F. Le *domaine de* f, noté $T(Sp)_{|f}$, est l'ensemble défini par :

$$T(Sp)_{|f} = \{f(t_1, \ldots, t_n) = t \mid f(t_1, \ldots, t_n) = t \in T(Sp)\}$$

C'est ce jeu de tests, pour chaque opération de la signature, qui va être découpé en utilisant les axiomes spécifiant cette opération. Considérons la spécification suivante de l'insertion d'un entier dans une liste triée.

spec LISTENAT =

types $Nat : 0 \mid s(Nat)$

$Liste ::= [\,] \mid \_ :: \_(Nat, Liste)$

op $insrer : Nat \times Liste \to Liste$

vars $n, m : Nat, l : Liste$

- $ins\acute{e}rer(n, [\,]) = n :: [\,]$
- $(n \leq m) = true \Rightarrow ins\acute{e}rer(n, m :: l) = n :: m :: l$
- $(n \leq m) = false \Rightarrow ins\acute{e}rer(n, m :: l) = m :: ins\acute{e}rer(n, l)$

end

Le domaine de l'opération $ins\acute{e}rer$ par exemple, est l'ensemble

$$\{\sigma(ins\acute{e}rer(x, L)) = \sigma(L') \mid \sigma : V \to T_\Sigma, Sp \models \sigma(ins\acute{e}rer(x, L)) = \sigma(L')\}$$

La formule générique $ins\acute{e}rer(x, L) = L'$ dont les instances sans variables constituent les cas de test est appelée objectif de test. Le but de la sélection est de découper ce jeu de tests ou, ce qui revient au même, l'ensemble des substitutions closes $\sigma : V \to T_\Sigma$ associé à ce jeu de tests. En fait, toute substitution close de V dans T_Σ pouvant s'écrire comme la composition d'une substitution des variables de V dans $T_\Sigma(V)$ et d'une substitution close de V dans T_Σ étendue aux termes, le découpage sera effectué sur l'ensemble des substitutions des variables de V dans $T_\Sigma(V)$ de façon à ce que le processus puisse être itéré comme on le verra par la suite.

L'opération $ins\acute{e}rer$ est décrite par cas dans la spécification, selon que la liste est vide, ou que l'élément à insérer est inférieur ou égal ou supérieur au premier élément de liste. Tester cette opération revient alors à tester chacun de ces trois cas, ce qui donne les trois jeux de tests suivants :

- celui associé à la substitution $\sigma_0 : x \mapsto n_0,\ L \mapsto [\,],\ L' \mapsto n_0 :: [\,]$, c'est-à-dire le jeu de tests

$$\{ins\acute{e}rer(n_0, [\,]) = n_0 :: [\,]\}$$

- celui associé à la substitution $\sigma_1 : x \mapsto n_0,\ L \mapsto m_0 :: l_0,\ L' \mapsto n_0 :: m_0 :: l_0$ telle que $n_0 \leq m_0$, c'est-à-dire le jeu de tests

$$\{ins\acute{e}rer(n_0, m_0 :: l_0) = n_0 :: m_0 :: l_0 \mid (n_0 \leq m_0) = true\}$$

- celui associé à la substitution $\sigma_2 : x \mapsto n_0,\ L \mapsto m_0 :: l_0,\ L' \mapsto m_0 :: ins\acute{e}rer(n_0, l_0)$ telle que $n_0 > m_0$, c'est-à-dire le jeu de tests

$$\{ins\acute{e}rer(n_0, m_0 :: l_0) = m :: ins\acute{e}rer(n_0, l_0) \mid (n_0 \leq m_0) = false\}$$

Les trois jeux de tests obtenus par ce découpage sont caractérisés par une substitution des variables et un ensemble de contraintes : le premier est caractérisé par σ_0 et un ensemble de contraintes vide, le deuxième par σ_1 et la contrainte $(n_0 \leq m_0) = true$ et le troisième par σ_2 et $(n_0 \leq m_0) = false$. De façon générale, un jeu de tests pour une opération est défini de la façon suivante.

DÉFINITION V.5 — *Jeu de tests contraint pour une opération.*

Soit $Sp = (\Sigma, Ax)$ une spécification conditionnelle positive. Soient $f : s_1 \times \ldots \times s_n \to s$ une opération de F et $x_1, \ldots, x_n, y \in V$. Soient $\mathcal{C}$ un ensemble de Σ-équations appelé Σ-*contraintes* et $\sigma : V \to T_\Sigma(V)$ une substitution des variables.

Le *jeu de tests pour* f *contraint par* $\mathcal{C}$ *et* σ, noté $T_{(\mathcal{C},\sigma),f(x_1,\ldots,x_n)=y}$, est l'ensemble d'équations sans variables défini par :

$$T_{(\mathcal{C},\sigma),f(x_1,\ldots,x_n)=y} = \{\rho(\sigma(f(x_1,\ldots,x_n))) = \rho(\sigma(y)) \mid \rho : V \to T_\Sigma, \forall c \in \mathcal{C}, Sp \models \rho(c)\}$$

Le couple $((\mathcal{C},\sigma), f(x_1,\ldots,x_n) = y)$ est appelé *objectif de test contraint.*

Il est à noter que le domaine d'une opération f est le jeu de tests pour f contraint par le couple $(\{f(x_1,\ldots,x_n) = y\}, \mathrm{Id})$.

Les couples substitution ensemble de contraintes peuvent être déduits des arbres de preuve possibles dont la conclusion est une instance de l'objectif de test $f(x_1,\ldots,x_n) = y$. Pour poursuivre l'exemple de l'opération $insérer$, les jeux de tests contraints décrits plus haut peuvent être déduits des trois arbres de preuve possibles de conclusion $\sigma(insérer(x, L)) = \sigma(L')$ où σ est une substitution des variables. En effet, la formule $insérer(t_1, t_2) = t$ où t_1, t_2 et t sont des termes quelconques peut être la conclusion de trois arbres de preuve différents, résultant de l'application de l'un des trois axiomes. Si la formule a été prouvée à partir du premier axiome, cela signifie que x a été instancié par un élément n_0, L par la liste vide et L' par la liste composée de l'unique élément n_0.

$$\dfrac{\dfrac{}{insérer(n,[\,]) = n :: [\,]}\mathsf{Ax}}{insérer(n_0,[\,]) = n_0 :: [\,]}\mathsf{Subs}$$

La substitution σ_0 provient donc de l'unification de l'objectif de test avec l'axiome à partir duquel il a pu être prouvé. De la même façon, si l'instance de $insérer(x, L) = L'$ a été prouvée à partir du deuxième axiome, alors x a été instancié par un élément n_0, L par une liste dont la tête est un élément m_0 et la fin une liste quelconque l_0, et L' par la liste dont les deux premiers éléments sont n_0 et m_0 dans cet ordre et dont la fin est l_0.

$$\dfrac{\dfrac{\dfrac{}{(n \le m) = true \Rightarrow insérer(n, m :: l) = n :: m :: l}\mathsf{Ax}}{(n_0 \le m_0) = true \Rightarrow insérer(n_0, m_0 :: l_0) = n_0 :: m_0 :: l_0}\mathsf{Subs} \quad \dfrac{\vdots}{(n_0 \le m_0) = true}}{insérer(n_0, m_0 :: l_0) = n_0 :: m_0 :: l_0}\mathsf{MP}$$

Avec cette unification, la formule n'a pu être déduite du deuxième axiome que si $(n_0 \le m_0) = true$, c'est ce qui permet d'appliquer le modus ponens. Ainsi apparaissent les contraintes caractérisant les jeux de tests obtenus par ce découpage. Ce sont les lemmes intermédiaires qu'il faut montrer pour pouvoir déduire l'instance de l'objectif de test visée à partir de l'axiome choisi. La même analyse se répète sur le dernier cas, dans lequel la formule a été prouvée par l'application du troisième axiome.

$$\dfrac{\dfrac{\dfrac{}{(n \le m) = false \Rightarrow insérer(n, m :: l) = m :: insérer(n, l)}\mathsf{Ax}}{(n_0 \le m_0) = false \Rightarrow insérer(n_0, m_0 :: l_0) = m_0 :: insérer(n_0, l_0)}\mathsf{Subs} \quad \dfrac{\vdots}{(n_0 \le m_0) = false}}{insérer(n_0, m_0 :: l_0) = m_0 :: insérer(n_0, l_0)}\mathsf{MP}$$

La substitution σ_2 naît de l'unification de la formule et de l'axiome, et la contrainte du lemme $(n_0 \leq m_0) = \mathit{false}$.

Le dernier jeu de tests faisant intervenir l'opération $\mathit{insérer}$, il est possible de le découper de nouveau selon les trois axiomes et d'obtenir les jeux de tests suivants :

- le jeu de tests associé à la substitution $\sigma_{2,0} : x \mapsto n_0,\ L \mapsto m_0 :: [],\ L' \mapsto m_0 :: n_0 :: []$ telle que $n_0 > m_0$,
$$\{\mathit{insérer}(n_0, m_0 :: []) = m_0 :: n_0 :: [] \mid (n_0 \leq m_0) = \mathit{false}\}$$
- le jeu de tests associé à la substitution $\sigma_{2,1} : x \mapsto n_0,\ L \mapsto m_0 :: p_0 :: l_1,\ L' \mapsto m_0 :: n_0 :: p_0 :: l_1$ telle que $n_0 > m_0$ et $n_0 \leq p_0$,
$$\{\mathit{insérer}(n_0, m_0 :: p_0 :: l_1) = m_0 :: n_0 :: p_0 :: l_1 \mid (n_0 \leq m_0) = \mathit{false}, (n_0 \leq p_0) = \mathit{true}\}$$
- le jeu de tests associé à la substitution $\sigma_{2,2} : x \mapsto n_0,\ L \mapsto m_0 :: p_0 :: l_1,\ L' \mapsto m_0 :: p_0 :: \mathit{insérer}(n_0, l_1)$ telle que $n_0 > m_0$ et $n_0 > p_0$,
$$\{\mathit{insérer}(n_0, m_0 :: p_0 :: l_1) = m_0 :: p_0 :: \mathit{insérer}(n_0, l_1) \mid (n_0 \leq m_0) = \mathit{false}, (n_0 \leq p_0) = \mathit{false}\}$$

Ces trois jeux de tests peuvent de nouveau être déduits des arbres de preuve correspondants. La preuve d'une instance de $\mathit{insérer}(x, y :: L) = y :: L'$ découle par transitivité de la preuve d'une instance de $\mathit{insérer}(x, y :: L) = y :: \mathit{insérer}(x, L)$, c'est-à-dire du troisième arbre de preuve ci-dessus, et de la preuve d'une instance de $y :: \mathit{insérer}(x, L) = y :: L'$. Pour schématiser, on a l'arbre de preuve suivant :

$$\frac{\mathit{insérer}(x, y :: L) = y :: \mathit{insérer}(x, L) \quad y :: \mathit{insérer}(x, L) = y :: L'}{\mathit{insérer}(x, y :: L) = y :: L'}\ \mathsf{Trans}$$

De la preuve de la première prémisse, on a obtenu la substitution σ_2 et la contrainte $(n_0 \leq m_0) = \mathit{false}$. De la preuve de la seconde prémisse, c'est-à-dire de la deuxième étape de dépliage, on obtient de nouveau une substitution et un ensemble de contraintes. Le couple substitution contraintes caractérisant le jeu de tests obtenu est alors formé de la composition des deux substitutions et de l'union des deux ensembles de contraintes. Dans le cas où la deuxième prémisse est obtenue à partir du premier axiome, on a l'arbre de preuve suivant :

$$\dfrac{\dfrac{\vdots}{\mathit{insérer}(n_0, m_0 :: []) = m_0 :: \mathit{insérer}(n_0, [])} \quad \dfrac{\dfrac{}{m_0 = m_0}\,\mathsf{Réf} \quad \dfrac{\dfrac{}{\mathit{insérer}(n, []) = n :: []}\,\mathsf{Ax}}{\mathit{insérer}(n_0, []) = n_0 :: []}\,\mathsf{Subs}}{m_0 :: \mathit{insérer}(n_0, []) = m_0 :: n_0 :: []}\,\mathsf{Cong}}{\mathit{insérer}(n_0, m_0 :: []) = m_0 :: n_0 :: []}\,\mathsf{Trans}$$

La liste l_0 a été substituée par la liste vide $[]$, la composition de ces deux substitutions est donc la substitution $\sigma_{2,0}$. L'ensemble de contraintes obtenu par cette partie de la preuve est vide, l'ensemble résultant est donc le même que celui résultant de la première étape $\{(n_0 \leq m_0) = \mathit{false}\}$. Si l'instance de $y :: \mathit{insérer}(x, L) = y :: L'$ a été déduite du deuxième axiome spécifiant l'opération $\mathit{insérer}$, l'arbre de preuve est le suivant, où φ désigne la formule $\mathit{insérer}(n_0, m_0 :: []) = m_0 :: \mathit{insérer}(n_0, [])$.

$$\dfrac{\dfrac{\vdots}{\varphi}\quad \dfrac{\dfrac{}{m_0=m_0}\,\text{Réf}\quad \dfrac{}{(n\leq m)=true\Rightarrow ins\acute{e}rer(n,m::l)=n::m::l}\,\text{Ax}}{(n_0\leq p_0)=true\Rightarrow ins\acute{e}rer(n_0,p_0::l_1)=n_0::p_0::l_1}\,\text{Subs}}{(n_0\leq p_0)=true\Rightarrow m_0::ins\acute{e}rer(n_0,p_0::l_1)=m_0::n_0::p_0::l_1}\,\text{Cong}\quad \dfrac{\vdots}{(n_0\leq p_0)=true}}{m_0::ins\acute{e}rer(n_0,p_0::l_1)=m_0::n_0::p_0::l_1}\,\text{MP}$$
$$ins\acute{e}rer(n_0,m_0::p_0::l_1)=m_0::n_0::p_0::l_1\ \text{Trans}$$

La liste l_0 a cette fois été substituée par une liste non vide $p_0 :: l_1$, ce qui donne la substitution $\sigma_{2,1}$. Cette partie de la preuve nécessite un lemme intermédiaire $(n_0 \leq p_0) = true$, qui va être ajouté à l'ensemble de contraintes. L'ensemble de contraintes obtenu après ces deux étapes de dépliage est donc $\{(n_0 \leq m_0) = false, (n_0 \leq p_0) = true\}$. Enfin, si la formule a été prouvée à partir du troisième axiome, on obtient la substitution $\sigma_{2,2}$ et la contrainte supplémentaire $(n_0 \leq p_0) = false$, qui donne l'ensemble de contraintes $\{(n_0 \leq m_0) = false, (n_0 \leq p_0) = false\}$.

$$\dfrac{\dfrac{\vdots}{\varphi}\quad \dfrac{\dfrac{}{m_0=m_0}\,\text{Réf}\quad \dfrac{}{(n\leq m)=false\Rightarrow ins\acute{e}rer(n,m::l)=m::ins\acute{e}rer(n,l)}\,\text{Ax}}{(n_0\leq p_0)=false\Rightarrow ins\acute{e}rer(n_0,p_0::l_1)=p_0::ins\acute{e}rer(n_0,l_1)}\,\text{Subs}}{(n_0\leq p_0)=false\Rightarrow m_0::ins\acute{e}rer(n_0,p_0::l_1)=m_0::p_0::ins\acute{e}rer(n_0,l_1)}\,\text{Cong}\quad \dfrac{\vdots}{(n_0\leq p_0)=false}}{m_0::ins\acute{e}rer(n_0,p_0::l_1)=m_0::p_0::ins\acute{e}rer(n_0,l_1)}\,\text{MP}$$
$$ins\acute{e}rer(n_0,m_0::p_0::l_1)=m_0::p_0::ins\acute{e}rer(n_0,l_1)\ \text{Trans}$$

On constate alors que les arbres de preuves qui permettent de construire les jeux de tests en donnant les couples substitution ensemble de contraintes qui les caractérisent respectent une certaine structure. Aux feuilles se trouvent les instances d'axiomes et de la règle de réflexivité, sous lesquelles apparaissent les instances des règles de substitution et de congruence, puis les instances de la règle de modus ponens et enfin celles de la règle de transitivité. La procédure de dépliage va en fait consister en la recherche de telles preuves, guidée par la connaissance de cette structure qui permet ainsi de restreindre l'espace de recherche.

Le découpage du domaine de chaque opération par dépliage des axiomes peut être exprimé de manière algorithmique. La procédure de dépliage prend en entrée :

- une spécification conditionnelle positive $Sp = (\Sigma, Ax)$;
- une opération $f \in F$, donnant l'objectif de test $f(x_1, \ldots, x_n) = y$ où $x_i, y \in V$ pour tout i, $1 \leq i \leq n$, qui peut également être vu comme l'objectif de test contraint $((\{f(x_1, \ldots, x_n) = y\}, \mathrm{Id}), f(x_1, \ldots, x_n) = y)$;
- un ensemble Ψ de couples composés d'un ensemble de Σ-contraintes et d'une substitution des variables.

Le but est de découper le domaine de l'opération f de manière à obtenir un ensemble de jeux de tests contraints. Soumettre un jeu de tests contraint revient à choisir une substitution close ρ respectant les contraintes du jeu de tests et à soumettre l'objectif de test $f(x_1, \ldots, x_n) = y$ dont les variables auront été substituées par ρ, ainsi que les contraintes de $\mathcal{C}$, auxquelles aura également été appliquée ρ. L'ensemble initial Ψ_0 contient donc uniquement le couple contraintes substitution associé à l'objectif de test contraint initial :

$$\Psi_0 = \{(\{f(x_1, \ldots, x_n) = y\}, \mathrm{Id})\}$$

où $x_i, y \in V$ pour tout i, $1 \leq i \leq n$.

Intuitivement, le principe de la procédure est d'unifier une des contraintes c d'un des ensembles contenus dans Ψ avec la conclusion d'un axiome $\alpha_1 \wedge \ldots \wedge \alpha_m \Rightarrow \alpha$ de la spécification par une substitution σ et d'ajouter les prémisses de cet axiome à l'ensemble des contraintes. En effet, si la contrainte c a pu être unifiée avec cet axiome, c'est qu'il existe un arbre de preuve de conclusion $\sigma(c)$ (qui est égal à $\sigma(\alpha)$) dont une feuille est l'axiome $\alpha_1 \wedge \ldots \alpha_m \Rightarrow \alpha$ et qui contient les preuves des lemmes intermédiaires α_i pour tout i compris entre 1 et m.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha_1 \wedge \ldots \wedge \alpha_m \Rightarrow \alpha}\,\mathsf{Ax}}{\sigma(\alpha_1) \wedge \ldots \wedge \sigma(\alpha_m) \Rightarrow \sigma(\alpha)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\sigma(\alpha_1)}}{\sigma(\alpha_2) \wedge \ldots \wedge \sigma(\alpha_m) \Rightarrow \sigma(\alpha)}\,\mathsf{MP} \quad \dfrac{\vdots}{\sigma(\alpha_2)}}{\sigma(\alpha_3) \wedge \ldots \wedge \sigma(\alpha_m) \Rightarrow \sigma(\alpha)}\,\mathsf{MP}}{\vdots}}{\sigma(\alpha_m) \Rightarrow \sigma(\alpha)}\,\mathsf{MP} \quad \dfrac{\vdots}{\sigma(\alpha_m)}}{\sigma(\alpha)}\,\mathsf{MP}$$

La procédure de dépliage est alors exprimée par les deux règles d'inférence suivantes [1] :

$$\textbf{Réduction}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{t = t\}, \sigma')\}}{\Psi \cup \{(\mathcal{C}, \sigma')\}}$$

$$\textbf{Dépliage}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{t = r\}, \sigma')\}}{\Psi \cup \bigcup_{(c,\sigma) \in Dep(t=r) \cup Dep(r=t)} \{(\sigma(\mathcal{C}) \cup c, \sigma \circ \sigma')\}}$$

où $Dep(t = r)$, pour t syntaxiquement différent de r, est l'ensemble de couples défini par :

$$\left\{ (\{\sigma(t[v]_\omega) = \sigma(r), \sigma(\alpha_1), \ldots, \sigma(\alpha_m)\}, \sigma) \;\middle|\; \begin{array}{c} t_{|_\omega} = g(u_1, \ldots, u_n) \\ \sigma(t_{|_\omega}) = \sigma(g(v_1, \ldots, v_n)), \\ (\bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow g(v_1, \ldots, v_n) = v \in Ax \\ \text{ou} \\ \bigwedge_{1 \leq i \leq m} \alpha_i \Rightarrow v = g(v_1, \ldots, v_n) \in Ax) \end{array} \right\}$$

La règle de réduction permet de retirer les tautologies des ensembles de contraintes, qui ne sont d'aucune utilité pour le test puisqu'elles sont trivialement validées par tout système. La règle de dépliage proprement dit cherche à unifier un sous-terme d'un des deux termes de l'équation $t = r$ avec un terme apparaissant dans la conclusion d'un axiome de la spécification, un de ceux spécifiant l'opération de tête de ce sous-terme. Lorsque cette unification est possible, les prémisses de cet axiome sont ajoutées comme contraintes du jeu de tests, ainsi que l'instance de l'objectif de test obtenue par l'unification. En effet, si un sous-terme de t par

[1] Un terme pouvant être vu comme un arbre, on se reportera aux définitions des notions de position dans un arbre, de contexte et de composition données aux pages 183 et 184 de l'annexe B.

exemple a pu être unifié avec le membre gauche de la conclusion d'un axiome, alors il existe une preuve de $\sigma(t) = \sigma(r)$ de la forme suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\alpha_1 \wedge \ldots \wedge \alpha_m \Rightarrow g(v_1,\ldots,v_n) = v}\,\mathsf{Ax}}{\sigma(\alpha_1) \wedge \ldots \wedge \sigma(\alpha_m) \Rightarrow \sigma(g(v_1,\ldots,v_n)) = \sigma(v)}\,\mathsf{Subs}}{\vdots}}{\sigma(\alpha_1) \wedge \ldots \wedge \sigma(\alpha_m) \Rightarrow \sigma(t[g(v_1,\ldots,v_n)]_\omega) = \sigma(t[v]_\omega)}\,\mathsf{Cong} \quad \dfrac{\vdots}{\sigma(\alpha_1)}}{\sigma(\alpha_2) \wedge \ldots \wedge \sigma(\alpha_m) \Rightarrow \sigma(t[g(v_1,\ldots,v_n)]_\omega) = \sigma(t[v]_\omega)}\,\mathsf{MP}}{\vdots} \quad \dfrac{\vdots}{\sigma(\alpha_m)}}{\varphi \qquad \sigma(\alpha_m) \Rightarrow \sigma(t[g(v_1,\ldots,v_n)]_\omega) = \sigma(t[v]_\omega)}\,\mathsf{MP}}{\sigma(t[g(v_1,\ldots,v_n)]_\omega) = \sigma(t[v]_\omega)}\,\mathsf{Trans} \quad \dfrac{\vdots}{\psi}}{\sigma(t) = \sigma(r)}\,\mathsf{Trans}$$

où φ est la formule $\sigma(t) = \sigma(t[g(v_1,\ldots,v_n)]_\omega)$ et ψ est la formule $\sigma(t[v]_\omega) = \sigma(r)$. La substitution σ qui permet l'unification est alors composée avec la substitution σ' courante, et les lemmes intermédiaires permettant de prouver $\sigma(t) = \sigma(r)$ sont ajoutés aux contraintes.

Chaque unification avec un axiome génère un couple $(\mathcal{C}, \sigma)$, l'objectif de test initial est donc remplacé par autant d'ensembles de formules qu'il y a d'axiomes avec lesquels elle puisse être unifiée. La définition de $Dep(t = r)$ étant fondée sur la relation de sous-terme et l'unification, cet ensemble est calculable si la spécification a un nombre fini d'axiomes.

Pour une équation $t = r$, on a alors le critère de sélection $C_{t=r}$ défini par :

$$C_{t=r}(T_{(\mathcal{C},\sigma'),f(x_1,\ldots,x_n)=y}) = \begin{cases} \{T_{(\mathcal{C}\setminus\{t=r\}\cup c,\sigma\circ\sigma',f(x_1,\ldots,x_n)=y}\}_{(c,\sigma)\in Dep(t=r)} & \text{si } t = r \in \mathcal{C} \\ T_{(\mathcal{C},\sigma'),f(x_1,\ldots,x_n)=y} & \text{sinon} \end{cases}$$

qui associe à tout jeu de tests contraint le jeu de tests obtenu par l'application de la règle **Dépliage** s'il est possible de l'appliquer, et est l'identité sinon. Il s'agit alors de montrer que ce critère de sélection est pertinent, c'est-à-dire que son application conserve exactement tous les cas de test de $T_{(\mathcal{C},\sigma'),f(x_1,\ldots,x_n)=y}$.

On écrit $(\Psi, f(x_1,\ldots,x_n) = y) \vdash_D (\Psi', f(x_1,\ldots,x_n) = y)$ lorsque l'ensemble de couples Ψ' a pu être dérivé de Ψ par l'application de **Réduction** ou de **Dépliage**. La procédure de dépliage prend donc en entrée une spécification conditionnelle positive Sp et une opération non constructeur f et applique successivement les règles **Réduction** et **Dépliage** pour générer la suite

$$(\Psi_0, f(x_1,\ldots,x_n) = y) \vdash_D (\Psi_1, f(x_1,\ldots,x_n) = y) \vdash_D (\Psi_2, f(x_1,\ldots,x_n) = y) \vdash_D \ldots$$

On a vu que la procédure de dépliage consiste en une recherche de preuve d'instances sans variables de l'objectif de test, guidée par une certaine stratégie induite par la forme des arbres de preuve recherchés. Le principe de la procédure n'est alors pas de construire entièrement les arbres de preuve, mais seulement de pousser assez loin la recherche pour construire un découpage du jeu de tests exhaustif initial aussi fin que

voulu. La recherche peut en effet être arrêtée à tout moment, lorsque le « testeur » estime que la partition est suffisante en termes de nombre de cas de test pour les besoins de validation du système.

Les jeux de tests pour les opérations se prolongent naturellement aux ensembles de couples ensembles de contraintes substitution. Si Ψ est un ensemble de tels couples et si f est une opération de Σ

$$T_{\Psi,f(x_1,\ldots,x_n)=y} = \bigcup_{(\mathcal{C},\sigma)\in\Psi} T_{(\mathcal{C},\sigma),f(x_1,\ldots,x_n)=y}$$

Pour établir la complétude de la méthode de sélection, il est nécessaire de poser l'hypothèse que les variables utilisées pendant la procédure sont différentes de celles utilisées dans la spécification : pour tout ensemble de couples $\Psi = (\mathcal{C}, \sigma)$ résultant de la procédure,

$$\forall c \in \mathcal{C}, \forall \varphi \in Ax,\ Var(c) \cap Var(\varphi) = \emptyset$$

THÉORÈME V.2. [AABLM05] *Si* $(\Psi, f(x_1,\ldots,x_n) = y) \vdash_D (\Psi', f(x_1,\ldots,x_n) = y)$ *alors* $T_{\Psi,f(x_1,\ldots,x_n)=y)} = T_{\Psi',f(x_1,\ldots,x_n)=y)}$.

Idée de la preuve. Prouver la correction de la procédure revient à montrer qu'il existe une preuve d'une instance sans variables de $t = r$ à partir des contraintes de $Dep(t = r)$ et de l'axiome avec lequel il a été unifié. On a donné l'intuition plus haut qu'une telle preuve existait.

Prouver la complétude revient à montrer que tout arbre de preuve d'une instance sans variables de $t = r$ peut s'écrire sous la forme donnée plus haut. En fait, on remarque que la procédure de dépliage définit une stratégie de recherche d'arbres de preuve qui limite la recherche aux arbres tels que :

- aucune instance de la règle de transitivité n'apparaît au-dessus d'instances des règles de symétrie, de substitution et de congruence, ni au-dessus d'instances de la règle de modus ponens lorsque la transitivité apparaît dans la prémisse gauche du modus ponens ;
- aucune instance de la règle de modus ponens n'apparaît au-dessus d'instances de la règle de symétrie, de substitution, et de congruence ;
- aucune instance des règles de symétrie et de congruence n'apparaît au-dessus d'instances de la règle de substitution.

On doit alors prouver que la dérivabilité restreinte par cette stratégie coïncide avec la dérivabilité pleine. On définit alors des transformations élémentaires d'arbres de preuve afin de réécrire tout arbre de preuve en un arbre ayant la forme voulue et on montre que la transformation globale induite est fortement normalisante. Ce résultat est montré en utilisant un ordre récursif sur les chemins sur les arbres de preuve.

La preuve se trouve dans la version longue de l'article de Marc Aiguier, Agnès Arnould, Clément Boin, Pascale Le Gall et Bruno Marre [AABLM05].

□

Chapitre VI

Sélection à partir de spécifications du premier ordre sans quantificateurs

L'inconvénient de l'approche développée dans le chapitre précédent est de considérer des tests sous forme d'équations alors que les spécifications sont conditionnelles positives. En effet, dans un tel cadre, nous devons imposer trois conditions sur la spécification et le système sous test pour obtenir l'exhaustivité de l'ensemble des conséquences sémantiques observables de la spécification. Les deux contraintes sur la spécification, étant purement syntaxiques, peuvent être facilement vérifiées. Le problème réside dans le condition d'initialité. En effet, cette dernière est aussi difficile à démontrer que la correction du système elle-même. Elle demande une connaissance profonde du système sous test alors que celui-ci est testé en boîte noire. La technique de test proposée est alors limitée, puisqu'il est impossible d'assurer la validité et le non biais du jeu de tests partitionné. La condition d'initialité est rendue nécessaire par le fait que les cas de test ont une forme restreinte par rapport aux axiomes de la spécification. Ceci a été mis en lumière par le corollaire à proposition V.1 qui n'impose aucune condition sur le système pour montrer l'exhaustivité des conséquences sémantiques observables de la spécification lorsque celle-ci est purement équationnelle. Afin d'éliminer la condition d'initialité sur le système sous test, nous allons alors pousser plus en avant cette idée de considérer des cas de test de la même forme que les axiomes de la spécification. Dans ce chapitre, les cas de test ainsi que les axiomes des spécifications seront toutes formules du premier ordre sans quantificateurs, les cas de tests ne contenant bien entendu pas de variables.

Ce chapitre propose alors une généralisation de la méthode de sélection de tests qui a été présentée au chapitre précédent. La méthode par dépliage telle qu'elle a été définie s'appuie sur des spécifications équationnelles, dont les axiomes sont restreints à des formules conditionnelles positives. La procédure tire ainsi profit des propriétés de l'égalité en tant que congruence. Le fait de manipuler des équations permet par exemple de déplier un sous-terme d'un des deux termes d'une équation, grâce à la compatibilité de l'égalité avec les opérations. La procédure tire également profit de la forme de la règle de modus ponens qui permet de déduire une équation d'une formule conditionnelle positive dont auraient été prouvées toutes les prémisses. En fait, le principe même du dépliage repose sur cette règle d'inférence. C'est elle qui permet de découper le domaine d'entrée d'une opération selon les différents cas qui définissent son comportement dans la spécification. Cependant, le modus ponens ne permet de traiter que les formules conditionnelles

positives, ce qui restreint le champ d'application de la méthode de dépliage. L'idée de la généralisation naît alors du constat que la généralisation du modus ponens à n'importe quel type de formules est la règle de coupure du calcul des séquents de Gentzen.

Il s'agit donc dans ce chapitre d'étendre la méthode de sélection de cas de test par dépliage des axiomes aux spécifications du premier ordre sans quantificateurs. La généralisation porte sur deux aspects. D'une part, les formules atomiques ne sont plus réduites à des équations mais peuvent faire intervenir n'importe quel prédicat. D'autre part, les formules ne sont plus seulement conditionnelles positives mais sont construites librement à partir des opérateurs booléens. Considérer d'autres prédicats que l'égalité fait perdre l'avantage de manipuler une relation de congruence, ces nouveaux prédicats n'ayant *a priori* aucune propriété particulière. Cela permet de traiter tous les prédicats de façon similaire, plus aucune règle d'inférence ne caractérisera les propriétés d'un prédicat particulier. Nous avons vu que le fait de considérer des prédicats permet de spécifier plus simplement certains types de données. Enfin, la logique du premier ordre est la logique sous-jacente [1] au langage de spécification CASL [ABK+02, CoFI04]. Le travail présenté dans ce chapitre peut alors être vu comme une première proposition à la définition d'une technique de test fondée sur le dépliage d'axiomes dédié au langage CASL.

1 Normalisation des séquents

Les formules contenues dans les séquents manipulés par le calcul des séquents présenté à la figure II.1 peuvent être de toute forme. On peut cependant remarquer qu'il est possible de ramener n'importe quel séquent à une famille de séquents plus simples, c'est-à-dire ne faisant pas intervenir de connecteurs logiques. En effet, les règles associées à ces connecteurs sont inversibles, c'est-à-dire intuitivement qu'on peut les utiliser indifféremment de haut en bas ou de bas en haut.

PROPOSITION VI.1. *Soit* $\dfrac{\varphi_1 \dots \varphi_n}{\varphi} r$ *une règle d'inférence parmi l'ensemble des règles* $\{@-\mathsf{gauche}, @-\mathsf{droit}\}$ *où* $@ \in \{\neg, \wedge, \vee, \Rightarrow\}$. *Alors* $\varphi_1 \wedge \ldots \wedge \varphi_n \equiv \varphi$.

On dira que la règle r *est* inversible.

Il est alors possible de définir un processus qui permet de transformer tout séquent $\mid\!\sim \varphi$, où φ est une formule du premier ordre sans quantificateurs, en un ensemble de séquents $\Gamma \mid\!\sim \Delta$ où chaque formule de Γ et de Δ est atomique. On dira qu'un tel séquent est normalisé.

DÉFINITION VI.1 — *Séquent normalisé.*

Un *séquent normalisé* est un séquent $\Gamma \mid\!\sim \Delta$ tel que pour toute formule $\varphi \in \Gamma \cup \Delta$, φ est atomique.

La transformation d'un séquent $\mid\!\sim \varphi$ consiste alors à appliquer les règles relatives aux connecteurs logiques de bas en haut, de façon à éliminer au fur et à mesure chaque connecteur présent dans φ. Par exemple, si φ est la formule $(((\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_3) \Rightarrow \varphi_4) \vee (\varphi_2 \wedge \varphi_4)$, on a l'arbre de preuve suivant :

[1] Elle correspond plus exactement à sa partie totale et sans sous-sortes. Nous ne considérerons pas ici la possibilité de spécifier des fonctions partielles ou de déclarer une sorte construite au-dessus d'une autre.

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{\mathrel{|\!\sim} \varphi_1,\varphi_4,\varphi_2 \quad \mathrel{|\!\sim} \varphi_2,\varphi_4}{\mathrel{|\!\sim} \varphi_1\wedge\varphi_2,\varphi_4,\varphi_2}\wedge\text{-droit}
      \quad \varphi_3 \mathrel{|\!\sim} \varphi_4,\varphi_2
    }{(\varphi_1\wedge\varphi_2)\Rightarrow\varphi_3 \mathrel{|\!\sim} \varphi_4,\varphi_2}\Rightarrow\text{-gauche}
  }{\mathrel{|\!\sim} ((\varphi_1\wedge\varphi_2)\Rightarrow\varphi_3)\Rightarrow\varphi_4,\varphi_2}\Rightarrow\text{-droit}
  \quad
  \dfrac{
    \dfrac{
      \dfrac{\mathrel{|\!\sim} \varphi_1,\varphi_4 \quad \mathrel{|\!\sim} \varphi_2,\varphi_4}{\mathrel{|\!\sim} \varphi_1\wedge\varphi_2,\varphi_4}\wedge\text{-droit}
      \quad \varphi_3 \mathrel{|\!\sim} \varphi_4
    }{(\varphi_1\wedge\varphi_2)\Rightarrow\varphi_3 \mathrel{|\!\sim} \varphi_4}\Rightarrow\text{-gauche}
  }{\mathrel{|\!\sim} ((\varphi_1\wedge\varphi_2)\Rightarrow\varphi_3)\Rightarrow\varphi_4,\varphi_4}\Rightarrow\text{-droit}
}{
  \dfrac{\mathrel{|\!\sim} ((\varphi_1\wedge\varphi_2)\Rightarrow\varphi_3)\Rightarrow\varphi_4,\varphi_2\wedge\varphi_4}{\mathrel{|\!\sim} (((\varphi_1\wedge\varphi_2)\Rightarrow\varphi_3)\Rightarrow\varphi_4)\vee(\varphi_2\wedge\varphi_4)}\vee\text{-droit}
}\wedge\text{-droit}
$$

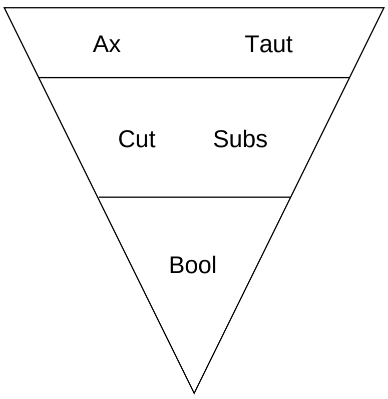
La famille de séquents normalisés correspondant au séquent

$$\mathrel{|\!\sim} (((\varphi_1 \wedge \varphi_2) \Rightarrow \varphi_3) \Rightarrow \varphi_4) \vee (\varphi_2 \wedge \varphi_4)$$

est donc

$$\{ \mathrel{|\!\sim} \varphi_1, \varphi_4, \varphi_2, \;\; \mathrel{|\!\sim} \varphi_2, \varphi_4, \;\; \varphi_3 \mathrel{|\!\sim} \varphi_4, \varphi_2, \;\; \mathrel{|\!\sim} \varphi_1, \varphi_4, \;\; \mathrel{|\!\sim} \varphi_2, \varphi_4, \;\; \varphi_3 \mathrel{|\!\sim} \varphi_4\}$$

Cette transformation n'a d'intérêt que si elle permet de simplifier la procédure de dépliage. Cette procédure consistant en la recherche d'un arbre de preuve pour un objectif de test donné, il va falloir s'assurer que le fait de ne considérer que des séquents normalisés ne réduit pas les capacités du calcul, c'est-à-dire que chercher la preuve d'une formule quelconque revient au même que chercher les preuves des séquents normalisés qui lui sont équivalents. Il faut pour cela montrer que la preuve d'une formule quelconque peut se décomposer en une preuve ne faisant intervenir que des séquents normalisés et une preuve ne contenant que des instances de règles associées aux connecteurs booléens. Il suffit en fait de montrer que toutes les instances de règles associées à ces connecteurs peuvent « descendre » dans l'arbre, c'est-à-dire que tout arbre de preuve peut être transformé en un arbre de même conclusion dans lequel les instances de règles associées aux connecteurs booléens se trouvent toutes sous les instances de règles de la substitution et de la coupure. Un tel arbre aurait donc la forme représentée par la figure suivante, où Bool représente l'ensemble des instances de règles associées aux connecteurs booléens : $\mathsf{Bool} = \{@-\mathsf{gauche}, @-\mathsf{droit} \mid @ \in \{\neg, \wedge, \vee, \Rightarrow\}\}$.



Il faut pour cela poser l'hypothèse que les axiomes de la spécification introduits par la règle Ax sont sous la forme de séquents normalisés, et de la même façon, que les formules introduites par la règle Taut sont atomiques. Les séquents manipulés en haut de l'arbre seront alors tous normalisés, tandis que le bas de l'arbre consistera uniquement en l'application de règles d'introduction des connecteurs booléens.

Cette transformation s'opère à l'aide de règles de transformation d'arbres de preuves élémentaires (voir Annexe B). Ces règles sont de deux sortes. La première concerne les règles qui permettent de faire remonter la règle de substitution sur une des règles associées à un connecteur booléen, par exemple ici la règle d'introduction de la négation dans le membre droit d'un séquent :

$$\dfrac{\dfrac{\Gamma, \varphi \mathrel{|\!\sim} \Delta}{\Gamma \mathrel{|\!\sim} \Delta, \neg\varphi}\,\neg\text{-droit}}{\sigma(\Gamma) \mathrel{|\!\sim} \sigma(\Delta), \neg\sigma(\varphi)}\,\mathsf{Subs} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, \varphi \mathrel{|\!\sim} \Delta}{\sigma(\Gamma), \sigma(\varphi) \mathrel{|\!\sim} \sigma(\Delta)}\,\mathsf{Subs}}{\sigma(\Gamma) \mathrel{|\!\sim} \sigma(\Delta), \neg\sigma(\varphi)}\,\neg\text{-droit}$$

Le deuxième groupe de règles contient celles qui permettent de faire remonter la coupure sur les règles booléennes. Deux cas sont à considérer : celui où la formule coupée n'est pas celle qui vient d'être modifiée par une règle booléenne, et celui où la formule coupée est celle qui vient d'être modifiée. Dans le premier cas, la transformation est simple :

$$\dfrac{\dfrac{\Gamma \mathrel{|\!\sim} \Delta, \psi, \varphi}{\Gamma, \neg\varphi \mathrel{|\!\sim} \Delta, \psi}\,\neg\text{-gauche} \qquad \Gamma', \psi \mathrel{|\!\sim} \Delta'}{\Gamma, \Gamma', \neg\varphi \mathrel{|\!\sim} \Delta, \Delta'}\,\mathsf{Cut} \quad \rightsquigarrow \quad \dfrac{\Gamma \mathrel{|\!\sim} \Delta, \psi, \varphi \qquad \Gamma', \psi \mathrel{|\!\sim} \Delta'}{\dfrac{\Gamma, \Gamma' \mathrel{|\!\sim} \Delta, \Delta', \varphi}{\Gamma, \Gamma', \neg\varphi \mathrel{|\!\sim} \Delta, \Delta'}\,\neg\text{-gauche}}\,\mathsf{Cut}$$

Dans le deuxième cas, il est nécessaire de considérer les hypothèses qui ont été posées plus haut, concernant la forme des séquents introduits par les schémas d'axiomes, pour qu'il soit possible de transformer ce type d'arbres de preuve. Ces hypothèses étant posées, il est impossible que la formule coupée, ici $\neg\varphi$, provienne de l'introduction d'un axiome ou d'une tautologie, elle est donc nécessairement la conclusion d'une instance de la règle d'introduction de la négation à droite, ce qui autorise la transformation suivante :

$$\dfrac{\dfrac{\Gamma, \varphi \mathrel{|\!\sim} \Delta}{\Gamma \mathrel{|\!\sim} \Delta, \neg\varphi}\,\neg\text{-gauche} \qquad \dfrac{\Gamma' \mathrel{|\!\sim} \Delta', \varphi}{\Gamma', \neg\varphi \mathrel{|\!\sim} \Delta'}\,\neg\text{-droit}}{\Gamma, \Gamma' \mathrel{|\!\sim} \Delta, \Delta'}\,\mathsf{Cut} \quad \rightsquigarrow \quad \dfrac{\Gamma, \varphi \mathrel{|\!\sim} \Delta \qquad \Gamma' \mathrel{|\!\sim} \Delta', \varphi}{\Gamma, \Gamma' \mathrel{|\!\sim} \Delta, \Delta'}\,\mathsf{Cut}$$

Avec l'ordre suivant sur les instances des règles d'inférence

$$\mathsf{Ax} \sim \mathsf{Taut} \succ \mathsf{Cut} \sim \mathsf{Subs} \succ \mathsf{Bool}$$

ces règles respectent les termes de la condition 1 (voir l'annexe B), la transformation globale qu'elles induisent est donc fortement normalisante. Tout arbre de preuve peut donc être transformé en un arbre dans lequel sont manipulés uniquement des séquents normalisés puis sont appliquées les règles associées aux opérateurs booléens, sous les conditions précédemment énoncées. Ne seront alors considérés dans la suite de ce chapitre que des séquents normalisés.

EXEMPLE VI.1 — *Spécification des listes triées de rationnels.*

La spécification fil rouge de ce chapitre sera celle des listes triées de rationnels. Cette spécification fait intervenir trois sortes, les entiers naturels, les rationnels et les listes de rationnels. Les entiers naturels sont construits à partir de la constante nulle 0 et de l'opération successeur s qui à un entier donné associe l'entier qui le suit. L'addition add et la multiplication $mult$ d'entiers sont définies de la manière usuelle, tout comme

le prédicat inférieur strict *infn*. Les rationnels sont définis comme les fractions d'entiers n/m. L'égalité *egr* de deux rationnels x/y et u/v est définie comme étant l'égalité entre les entiers $x \times v$ et $u \times y$. Les rationnels considérés étant positifs, un rationnel x/y est inférieur à un rationnel u/v (prédicat *infr*) si $x \times v$ est inférieur à $u \times y$.

Les listes sont construites à partir de la liste vide [] et de l'opération de concaténation en tête de liste notée _ :: _. L'insertion *insérer* d'un rationnel dans une liste triée est spécifiée selon quatre cas : la liste est vide ; le rationnel à insérer est égal au premier élément de la liste, auquel cas cet élément n'est pas répété ; le rationnel à insérer est inférieur au premier élément de la liste, il est alors inséré à la tête de la liste ; l'élément à insérer est supérieur au premier élément de la liste, il est alors inséré dans la suite de la liste. Le prédicat d'appartenance *estdans* d'un rationnel à une liste est défini par deux axiomes : aucun élément n'appartient à la liste vide ; un rationnel appartient à une liste non vide s'il est égal au premier élément de la liste ou s'il se trouve dans le reste de la liste.

Le comportement des opérations *add*, *mult* et *insérer* est généralement spécifié par des équations. Dans le cadre de la logique du premier ordre, cela demande d'introduire trois prédicats $=_{Nat}$: $Nat \times Nat$, $=_{Rat}$: $Rat \times Rat$ et $=_{Liste}$: $Liste \times Liste$, chacun spécifié par les axiomes suivants :

$$
\begin{aligned}
& x =_{@} x \\
& x =_{@} y \Rightarrow y =_{@} x \\
& x =_{@} y \wedge y =_{@} z \Rightarrow x =_{@} z \\
& x_1 =_{@_1} y_1 \wedge \ldots \wedge x_n =_{@_n} y_n \Rightarrow f(x_1, \ldots, x_n) =_{@} f(y_1, \ldots, y_n) \\
& x_1 =_{@_1} y_1 \wedge \ldots \wedge x_n =_{@_n} y_n \wedge p(x_1, \ldots, x_n) \Rightarrow p(y_1, \ldots, y_n)
\end{aligned}
$$

où $@, @_i \in \{Nat, Rat, Liste\}$, $f : @_1 \times \ldots \times @_n \to @$ et $p : @_1 \times \ldots \times @_n$. De façon à ce que les spécifications manipulées ne soient pas trop lourdes, l'approche suivie ici sera celle qui consiste à transformer chaque opération $f : s_1 \times \ldots \times s_n \to s$ en un prédicat $f : s_1 \times \ldots \times s_n \times s$, l'égalité devenant ainsi implicite. Cette approche permet également d'alléger la procédure de dépliage.

spec LISTERAT =

types $Nat ::= 0 \mid s(Nat)$
$Rat ::= \_/\_(Nat, Nat)$
$Liste ::= [] \mid \_ :: \_(Rat, Liste)$

preds $add : Nat \times Nat \times Nat$
$mult : Nat \times Nat \times Nat$
$infn : Nat \times Nat$
$egr : Rat \times Rat$
$infr : Rat \times Rat$
$insérer : Rat \times Liste \times Liste$
$estdans : Rat \times Liste$

vars $x, y, z, u, v, n, m : Nat$; $e : Rat$; $l, l' : Liste$

- $add(x, 0, x)$

- $add(x, s(y), s(z)) \Leftrightarrow add(x, y, z)$
- $mult(x, 0, 0)$
- $add(x, u, z) \wedge mult(x, y, u) \Rightarrow mult(x, s(y), z)$
- $infn(0, s(x))$
- $\neg\, infn(x, 0)$
- $infn(s(x), s(y)) \Leftrightarrow infn(x, y)$
- $mult(x, s(v), n) \wedge mult(u, s(y), n) \Rightarrow egr(x/s(y), u/s(v))$
- $infn(m, n) \wedge mult(x, s(v), m) \wedge mult(u, s(y), n) \Rightarrow infr(x/s(y), u/s(v))$
- $insérer(x/s(y), [\,], x/s(y) :: [\,])$
- $egr(x/s(y), e) \Rightarrow insérer(x/s(y), e :: l, e :: l)$
- $infr(x/s(y), e) \Rightarrow insérer(x/s(y), e :: l, x/s(y) :: e :: l)$
- $infr(e, x/s(y)) \wedge insérer(x/s(y), l, l') \Rightarrow insérer(x/s(y), e :: l, e :: l')$
- $\neg\, estdans(x/s(y), [\,])$
- $estdans(x/s(y), e :: l) \Leftrightarrow egr(x/s(y), e) \vee estdans(x/s(y), l)$

end

Les axiomes de la spécification sont alors transformés en séquents normalisés, pour les raisons indiquées précédemment. Cette transformation est aisément automatisable.

1. $\mid\!\sim\ add(x, 0, x)$
2. $add(x, s(y), s(z)) \mid\!\sim\ add(x, y, z)$
3. $add(x, y, z) \mid\!\sim\ add(x, s(y), s(z))$
4. $\mid\!\sim\ mult(x, 0, 0)$
5. $add(x, u, z), mult(x, y, u) \mid\!\sim\ mult(x, s(y), z)$
6. $\mid\!\sim\ infn(0, s(x))$
7. $infn(x, 0) \mid\!\sim$
8. $infn(s(x), s(y)) \mid\!\sim\ infn(x, y)$
9. $infn(x, y) \mid\!\sim\ infn(s(x), s(y))$
10. $mult(x, s(v), n), mult(u, s(y), n) \mid\!\sim\ egr(x/s(y), u/s(v))$
11. $infn(m, n), mult(x, s(v), m), mult(u, s(y), n) \mid\!\sim\ infr(x/s(y), u/s(v))$
12. $\mid\!\sim\ insérer(x/s(y), [\,], x/s(y) :: [\,])$
13. $egr(x/s(y), e) \mid\!\sim\ insérer(x/s(y), e :: l, e :: l)$
14. $infr(x/s(y), e) \mid\!\sim\ insérer(x/s(y), e :: l, x/s(y) :: e :: l)$
15. $infr(e, x/s(y)), insérer(x/s(y), l, l') \mid\!\sim\ insérer(x/s(y), e :: l, e :: l')$
16. $estdans(x/s(y), [\,]) \mid\!\sim$
17. $estdans(x/s(y), e :: l) \mid\!\sim\ egr(x/s(y), e), estdans(x/s(y), l)$
18. $estdans(x/s(y), l) \mid\!\sim\ estdans(x/s(y), e :: l)$
19. $egr(x/s(y), e) \mid\!\sim\ estdans(x/s(y), e :: l)$

Ce sont ces axiomes qui seront considérés par la suite. ◇

2 Exhaustivité

Maintenant posé le formalisme dans lequel la généralisation est possible, il faut étudier les conditions de cette généralisation. La première étape est de définir le jeu de tests de référence à partir duquel la sélection peut être effectuée. Comme il a déjà été dit, le plus grand ensemble de formules qu'un système doit valider et qu'il suffit qu'il valide est l'ensemble des conséquences sémantiques observables de la spécification, l'ensemble $Sp^\bullet \cap Obs$. Il s'agit tout d'abord de définir, dans le cadre qui a été posé, quelles sont les formules observables.

Ici, l'ensemble Obs des formules observables est l'ensemble de toutes les formules sans variables, sans restriction. On suppose en effet que toute formule, quelle que soit sa forme, peut être soumise au système. Le jeu de tests dont il faut montrer l'exhaustivité est donc l'ensemble des instances sans variables des conséquences sémantiques de la spécification. Il a été montré au chapitre précédent que le jeu de tests $Sp^\bullet \cap Obs$ est exhaustif lorsque Sp est une spécification conditionnelle positive et Obs est l'ensemble des équations sans variables, pour tout système admettant la condition d'initialité sur les instances sans variables des équations apparaissant dans les prémisses des axiomes de la spécification. Ici, ne manipulant pas des équations mais des prédicats introduits dans la signature, nous supposons que tous ont un équivalent dans le système sous test, selon l'hypothèse que nous avons formulée dans le chapitre IV. Ainsi, la spécification n'est soumise à aucune restriction, exceptée celle de posséder une constante de chaque sorte. Nous montrons enfin, comme nous le souhaitions, que le système sous test n'a pas de condition particulière à respecter pour que $Sp^\bullet \cap Obs$ soit exhaustif pour celui-ci.

THÉORÈME VI.1. *Soient $Sp = (\Sigma, Ax)$ une spécification du premier ordre sans quantificateurs. L'ensemble $Sp^\bullet \cap Obs$ des conséquences sémantiques observables de la spécification est exhaustif pour $Mod(\Sigma)$.*

Pour les besoins de la preuve, on introduit la notion suivante de modèle de Herbrand d'un ensemble de formules.

DÉFINITION VI.2 — *Modèle de Herbrand.*

Soit Σ une signature. Soit $\Psi \subseteq For(\Sigma)$ un ensemble de formules sur Σ. On appelle *modèle de Herbrand de* Ψ, noté $\mathcal{H}_{T_\Sigma}$, le Σ-modèle

- défini comme étant l'ensemble T_Σ, muni pour chaque opération $f : s_1 \times \ldots \times s_n \to s \in F$ de l'application $f^{\mathcal{H}_{T_\Sigma}} : T_{\Sigma_{s_1}} \times \ldots \times T_{\Sigma_{s_n}} \to T_{\Sigma_s}$ qui à tout n-uplet $(t_1, \ldots, t_n)$ associe le terme $f(t_1, \ldots, t_n)$;
- caractérisé par l'ensemble de formules sans variables

$$\{r(t_1, \ldots, t_n) \mid r : s_1 \times \ldots \times s_n \in R, (t_1, \ldots, t_n) \in T_{\Sigma_{s_1}} \times \ldots \times T_{\Sigma_{s_n}}, \Psi \models r(t_1, \ldots, t_n)\}$$

Le modèle de Herbrand de Ψ est alors un modèle de Ψ, comme l'énonce le théorème suivant.

THÉORÈME VI.2. (Herbrand [Her68]) *Pour toute formule sans variables $\varphi \in For(\Sigma)$,*

$$\Psi \models \varphi \Leftrightarrow \mathcal{H}_{T_\Sigma} \models \varphi$$

On peut maintenant procéder à la preuve du théorème VI.1.

Preuve. Soit $\mathcal{S} \in Mod(\Sigma)$ un système tel que $\mathcal{S} \models Sp^\bullet \cap Obs$. Montrons que $Correct_{obs}(\mathcal{S}, Sp)$. On note $Th(\mathcal{S}) = \{\varphi \in Obs \mid \mathcal{S} \models \varphi\}$. Soit $\mathcal{H}_{T_\Sigma} \in Mod(\Sigma)$ le modèle de Herbrand de $Th(\mathcal{S})$. Par définition, le système et ce modèle sont élémentairement équivalents sur $Th(\mathcal{S})$, donc on a $\mathcal{S} \equiv_{obs} \mathcal{H}_{T_\Sigma}$. Montrons que ce modèle de Herbrand est un modèle de la spécification, c'est-à-dire que $\mathcal{H}_{T_\Sigma} \in Mod(Sp)$. Soit φ un axiome de Sp. Soit $\nu : V \to \mathcal{H}_{T_\Sigma}$ une interprétation. Par définition du modèle de Herbrand, $\nu(\varphi)$ est une formule sans variables. Donc par hypothèse, $\mathcal{S} \models \nu(\varphi)$, et comme $\mathcal{H}_{T_\Sigma}$ est élémentairement équivalent à $\mathcal{S}$ sur Obs, on a $\mathcal{H}_{T_\Sigma} \models \nu(\varphi)$. On a alors $\mathcal{H}_{T_\Sigma} \models_\nu \varphi$, et $\mathcal{H}_{T_\Sigma}$ est bien un modèle de Sp. Il existe donc un modèle de Sp élémentairement équivalent sur Obs à $\mathcal{S}$.

On suppose maintenant que $Correct_{obs}(\mathcal{S}, Sp)$ et on montre que $\mathcal{S} \models Sp^\bullet \cap Obs$. Si le système est correct par rapport à sa spécification via Obs, alors il existe $\mathcal{M} \in Mod(Sp)$ tel que $\mathcal{M} \equiv_{obs} \mathcal{S}$. Soit $\varphi \in Sp^\bullet \cap Obs$. Comme $\mathcal{M}$ est un modèle de Sp, alors $\mathcal{M} \models \varphi$, donc $\mathcal{S} \models \varphi$ également.

$Sp^\bullet \cap Obs$ est donc un jeu de tests exhaustif pour tout $\mathcal{S} \in Mod(\Sigma)$. □

Il serait donc possible de montrer la correction de tout système si le jeu de tests $Sp^\bullet \cap Obs$ pouvait être soumis dans son intégralité. C'est donc à partir de ce jeu de tests que va s'effectuer la sélection d'un jeu de tests effectif à soumettre au système.

3 Dépliage

Il s'agit ici de généraliser la procédure de sélection par dépliage des axiomes, précédemment définie pour des spécifications conditionnelles positives et pour des objectifs de test sous la forme d'équations. La méthode consiste, comme dans le cadre conditionnel positif, à découper un jeu de tests initial, le jeu de tests exhaustif, en sous-jeux de tests à l'aide de critères de sélection fondés sur les axiomes de la spécification.

Le jeu de tests à découper est l'ensemble $Sp^\bullet \cap Obs$ des conséquences sémantiques sans variables de la spécification qui a été montré exhaustif pour tout système à la section précédente. Diviser cet ensemble revient à diviser chacun des ensembles associé à une formule particulière, à un objectif de test particulier. Le jeu de tests associé à une opération dans le cadre conditionnel positif devient alors ici plus généralement un jeu de tests associé à une formule.

DÉFINITION VI.3 — *Jeu de tests associé à une formule.*

Soient $Sp = (\Sigma, Ax)$ une spécification du premier ordre sans quantificateurs. Soit φ une formule du premier ordre sans quantificateurs, appelée *objectif de test*. Le *jeu de tests associé à* φ, noté T_φ, est l'ensemble défini par :

$$T_\varphi = \{\rho(\varphi) \mid \rho : V \to T_\Sigma, \rho(\varphi) \in Sp^\bullet \cap Obs\}$$

Le jeu de tests associé à une formule est donc l'ensemble des instances sans variables de cette formule. Le domaine d'une opération f tel qu'il a été défini dans le cadre conditionnel positif est en fait le jeu de tests associé à la formule $f(x_1, \ldots, x_n) = y$. Toutes les formules du jeu de tests exhaustif sont ainsi couvertes puisque ce sont des équations, et puisque toute équation dans ce cadre peut s'écrire sous cette forme. Ici, comme aucune hypothèse n'est faite sur la spécification ni sur la forme des formules à soumettre au système, un objectif de test peut être n'importe quelle formule. On remarque même que la formule φ prise comme objectif de test peut ne pas être une conséquence sémantique de la spécification. C'est lors du choix de la

substitution close ρ que sera construite une conséquence sémantique de Sp de la forme $\rho(\varphi)$. On verra plus loin la manière dont sera gérée la possibilité de considérer toute formule comme un objectif de test.

Dans le cadre conditionnel positif, la procédure de dépliage permet de diviser le jeu de tests associé à une opération en sous-jeux de tests caractérisés par un couple formé d'une substitution des variables et d'un ensemble de contraintes. De la même manière, ici, le jeu de tests associé à une formule va être découpé en sous-jeux de tests contraints, définis de la manière suivante.

DÉFINITION VI.4 — *Jeu de tests contraint.*

Soient $Sp = (\Sigma, Ax)$ une spécification du premier ordre sans quantificateurs. Soit φ une formule du premier ordre sans quantificateurs. Soient $\mathcal{C} \subseteq For(\Sigma)$ un ensemble de formules appelées Σ-*contraintes* et $\sigma : V \to T_\Sigma(V)$ une substitution des variables.

Le *jeu de tests associé à φ contraint par $\mathcal{C}$ et σ*, noté $T_{(\mathcal{C},\sigma),\varphi}$, est l'ensemble de formules sans variables définir par :

$$T_{(\mathcal{C},\sigma),\varphi} = \{\rho(\sigma(\varphi)) \mid \rho : V \to T_\Sigma, \forall c \in \mathcal{C}, Sp \models \rho(c)\}$$

Le couple $((\mathcal{C}, \sigma), \varphi)$ est appelé *objectif de test contraint*.

Un jeu de tests contraint associé à une opération dans le cadre conditionnel positif est donc en fait un jeu de tests contraint, au sens de cette définition, associé à la formule $f(x_1, \ldots, x_n) = y$. On remarque que l'objectif de test φ de la définition précédente peut être vu comme l'objectif de test contraint $((\{\varphi\}, \mathrm{Id}), \varphi)$.

Dans la pratique, l'objectif de test initial n'est pas contraint. Le but de la procédure de dépliage est de le remplacer par un ensemble d'objectifs de test contraints, à l'aide des axiomes de la spécification.

La procédure prend alors en entrée :

- une spécification du premier ordre sans quantificateurs $Sp = (\Sigma, Ax)$ dont les axiomes ont été mis sous la forme de séquents normalisés ;
- une formule du premier ordre sans quantificateurs φ transformée en séquent normalisé, et vue comme l'objectif de test contraint initial $((\{\varphi\}, \mathrm{Id}), \varphi)$;
- une famille Ψ de couples $(\mathcal{C}, \sigma)$ où $\mathcal{C}$ est un ensemble de Σ-contraintes sous la forme de séquents normalisés, et σ est une substitution des variables $V \to T_\Sigma(V)$.

Le premier ensemble Ψ_0 correspond aux contraintes initiales de l'objectif de test, et contient donc uniquement le couple composé de l'ensemble de séquents normalisés obtenu à partir de l'objectif de test φ et la substitution identité.

Les jeux de tests pour des formules du premier ordre sans quantificateurs sont naturellement prolongés aux ensembles de couples Ψ de la façon suivante :

$$T_{\Psi,\varphi} = \bigcup_{(\mathcal{C},\sigma)\in\Psi} T_{(\mathcal{C},\sigma),\varphi}$$

Le principe de la procédure est de chercher les arbres de preuve qui permettent de déduire l'objectif de test initial des axiomes de la spécification. Dans le cadre conditionnel positif, cette recherche était effectuée en cherchant à unifier l'objectif de test, sous la forme d'une équation, à la conclusion d'un axiome de la spécification. Lorsque l'unification était possible, les prémisses de cet axiome étaient alors ajoutées à l'ensemble de contraintes. La règle de modus ponens permettait en effet de déduire l'objectif de test de

l'axiome avec la conclusion duquel il avait pu être unifié, en éliminant progressivement toutes les prémisses de cet axiome.

Ici, c'est la règle de coupure qui va être utilisée à la place du modus ponens. L'objectif de test initial n'est plus une formule atomique mais peut être n'importe quelle formule. La procédure va alors consister à chercher une preuve de cette formule à partir d'un des axiomes de la spécification. On va donc chercher à unifier cette formule avec un axiome, plus exactement, on va chercher à unifier une partie des sous-formules de cette formule avec une partie des sous-formules d'un axiome. Si l'objectif de test considéré est un séquent normalisé de la forme $\gamma_1, \ldots, \gamma_p, \ldots, \gamma_m \mid\sim \delta_1, \ldots, \delta_q, \ldots, \delta_n$, on va chercher à unifier un sous-ensemble de $\{\gamma_1, \ldots, \gamma_m, \delta_1, \ldots, \delta_n\}$ avec un sous-ensemble des formules constituant un axiome. On cherche alors un axiome de la forme $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mid\sim \varphi_1, \ldots, \varphi_q, \zeta_1, \ldots, \zeta_k$ tel qu'on puisse unifier ψ_i et γ_i pour tout i entre 1 et p, et φ_i et δ_i pour tout i entre 1 et q.

Il reste à s'assurer qu'il est bien possible de prouver l'objectif de test à partir de l'axiome. C'est ici qu'intervient la règle de coupure. Elle va permettre à la fois d'éliminer les sous-formules de l'axiome qui n'ont pas pu être unifiées avec des sous-formules de l'objectif de test, et d'ajouter les sous-formules de l'objectif de test qui n'existent pas dans l'axiome. Pour schématiser, on a la correspondance suivante entre les deux formules :

$$\begin{array}{lcccc}
 & & \text{à éliminer} & & \text{à éliminer} \\
\text{Axiome :} & \underbrace{\psi_1, \ldots, \psi_p}, & \overbrace{\xi_1, \ldots, \xi_l} & \mid\sim \underbrace{\varphi_1, \ldots, \varphi_q}, & \overbrace{\zeta_1, \ldots, \zeta_k} \\
 & \text{unification} & & \text{unification} & \\
\text{Objectif de test :} & \overbrace{\gamma_1, \ldots, \gamma_p}, & \underbrace{\gamma_{p+1}, \ldots, \gamma_m} & \mid\sim \overbrace{\delta_1, \ldots, \delta_q}, & \underbrace{\delta_{q+1}, \ldots, \delta_n} \\
 & & \text{à ajouter} & & \text{à ajouter}
\end{array}$$

La preuve va donc consister d'une part à couper une à une chaque sous-formule de l'axiome qui n'a pas pu être unifiée avec une sous-formule de l'objectif de test, donnant ainsi les contraintes sous lesquelles la preuve est possible. D'autre part, la règle de coupure permet d'ajouter des formules à un séquent, puisque dans les prémisses, les contextes Γ, Δ et Γ', Δ' peuvent être différents. Si on note Γ l'ensemble de formules $\{\gamma_1, \ldots, \gamma_p\}$, Γ' l'ensemble $\{\gamma_{p+1}, \ldots, \gamma_m\}$, Δ l'ensemble $\{\delta_1, \ldots, \delta_q\}$ et Δ' l'ensemble $\{\delta_{q+1}, \ldots, \delta_n\}$, on a alors un arbre de preuve de la forme suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \xi_1, \ldots, \xi_l \mid\sim \zeta_1, \ldots, \zeta_k, \Delta}\,\mathsf{Ax}}{\sigma(\Gamma), \sigma(\xi_1), \ldots, \sigma(\xi_l) \mid\sim \sigma(\zeta_1), \ldots, \sigma(\zeta_k), \sigma(\Delta)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\sigma(\Gamma'), \sigma(\zeta_1) \mid\sim \sigma(\Delta')}}{\sigma(\Gamma), \sigma(\Gamma'), \sigma(\xi_1), \ldots, \sigma(\xi_l) \mid\sim \sigma(\zeta_2), \ldots, \sigma(\zeta_k), \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}}{\vdots \qquad\qquad \vdots}\,\mathsf{Cut}}{\sigma(\Gamma') \mid\sim \sigma(\xi_1), \sigma(\Delta') \qquad \sigma(\Gamma), \sigma(\Gamma'), \sigma(\xi_1), \ldots, \sigma(\xi_l) \mid\sim \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma'), \sigma(\xi_2), \ldots, \sigma(\xi_l) \mid\sim \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}}{\vdots}\,\mathsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma') \mid\sim \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}$$

Après avoir effectué la substitution qui permet d'unifier certaines sous-formules de l'axiome à certaines sous-formules de la formule à prouver, $l+k$ applications de la règle de coupure permettent d'éliminer les l sous-formules du membre gauche de l'axiome et les k sous-formules de son membre droit, et permettent également l'introduction des formules de Γ' et Δ'.

La procédure est alors exprimée au travers des deux règles suivantes :

$$\textbf{Réduction}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\Gamma \mid\sim \Delta\}, \sigma')\}}{\Psi \cup \{(\sigma(\mathcal{C}), \sigma \circ \sigma')\}} \quad \exists \gamma \in \Gamma, \exists \delta \in \Delta \text{ t.q. } \sigma(\gamma) = \sigma(\delta)$$

$$\textbf{Dépliage}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\phi\}, \sigma')\}}{\Psi \cup \bigcup_{(c,\sigma) \in Dep(\phi)} \{(\sigma(\mathcal{C}) \cup c, \sigma \circ \sigma')\}}$$

où $Dep(\phi)$ pour $\phi = \gamma_1, \ldots, \gamma_m \mid\sim \delta_1, \ldots, \delta_n$ est l'ensemble défini par :

$$\left\{ \left(\begin{array}{l} \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m), \sigma(\zeta_i) \mid\sim \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1 \leq i \leq k} \\ \cup \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m) \mid\sim \sigma(\xi_i), \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1 \leq i \leq l} \end{array}, \sigma \right) \middle| \begin{array}{l} \psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mid\sim \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q \in Ax, \\ 1 \leq p \leq m, \forall 1 \leq i \leq p, \sigma(\psi_i) = \sigma(\gamma_i), \\ 1 \leq q \leq n, \forall 1 \leq i \leq q, \sigma(\varphi_i) = \sigma(\delta_i), \\ k, l \in \mathbb{N} \end{array} \right\}$$

La règle **Réduction** élimine les tautologies des ensembles de contraintes. En effet, si un ensemble contient un séquent $\Gamma \mid\sim \Delta$ tel qu'il soit possible d'unifier une sous-formule de Γ et une sous-formule de Δ, alors ce séquent est une tautologie.

La règle **Dépliage** remplace la contrainte ϕ par chacun des ensembles de contraintes de $Dep(\phi)$. S'il est possible d'unifier une partie des sous-formules de ϕ avec une partie des sous-formules d'un axiome, alors comme le montre l'arbre de preuve dessiné plus haut, il est possible de déduire ϕ à partir de cet axiome sous les contraintes qui permettent de couper chacune des sous-formules restantes de l'axiome.

Chaque unification avec un axiome génère un couple (c, σ), la formule initiale ϕ est donc remplacée par autant d'ensembles de contraintes qu'il y a d'axiomes avec lesquels elle puisse être unifiée. La définition de $Dep(\phi)$ étant fondée sur l'unification, cet ensemble est calculable si la spécification Sp a un nombre fini d'axiomes. Étant donnée une formule ϕ, on a le critère de sélection C_ϕ qui associe à tout jeu de tests contraint $T_{(\mathcal{C},\sigma'),\varphi}$ l'ensemble de jeux de tests contraints $(T_{(\sigma(\mathcal{C} \setminus \{\phi\}) \cup c, \sigma \circ \sigma'),\varphi})_{(c,\sigma) \in Dep(\phi)}$ si $\phi \in \mathcal{C}$, et lui-même sinon.

On écrit $(\Psi, \varphi) \vdash_D (\Psi', \varphi)$ si Ψ' peut être dérivé de Ψ en appliquant la règle **Réduction** ou **Dépliage**. La procédure de dépliage prend donc en entrée une spécification du premier ordre sans quantificateurs Sp et une formule du premier ordre sans quantificateurs φ et applique successivement les règles **Réduction** et **Dépliage** pour générer la suite de dérivations suivante :

$$(\Psi_0, \varphi) \vdash_D (\Psi_1, \varphi) \vdash_D (\Psi_2, \varphi) \ldots$$

La terminaison de la procédure n'est pas assurée puisqu'il n'est pas vérifié, durant l'exécution, si la formule φ est une conséquence sémantique de la spécification. Cela ne sera fait qu'à la phase de génération,

automatiquement si la spécification considérée est décidable et manuellement sinon. La procédure effectue la recherche des preuves des différentes instances sans variables de l'objectif de test jusqu'à ce que la division du jeu de tests initial soit suffisamment fine selon le testeur. Si une instance de l'objectif de test n'est pas une conséquence sémantique de la spécification, cela signifie qu'il y a, parmi les contraintes du jeu de tests associé, des contraintes qui ne peuvent pas être validées. Cette instance de l'objectif de test ne sera alors pas soumise au système comme un test.

EXEMPLE VI.2 — *Dépliage d'une formule du premier ordre sans quantificateurs.*

On veut tester la formule $estdans(r, L) \Rightarrow insérer(r, L, L')$. Remarquons que toutes les instances sans variables de cette formule ne sont pas des conséquences de la spécification. Seules celles pour lesquelles la valeur de L est égale à la valeur de L' le sont, puisque l'opération d'insertion est définie de manière à ce que les listes ne contiennent pas de doublons.

$$\Psi_0 = \{(\{estdans(r, L) \mid\sim\ insérer(r, L, L')\}, \mathrm{Id})\ \}$$

$$\begin{aligned}
\Psi_1 = \{\ & (\emptyset, \sigma_1), && (16)\\
& (\{egr(x_0/s(y_0), e_0) \mid\sim\ insérer(x_0/s(y_0), e_0 :: l_0, l'_0),\\
& \quad estdans(x_0/s(y_0), l_0) \mid\sim\ insérer(x_0/s(y_0), e_0 :: l_0, l'_0)\}, \sigma_2), && (17)\\
& (\{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim\ insérer(x_0/s(y_0), l_0, l'_0)\}, \sigma_3), && (19)\\
& (\emptyset, \sigma_4), && (12)\\
& (\{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim\ egr(x_0/s(y_0), e_0)\}, \sigma_5), && (13)\\
& (\{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim\ infr(x_0/s(y_0), e_0)\}, \sigma_6), && (14)\\
& (\{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim\ infr(e_0, x_0/s(y_0)),\\
& \quad estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim\ insérer(x_0/s(y_0), l_0, l'_0)\}, \sigma_7)\ \} && (15)
\end{aligned}$$

où

	r	L	L'	x	y	e	l	l'
σ_1	$x_0/s(y_0)$	$[\,]$		x_0	y_0			
σ_2	$x_0/s(y_0)$	$e_0 :: l_0$	l'_0	x_0	y_0	e_0	l_0	
σ_3	$x_0/s(y_0)$	l_0	l'_0	x_0	y_0		l_0	
σ_4	$x_0/s(y_0)$	$[\,]$	$x_0/s(y_0) :: [\,]$	x_0	y_0			
σ_5	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l_0$	x_0	y_0	e_0	l_0	
σ_6	$x_0/s(y_0)$	$e_0 :: l_0$	$x_0/s(y_0) :: e_0 :: l_0$	x_0	y_0	e_0	l_0	
σ_7	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l'_0$	x_0	y_0	e_0	l_0	l'_0

Chaque couple de Ψ_1 est étiqueté par le numéro de l'axiome (sous la forme de séquent normalisé) utilisé pour le dépliage de la formule initiale, c'est-à-dire l'axiome avec lequel la formule a été unifiée.

Pour reprendre les notations de la définition de la procédure, l'objectif de test $estdans(r, L) \Rightarrow insérer(r, L, L')$ est le séquent $\gamma_1 \mid\sim \delta_1$ avec $\gamma_1 = estdans(r, L)$ et $\delta_1 = insérer(r, L, L')$.

Le premier couple provient de l'unification de l'objectif de test avec l'axiome $estdans(x/s(y), [\,]) \mid\sim$.

$$\dfrac{\dfrac{}{estdans(x/s(y),[\,]) \mid\sim}\,\mathsf{Ax}}{estdans(x_0/s(y_0),[\,]) \mid\sim}\,\mathsf{Subs}$$

Comme $estdans(r,L) \mid\sim ins\acute{e}rer(r,L,L')$ avec $r = x/s(y)$ et $L = [\,]$ est une conséquence directe de cet axiome, aucune contrainte n'est générée, uniquement la substitution.

Le deuxième couple vient de l'unification de l'objectif de test avec l'axiome

$$estdans(x/s(y), e :: l) \mid\sim egr(x/s(y), e), estdans(x/s(y), l)$$

Selon les notations de la définition, cet axiome est le séquent $\psi_1 \mid\sim \zeta_1, \zeta_2$, avec $\psi_1 = estdans(x/s(y), e :: l)$, $\zeta_1 = egr(x/s(y), e)$ et $\zeta_2 = estdans(x/s(y), l)$. La substitution σ_2 permet d'unifier ψ_1 et γ_1, on peut donc déduire l'objectif de test auquel a été appliquée σ_2 de cet axiome, sous les deux contraintes $\sigma_2(\zeta_1) \mid\sim \sigma_2(\delta_1)$ et $\sigma_2(\zeta_2) \mid\sim \sigma_2(\delta_1)$. On a alors l'arbre de preuve suivant :

$$\dfrac{SA \qquad \dfrac{\vdots}{\sigma_2(\zeta_2) \mid\sim \sigma_2(\delta_1)}}{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim ins\acute{e}rer(x_0/s(y_0), e_0 :: l_0, l_0')}\,\mathsf{Cut}$$

où SA est le sous-arbre suivant :

$$\dfrac{\dfrac{\dfrac{}{estdans(x/s(y), e :: l) \mid\sim egr(x/s(y), e), estdans(x/s(y), l)}\,\mathsf{Ax}}{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim egr(x_0/s(y_0), e_0), estdans(x_0/s(y_0), l_0)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\sigma_2(\zeta_1) \mid\sim \sigma_2(\delta_1)}}{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim estdans(x_0/s(y_0), l_0), ins\acute{e}rer(x_0/s(y_0), e_0 :: l_0, l_0')}\,\mathsf{Cut}$$

Le sixième couple est issu de l'unification de l'objectif de test avec l'axiome

$$infr(x/s(y), e) \mid\sim ins\acute{e}rer(x/s(y), e :: l, x/s(y) :: e :: l)$$

Si on note cet axiome $\xi_1 \mid\sim \varphi_1$, la substitution σ_6 permet d'unifier δ_1 et φ_1. L'objectif de test auquel a été appliquée σ_6 peut donc être déduit de cet axiome sous la contrainte $\sigma_6(\gamma_1) \mid\sim \sigma_6(\xi_1)$:

$$\dfrac{\dfrac{}{\sigma_6(\gamma_1) \mid\sim \sigma_6(\xi_1)} \quad \dfrac{\dfrac{}{infr(x/s(y), e) \mid\sim ins\acute{e}rer(x/s(y), e :: l, x/s(y) :: e :: l)}\,\mathsf{Ax}}{infr(x_0/s(y_0), e_0) \mid\sim ins\acute{e}rer(x_0/s(y_0), e_0 :: l_0, x_0/s(y_0) :: e_0 :: l_0)}\,\mathsf{Subs}}{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim ins\acute{e}rer(x_0/s(y_0), e_0 :: l_0, x_0/s(y_0) :: e_0 :: l_0)}\,\mathsf{Cut}$$

Si L n'est pas la liste vide, la formule initiale $estdans(r,L) \mid\sim ins\acute{e}rer(r,L,L')$ est vraie si et seulement si L et L' prennent la même valeur. Le dépliage de cette formule lorsque L n'est pas vide va générer deux types de contraintes : celles pour lesquelles L et L' prennent la même valeur, qui vont devenir des cas de test puisque ces formules sont des conséquences de la spécification, et celles pour lesquelles L et L' seront interprétés différemment, qui n'engendreront pas de cas de test. Par exemple, le cinquième couple $(\{estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim egr(x_0/s(y_0), e_0)\}, \sigma_5)$ est un cas de test potentiel car $estdans(x_0/s(y_0), e_0 :: l_0)$ et $egr(x_0/s(y_0), e_0)$ sont vraies ou fausses simultanément pour

toute substitution close. Au contraire, le sixième couple, dont la contrainte est $estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim infr(x_0/s(y_0), e_0)$, n'engendrera pas de cas de test. En effet, si l'élément $x_0/s(y_0)$ est dans la liste $e_0 :: l_0$, comme cette liste est triée, il n'existe pas de substitution pour laquelle il est inférieur à e_0.

Comme il a déjà été expliqué plus haut, la procédure de dépliage ne peut pas distinguer ces deux types de contraintes. Cependant, avant qu'un objectif de test contraint soit soumis au système, une substitution close ρ lui est appliquée. Puisque, par définition, $\rho(\phi)$ doit être une conséquence de la spécification, les contraintes telles que L et L' sont interprétées différemment ne seront pas soumises en tant que cas de test au système, puisqu'elles seront interprétées comme fausses.

Une seconde étape de dépliage, de la formule $estdans(x_0/s(y_0), e_0 :: l_0) \mid\sim egr(x_0/s(y_0), e_0)$ par exemple engendre l'ensemble suivant :

$$\begin{aligned}
&\{\ (\{estdans(x_0/s(y_0), l_0) \mid\sim\}, \sigma'_1), && (17)\\
&\ (\{estdans(x_0/s(y_0), e_1 :: e_0 :: l_0) \mid\sim egr(x_0/s(y_0), e_0)\}, \sigma'_2), && (18)\\
&\ (\{estdans(x_0/s(y_0), u_0/s(v_0) :: l_0) \mid\sim mult(x_0, s(v_0), n_0),\\
&\quad estdans(x_0/s(y_0), u_0/s(v_0) :: l_0) \mid\sim mult(u_0, s(y_0), n_0)\}, \sigma'_3)\ \} && (10)
\end{aligned}$$

où

	r	L	L'	x	y	e	e'	l	u	v	n
σ'_1	$x_0/s(y_0)$	$e_0 :: l_0$	$e_0 :: l_0$	x_0	y_0	e_0		l_0			
σ'_2	$x_0/s(y_0)$	$e_1 :: e_0 :: l_0$	$e_1 :: e_0 :: l_0$	x_0	y_0	e_0	e_1	l_0			
σ'_3	$x_0/s(y_0)$	$u_0/s(v_0) :: l_0$	$u_0/s(v_0) :: l_0$	x_0	y_0	$u_0/s(v_0)$		l_0	u_0	v_0	n_0

Notons qu'il a fallu renommer la variable e de l'axiome (18) en e' pour ne pas associer deux valeurs différentes à la même variable. ◇

Ici, la procédure de dépliage a été définie de façon à couvrir les comportements d'un objectif de test donné, représenté par la formule φ. Il s'agit ensuite de couvrir le plus largement possible l'ensemble du jeu de tests exhaustif $Sp^\bullet \cap Obs$. Une stratégie consiste à ordonner toutes les formules du premier ordre sans quantificateurs selon leur taille, de la façon suivante :

$$\Phi_0 = \{\ \mid\sim p(x_1, \ldots, x_n) \mid p : s_1 \times \ldots \times s_n \in P, \forall i, 1 \leq i \leq n, x_i \in V_{s_i}\}$$

$$\begin{aligned}
\Phi_{n+1} = \{&p(x_1, \ldots, x_n), \Gamma \mid\sim \Delta, \quad \Gamma \mid\sim \Delta, p(x_1, \ldots, x_n) \mid\\
&\Gamma \mid\sim \Delta \in \Phi_n, p : s_1 \times \ldots \times s_n \in P, \forall i, 1 \leq i \leq n, x_i \in V_{s_i}\}
\end{aligned}$$

Ensuite, pour gérer la taille souvent infinie de $Sp^\bullet \cap Obs$, il suffit de choisir un entier $k \in \mathbb{N}$ puis d'appliquer, pour chaque i entre 1 et k, la procédure de dépliage à chaque $p(x_1, \ldots, x_n), \Gamma \mid\sim \Delta$ et $\Gamma \mid\sim \Delta, p(x_1, \ldots, x_n)$ appartenant à Φ_i. Il faut évidemment que la signature de la spécification considérée soit finie pour que Φ_i le soit également.

On prouve ici les deux propriétés qui justifient la pertinence de la méthode de sélection par dépliage dans le cadre de la logique du premier ordre sans quantificateurs. La procédure est montrée correcte et complète, elle partitionne donc de manière appropriée le jeu de tests exhaustif initial.

THÉORÈME VI.3. *Si* $(\Psi, \varphi) \vdash_U (\Psi', \varphi)$, *alors* $T_{\Psi,\varphi} = T_{\Psi',\varphi}$.

Preuve.
(Correction) Montrons que si $(\Psi, \varphi) \vdash_U (\Psi', \varphi)$, alors $T_{\Psi',\varphi} \subseteq T_{\Psi,\varphi}$.

Si la règle appliquée est **Réduction**, c'est trivial. S'il s'agit de la règle **Dépliage**, par définition, on doit prouver que pour tout $(\mathcal{C}, \sigma') \in \Psi$, pour tout $\phi \in \mathcal{C}$, pour tout $(c, \sigma) \in Dep(\phi)$, $T_{(c,\sigma\circ\sigma'),\varphi} \subseteq T_{(\{\phi\},\sigma'),\varphi}$. On doit alors prouver que pour toute substitution close $\rho : V \to T_\Sigma$ telle que $Sp \models \rho(\chi)$, pour tout $\chi \in c$, il existe $\rho' : V \to T_\Sigma$ tel que $Sp \models \rho'(\phi)$.

On pose l'hypothèse que la formule ϕ est de la forme $\gamma_1, \ldots, \gamma_m \mid\sim \delta_1, \ldots, \delta_n$, et que l'ensemble c tel que $(c, \sigma) \in Dep(\phi)$ est de la forme

$$\{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m), \sigma(\zeta_i) \mid\sim \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1\leq i\leq k}$$
$$\cup \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m) \mid\sim \sigma(\xi_i), \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1\leq i\leq l}$$

où $1 \leq p \leq m$ et $1 \leq q \leq n$ sont tels que $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mid\sim \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q \in Ax$, $\sigma(\psi_i) = \sigma(\gamma_i)$ pour tout $1 \leq i \leq p$ et $\sigma(\varphi_i) = \sigma(\delta_i)$ pour tout $1 \leq i \leq q$. On a alors l'arbre de preuve suivant, où $\Gamma = \{\psi_1, \ldots, \psi_p\}$, $\Delta = \{\varphi_1, \ldots, \varphi_q\}$, $\Gamma' = \{\gamma_{p+1}, \ldots, \gamma_m\}$, $\Delta' = \{\delta_{q+1}, \ldots, \delta_n\}$, pour tout i, $1 \leq i \leq l$, $\Omega_i = \{\xi_i, \ldots, \xi_l\}$ et pour tout i, $1 \leq i \leq k$, $\Lambda_i = \{\zeta_i, \ldots, \zeta_k\}$. Pour plus de lisibilité, la composition $\sigma' \circ \sigma$ de deux substitutions $\sigma : V \to T_\Sigma(V)$ et $\sigma' : T_\Sigma(V) \to T_\Sigma(V)$, appliquée à la formule φ, est notée $\sigma'\sigma(\varphi)$.

$$\dfrac{\dfrac{}{\rho\sigma(\Gamma') \mid\sim \rho\sigma(\xi_l), \rho\sigma(\Delta')} \quad \dfrac{\dfrac{\dfrac{}{\rho\sigma(\Gamma') \mid\sim \rho\sigma(\xi_2), \rho\sigma(\Delta')} \quad \dfrac{\dfrac{}{\rho\sigma(\Gamma') \mid\sim \rho\sigma(\xi_1), \rho\sigma(\Delta')}\ SA}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_2) \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\ \mathsf{Cut}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_l) \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\ \mathsf{Cut}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma') \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\ \mathsf{Cut}$$

où SA est le sous-arbre de preuve suivant :

$$\dfrac{\dfrac{\dfrac{\Gamma, \Omega_1 \mid\sim \Lambda_1, \Delta}{\rho\sigma(\Gamma), \rho\sigma(\Omega_1) \mid\sim \rho\sigma(\Lambda_1), \rho\sigma(\Delta)}\ \mathsf{Subs} \quad \dfrac{}{\rho\sigma(\Gamma'), \rho\sigma(\zeta_1) \mid\sim \rho\sigma(\Delta')}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_1) \mid\sim \rho\sigma(\Lambda_2), \rho\sigma(\Delta), \rho\sigma(\Delta')}\ \mathsf{Cut} \quad \dfrac{}{\rho\sigma(\Gamma'), \rho\sigma(\zeta_2) \mid\sim \rho\sigma(\Delta')}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_1) \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\ \mathsf{Cut}\ \mathsf{Cut}$$

(Complétude) Montrons que si $(\Psi, \varphi) \vdash_U (\Psi', \varphi)$, alors $T_{\Psi,\varphi} \subseteq T_{\Psi',\varphi}$.

Par définition de la règle **Dépliage**, on doit prouver que $T_{(\{\phi\},\sigma'),\varphi} \subseteq \bigcup_{(c,\sigma)\in Dep(\phi)} T_{(c,\sigma\circ\sigma'),\varphi}$. On doit alors prouver que pour chaque substitution close $\rho : V \to T_\Sigma$ telle que $Sp \models \rho(\phi)$, il existe $(c, \sigma) \in Dep(\phi)$ tel qu'il existe $\rho' : V \to T_\Sigma$ tel que $Sp \models \rho'(\chi)$ pour tout $\chi \in c$. En d'autres termes, on doit prouver que $\rho(\phi)$ peut être déduit de la spécification Sp s'il existe $(c, \sigma) \in Dep(\phi)$, et $\rho' : V \to T_\Sigma$ tels que $Sp \models \rho'(\chi)$ pour tout $\chi \in c$.

Tout d'abord, remarquons que la procédure de dépliage définit une stratégie qui limite l'espace de recherche des arbres de preuve à une classe d'arbres possédant une certaine structure. La procédure définit une stratégie de rechercher de preuve qui sélectionne des arbres de preuve dans lesquels :

- aucune instance de la règle de coupure ne se trouve au-dessus d'instances de la règle de substitution ;
- il n'y a pas d'instance de la règle de coupure dont les prémisses sont toutes les deux les conclusions d'instances de règles de coupure.

Nous devons alors prouver qu'il existe un arbre de preuve de conclusion $\rho(\phi)$ possédant cette structure. En fait, c'est un résultat plus fort que nous allons prouver : nous allons définir un ensemble de transformations élémentaires d'arbres de preuve, qui permettent de réécrire des combinaisons élémentaires de règles d'inférence, puis nous prouverons que la transformation d'arbres de preuve globale résultante est faiblement normalisante, les formes normales étant des arbres de preuves ayant la structure décrite précédemment.

Le cas où une instance de la règle de coupure se trouve au-dessus d'une instance de la règle de substitution :

$$\dfrac{\dfrac{\dfrac{\Gamma \mathrel{|\!\sim} \Delta, \varphi \quad \Gamma', \varphi \mathrel{|\!\sim} \Delta'}{\Gamma, \Gamma' \mathrel{|\!\sim} \Delta, \Delta'}\,\mathsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma') \mathrel{|\!\sim} \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Subs}}{} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \mathrel{|\!\sim} \Delta, \varphi}{\sigma(\Gamma) \mathrel{|\!\sim} \sigma(\Delta), \sigma(\varphi)}\,\mathsf{Subs} \quad \dfrac{\Gamma', \varphi \mathrel{|\!\sim} \Delta'}{\sigma(\Gamma'), \sigma(\varphi) \mathrel{|\!\sim} \sigma(\Delta')}\,\mathsf{Subs}}{\sigma(\Gamma), \sigma(\Gamma') \mathrel{|\!\sim} \sigma(\Delta), \sigma(\Delta')}\,\mathsf{Cut}$$

Le cas où deux instances de la règle de coupure se trouve au-dessus d'une troisième doit être divisé en deux cas, selon la position de la dernière formule coupée dans les prémisses de l'instance droite de la règle de coupure.

Le cas où la formule φ est dans la prémisse gauche. Par exemple :

$$\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1, \varphi \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1'}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\,\mathsf{Cut} \quad \dfrac{\Gamma_2, \varphi \mathrel{|\!\sim} \Delta_2, \varphi_2 \quad \Gamma_2', \varphi_2 \mathrel{|\!\sim} \Delta_2'}{\Gamma_2, \Gamma_2', \varphi \mathrel{|\!\sim} \Delta_2, \Delta_2'}\,\mathsf{Cut}}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\,\mathsf{Cut}$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1, \varphi \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1'}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\,\mathsf{Cut} \quad \Gamma_2, \varphi \mathrel{|\!\sim} \Delta_2, \varphi_2}{\Gamma_1, \Gamma_1', \Gamma_2 \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \varphi_2}\,\mathsf{Cut} \quad \Gamma_2', \varphi_2 \mathrel{|\!\sim} \Delta_2'}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\,\mathsf{Cut}$$

Le cas où la formule φ est dans la prémisse droite. Par exemple :

$$\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1 \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1', \varphi}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\,\mathsf{Cut} \quad \dfrac{\Gamma_2 \mathrel{|\!\sim} \Delta_2, \varphi_2 \quad \Gamma_2', \varphi_2, \varphi \mathrel{|\!\sim} \Delta_2'}{\Gamma_2, \Gamma_2', \varphi \mathrel{|\!\sim} \Delta_2, \Delta_2'}\,\mathsf{Cut}}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\,\mathsf{Cut}$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1 \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1', \varphi}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\,\mathsf{Cut} \quad \Gamma_2', \varphi_2', \varphi \mathrel{|\!\sim} \Delta_2'}{\Gamma_1, \Gamma_1', \Gamma_2', \varphi_2 \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2'}\,\mathsf{Cut} \quad \Gamma_2 \mathrel{|\!\sim} \Delta_2, \varphi_2}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\,\mathsf{Cut}$$

Pour un arbre $\pi = \dfrac{\pi_1 \quad \pi_2}{\Gamma_1, \Gamma_2 \mid\sim \Delta_1, \Delta_2}\mathsf{Cut}$, notons $m(\pi)$ la mesure de π, définie comme étant le nombre d'instances de la règle de coupure dans π_2. Un arbre de preuve est dit maximal si et seulement s'il est de la forme

$$\dfrac{\dfrac{\pi_{1_1} \quad \pi_{1_2}}{\Gamma_1 \mid\sim \Delta_1, \varphi}\mathsf{Cut} \qquad \dfrac{\pi_{2_1} \quad \pi_{2_2}}{\Gamma_2, \varphi \mid\sim \Delta_2}\mathsf{Cut}}{\Gamma_1, \Gamma_2 \mid\sim \Delta_1, \Delta_2}\mathsf{Cut}$$

et $m(\pi) = 1$. En appliquant la stratégie qui consiste à réduire les arbres de preuve maximaux, il est facile de montrer que la mesure m décroît pour chaque transformation élémentaire donnée plus haut.

Puisque par hypothèse, $Sp \models \rho(\phi)$, et comme ϕ n'est pas une tautologie, il existe nécessairement un axiome $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mid\sim \zeta_1, \ldots, \zeta_r, \varphi_1, \ldots, \varphi_q$ et une substitution close ρ' tels que $\rho'(\psi_i) = \rho'(\gamma_i)$ pour tout $1 \leq i \leq p$ et $\rho'(\varphi_i) = \rho'(\delta_i)$ pour tout $1 \leq i \leq q$. La substitution ρ' est alors un unificateur de chaque ψ_i et γ_i, et de chaque φ_i et δ_i. Il existe donc un arbre de preuve résultant de la transformation définie plus haut, de conclusion $\rho(\phi)$, où $\rho = \rho'$, et de la forme :

$$\dfrac{\dfrac{\vdots}{\rho\sigma(\Gamma') \mid\sim \rho\sigma(\xi_l), \rho\sigma(\Delta')} \quad \dfrac{\dfrac{\dfrac{\vdots}{\rho\sigma(\Gamma') \mid\sim \rho\sigma(\xi_2), \rho\sigma(\Delta')} \quad \dfrac{\dfrac{\vdots}{\rho\sigma(\Gamma') \mid\sim \rho\sigma(\xi_1), \rho\sigma(\Delta')} \quad SA}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_2) \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\mathsf{Cut}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_l) \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\mathsf{Cut}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma') \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\mathsf{Cut}$$

où SA est le sous-arbre suivant :

$$\dfrac{\dfrac{\dfrac{\Gamma, \Omega_1 \mid\sim \Lambda_1, \Delta}{\rho\sigma(\Gamma), \rho\sigma(\Omega_1) \mid\sim \rho\sigma(\Lambda_1), \rho\sigma(\Delta)}\mathsf{Subs} \quad \dfrac{\vdots}{\rho\sigma(\Gamma'), \rho\sigma(\zeta_1) \mid\sim \rho\sigma(\Delta')}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_1) \mid\sim \rho\sigma(\Lambda_2), \rho\sigma(\Delta), \rho\sigma(\Delta')}\mathsf{Cut} \quad \dfrac{\vdots}{\rho\sigma(\Gamma'), \rho\sigma(\zeta_2) \mid\sim \rho\sigma(\Delta')}}{\rho\sigma(\Gamma), \rho\sigma(\Gamma'), \rho\sigma(\Omega_1) \mid\sim \rho\sigma(\Delta), \rho\sigma(\Delta')}\mathsf{Cut}$$

et où $\Gamma = \{\psi_1, \ldots, \psi_p\}$, $\Delta = \{\varphi_1, \ldots, \varphi_q\}$, $\Gamma' = \{\gamma_{p+1}, \ldots, \gamma_m\}$, $\Delta' = \{\delta_{q+1}, \ldots, \delta_n\}$, pour tout i, $1 \leq i \leq l$, $\Omega_i = \{\xi_i, \ldots, \xi_l\}$ et pour tout i, $1 \leq i \leq k$, $\Lambda_i = \{\zeta_i, \ldots, \zeta_k\}$. □

4 Test à partir de spécifications du premier ordre

Nous avons insisté dans ce chapitre sur le fait que les spécifications considérées ne comprenaient pas de quantificateur universel ou existentiel. Nous étudions ici les raisons de cette restriction. Nous supposerons alors dans cette section que les axiomes des spécifications manipulées sont construits inductivement à partir des formules atomiques, des connecteurs booléens et des quantificateurs. Nous considérerons toujours que les formules observables sont toutes les formules sans variables.

Nous allons montrer qu'assurer l'exhaustivité de l'ensemble des conséquences sémantiques observables de la spécification revient en fait à montrer la correction du système sous test.

Étant donné un système $\mathcal{S}$, on note $Th(\mathcal{S})$ la théorie close de $\mathcal{S}$ c'est-à-dire l'ensemble

$$Th(\mathcal{S}) = \{\varphi \in For(\Sigma) \mid \mathcal{S} \models_\Sigma \varphi, \varphi \text{ close}\}$$

L'institution de la logique du premier ordre possédant la négation, d'après le théorème I.1, tout modèle $\mathcal{M}$ de $Th(\mathcal{S})$ est élémentairement équivalent à $\mathcal{S}$ sur les formules closes : pour toute formule close φ,

$$\mathcal{M} \models_\Sigma \varphi \Leftrightarrow \mathcal{S} \models_\Sigma \varphi$$

THÉORÈME VI.4. *Soit Sp une spécification consistante. Le jeu de tests $Sp^\bullet \cap Obs$ est exhaustif pour $\mathcal{K}$ si et seulement si $\mathcal{K}$ est l'ensemble des modèles finiment engendrés tels que l'ensemble $Ax \cup Th(\mathcal{S})$ est consistant :*

$$\mathcal{K} = \{\mathcal{S} \in Gen(\Sigma) \mid Ax \cup Th(\mathcal{S}) \textit{ consistant}\}$$

Preuve.

($\Leftarrow$) Soit un système $\mathcal{S} \in Gen(\Sigma)$ tel que $Ax \cup Th(\mathcal{S})$ soit consistant. On suppose que $\mathcal{S} \models Sp^\bullet \cap Obs$. Montrons que $Correct_{obs}(\mathcal{S}, Sp)$. Comme l'ensemble de formules $Ax \cup Th(\mathcal{S})$ est consistant, il existe un modèle $\mathcal{M}$ de Σ tel que $\mathcal{M} \models_\Sigma Ax \cup Th(\mathcal{S})$. Puisque $Th(\mathcal{S})$ est une théorie complète, $\mathcal{S}$ et $\mathcal{M}$ sont élémentairement équivalents sur les formules closes. Par conséquent, pour toute formule $\varphi \in Obs$, $\mathcal{M} \models \varphi \Leftrightarrow \mathcal{S} \models \varphi$. D'où $\mathcal{M} \in Mod(Sp)$.
On suppose qu'il existe un modèle $\mathcal{M} \in Mod(Sp)$ tel que $\mathcal{M} \equiv_{obs} \mathcal{S}$. Soit $\varphi \in Sp^\bullet \cap Obs$. Par hypothèse $\mathcal{M} \models \varphi$ donc $\mathcal{S} \models \varphi$ également.

($\Rightarrow$) (Par contraposée.) On suppose que $Ax \cup Th(\mathcal{S})$ n'est pas consistant. Montrons que $Sp^\bullet \cap Obs$ n'est pas exhaustif. On suppose que $\mathcal{S} \models Sp^\bullet \cap Obs$. Comme $Ax \cup Th(\mathcal{S})$ n'est pas consistant, la seule possibilité[2] est que pour tout modèle $\mathcal{M} \in Mod(Sp)$, il existe $\varphi \in Th(\mathcal{S})$ telle que $\mathcal{M} \not\models \varphi$. Comme $\mathcal{S}$ est finiment engendré, cela signifie que pour tout modèle $\mathcal{M} \in Mod(Sp)$, $\mathcal{M} \not\equiv_{obs} \mathcal{S}'$ et alors $\mathcal{M} \not\equiv_{obs} \mathcal{S}$. On en conclut alors que $Correct_{obs}(\mathcal{S}, Sp)$ n'est pas vérifié.

□

Considérons la spécification suivante des entiers relatifs.

spec RELATIFS =

type $Int ::= 0 \mid s(Int)$

op $+ : Int \times Int \to Int$

pred $< : Int \times Int$

- $\forall x, x + 0 = x$
- $\forall x, \forall y, x + s(y) = s(x + y)$
- $\forall x, \exists y, y < x$

end

Considérons alors le système $\mathcal{S}$ qui implante la sorte Int par l'ensemble des entiers naturels $\mathbb{N}$. Il est évident que ce système n'est pas correct puisqu'il ne valide pas le troisième axiome. En effet, l'ensemble de

[2] L'autre possibilité est qu'il existe $\varphi \in Ax$ telle que $\mathcal{S} \not\models \varphi$. Mais dans ce cas, $\mathcal{S} \not\models Sp^\bullet \cap Obs$ ce qui est impossible puisque nous avons supposé le contraire.

formules $Ax \cup Th(\mathcal{S})$ n'est pas consistant. Pourtant le système valide toutes les formules de RELATIFS$^\bullet \cap Obs$. En effet, comme l'opération $<$ n'est pas spécifiée, il n'est pas possible de construire, pour chaque entier, un entier qui lui soit inférieur. Il n'y a donc aucun moyen de donner de valeur de vérité à une formule construite sur ce prédicat.

Nous pouvons observer que la consistance de $Ax \cup Th(\mathcal{S})$ implique que $\mathcal{S}$ est correct par rapport à sa spécification Sp. En effet, si $Ax \cup Th(\mathcal{S})$ est consistant, il existe un modèle $\mathcal{M}$ de Σ tel que $\mathcal{M} \models Ax \cup Th(\mathcal{S})$. Alors $\mathcal{M}$ et $\mathcal{S}$ sont élémentairement équivalents. Prouver l'exhaustivité de $Sp^\bullet \cap Obs$ revient alors à prouver la correction de $\mathcal{S}$ par rapport à sa spécification. Ceci peut s'expliquer par le fait que pour tester une formule de la forme $\exists x \varphi$, il est nécessaire d'exhiber une valeur a telle que φ soit interprétée comme « vraie »par le système lorsque x est remplacé par a dans φ. Il n'y a pas de moyen effectif de trouver une telle valeur, à moins de prouver que la formule $\exists x \varphi$ est vérifiée par le système. C'est la raison pour laquelle nous avons restreint notre étude aux spécifications dont les axiomes ne font pas intervenir de quantificateurs.

SECONDE PARTIE

Test à partir de spécifications modales du premier ordre

NOUS présentons maintenant les extensions aux spécifications modales du premier ordre de la méthode de sélection par dépliage des axiomes, ainsi que les résultats d'exhaustivité sur lesquels repose la sélection. La première étape vers la généralisation à des spécifications exprimées dans la logique sous-jacente au langage COCASL est l'adaptation de la procédure de dépliage à des spécifications modales simples, munies d'une unique modalité et ayant la propriété d'être conditionnelles positives (une conjonction de formules modales implique une autre formule modale). Cette restriction sur la forme des axiomes est due à la volonté d'adapter la procédure de dépliage initialement définie pour des spécifications équationnelles conditionnelles positives. La présence d'une unique modalité permet alors d'étudier l'influence de celle-ci dans la définition de la procédure. Cependant, la généralisation du dépliage aux spécifications modales du premier ordre sans quantificateurs s'est avérée plus proche de la procédure définie pour les spécifications du premier ordre sans quantificateurs que de celle définie dans le cadre modal restreint. Ceci est dû au fait que la méthode de sélection dans ce dernier cadre tire très fortement profit de la forme des formules, laquelle n'est plus exploitable dans le cadre général. Le problème n'apparaît pas dans le cadre du premier ordre car les formules manipulées sont ramenées à une forme simple, ce qui n'est pas le cas dans le cadre modal où les modalités compliquent de fait la forme des formules.

L'étude de l'exhaustivité dans le cadre du test à partir de spécifications exprimées en logique modale repose entièrement sur les résultats de la théorie des coalgèbres présentés au chapitre III. La notion de jeu de tests exhaustif est liée à la sémantique du formalisme de spécification utilisé. Les modèles de la logique modale du premier ordre étant des coalgèbres, la preuve de l'exhaustivité d'un jeu de tests nécessite la construction d'une coalgèbre terminale, tout comme la preuve d'exhaustivité dans le cadre algébrique demandait de construire une algèbre initiale.

Suite à la présentation du test à partir de spécifications modales conditionnelles positives se trouve une étude comparative de notre approche du test des systèmes dynamiques avec celle du test de conformité. Plus précisément, nous rapprochons notre cadre de test de celui dans lequel est définie la relation de conformité *ioco* [Tre95]. Les notions de correction, d'exhaustivité, d'objectif de test sont mises en regard afin de donner une vision d'ensemble des similitudes et des différences de ces deux approches.

Chapitre VII

Sélection à partir de spécifications modales conditionnelles positives

Nous présentons dans ce chapitre un premier travail, qui consiste à étendre le cadre formel de sélection de tests présenté dans la partie précédente aux systèmes dynamiques et réactifs, spécifiés à l'aide d'une logique modale simple. La logique qui a été choisie est l'extension modale de la logique conditionnelle positive présentée à la section 3 du chapitre III. Les raisons de ce choix se trouvent à la fois dans la similitude et dans la dualité de ce formalisme avec celui de la logique équationnelle conditionnelle positive. La définition d'un jeu de tests exhaustif dans un formalisme donné étant lié au type de sémantique utilisée pour interpréter ce formalisme, utiliser une logique modale permet de tirer profit de la dualité des modèles coalgébriques par rapport aux modèles de type algébrique pour prouver l'existence d'un tel jeu de tests. D'autre part, le calcul pour cette logique modale est très similaire à celui des spécifications conditionnelles positives. Le dépliage étant fondé sur le calcul associé aux spécifications manipulées, adapter la procédure de sélection par dépliage des axiomes à ce formalisme permet de capturer ce qui différencie exactement ces deux instanciations du cadre général de test, en quoi et de quelle manière les modalités jouent un rôle.

Une précision s'avère nécessaire quant au jeu de tests à partir duquel sera effectuée la sélection. Dans le cadre du premier ordre, les calculs dont nous disposons étant corrects et complets, l'ensemble des conséquences sémantiques et celui des théorèmes de la spécification sont confondus. Il est donc possible de construire un arbre de preuve de chacune des conséquences sémantiques de la spécifications. C'est sur cette propriété que s'appuie la procédure de sélection par dépliage des axiomes. Dans le cadre des logiques modales cependant, nous ne disposons pas de preuve de la complétude des calculs que nous manipulons, bien que nous pensions que ce soit le cas de celui donné pour les spécifications conditionnelles positives. Nous ne pouvons donc pas nous appuyer sur la complétude pour déplier toutes les conséquences sémantiques observables. La procédure de dépliage sera alors restreinte aux théorèmes observables de la spécification, dont l'ensemble est inclus dans celui des conséquences sémantiques observables. C'est bien l'exhaustivité de ce dernier ensemble qui sera montrée, mais le dépliage sera effectué sur le sous-ensemble des formules possédant une preuve.

1 Exhaustivité

Le point de départ de la définition d'un jeu de tests exhaustif de référence est la définition de la notion d'observabilité. Comme dans le cadre de la sélection de tests à partir de spécifications équationnelles conditionnelles positives, les cas de tests sont ici réduits à des formules d'une forme plus simple que celle des axiomes, appelées par la suite formules modales, et qui sont de la forme $[\alpha_1] \dots [\alpha_m] t = t'$, avec $\alpha_1, \dots, \alpha_m$ des termes de sorte non-observable et t et t' des termes de sorte observable. De plus, afin de pouvoir être interprétées par le système sous test, les formules qui vont être soumises au système sont nécessairement sans variables. L'ensemble Obs des formules observables est alors l'ensemble suivant :

$$Obs = \{[\alpha_1] \dots [\alpha_m] t = t' \mid \forall i, 1 \leq i \leq m, \alpha_i \in T_{\Sigma_s}, s \in T, t, t' \in T_{\Sigma_{s'}}, s' \in S_{obs}\}$$

On va montrer que l'ensemble des conséquences sémantiques observables de la spécification, $Sp^\bullet \cap Obs$, est exhaustif pour une certaine classe $\mathcal{K}$ de systèmes. Les systèmes appartenant à cette classe doivent respecter deux conditions : d'une part, les modèles du premier ordre sous-jacents à ces systèmes doivent être finiment engendrés, et d'autre part, ces systèmes doivent être terminaux, au sens qui va être défini. Cette condition de terminalité trouve sa justification de la même manière que la condition d'initialité posée à la section 1 du chapitre V. Prenons l'exemple de la machine à thé suivante, qui n'accepte que des pièces de deux euros, et qui délivre un thé immédiatement après l'introduction d'une pièce.

spec MACHINE =
  types $Bool ::= true \mid false$
    $Nat ::= 0 \mid 1 \mid 2$
  cotype $\acute{E}tat$
  obs $montant : \acute{E}tat \to Nat$
    $init : \acute{E}tat \to \acute{E}tat$
    $2euros : \acute{E}tat \to \acute{E}tat$
    $th\acute{e} : \acute{E}tat \to \acute{E}tat$
  vars $m : Nat$
  • $[init]\, montant = 0$
  • $montant = 0 \Rightarrow [2euros]\, montant = 2$
  • $montant = 2 \Rightarrow [th\acute{e}]\, montant = 0$
  • $montant = 1 \Rightarrow true = false$
end

Les conséquences sémantiques observables de cette spécification sont les formules de la forme $[\alpha_1] \dots [\alpha_m]\, montant = m$ où $m = 0$ ou $m = 2$, et où $\alpha_1, \dots, \alpha_m \in \{init, 2euros, th\acute{e}\}$ sont tels que $2euros$ est toujours précédé de $init$ ou $th\acute{e}$ et $th\acute{e}$ est toujours précédé de $2euros$. Autrement dit, les états accessibles sont ceux dans lequels le montant est égal à 0 ou à 2. Considèrons un système implantant cette spécification tel que l'état dans lequel $montant = 1$ soit accessible. Ce système implante d'autre part correctement les entiers naturels et les booléens. Ce système valide bien toutes les conséquences sémantiques

observables de MACHINE, mais ne valide pas le dernier axiome puisque dans l'état où $montant$ est égal à 1, $true$ et $false$ ne sont pas égaux. Le jeu de tests MACHINE$^\bullet \cap Obs$ n'est donc pas valide, un système incorrect pouvant passer ce jeu de tests en succès.

On retrouve un problème similaire à celui décrit à la section 1 du chapitre V : la prémisse de cet axiome n'étant pas (la conclusion d') une conséquence sémantique de la spécification, sa conclusion n'est pas un test, le système peut donc passer tous les cas de test de $Sp^\bullet \cap Obs$ en succès tout en ne validant pas cet axiome. Il faut alors imposer que si la spécification ne donne aucun moyen d'atteindre un état validant une certaine propriété, il ne faut pas qu'un tel état soit présent dans le système. Toutes les propriétés vérifiées par le système doivent être spécifiées comme étant accessibles, c'est-à-dire que pour chacune de ces propriétés φ, il doit exister une conséquence sémantique de la spécification de la forme $[\alpha_1] \dots [\alpha_m]\varphi$. Comme la condition d'initialité, la condition à imposer au système ne doit pas lui permettre d'avoir un comportement qui ne serait pas prévu par la spécification. Le modèle de la spécification dont le comportement est exactement celui décrit par la spécification est le modèle terminal de cette spécification, on appelle donc cette propriété la terminalité du système.

DÉFINITION VII.1 — *Terminalité.*

Soit $Sp = (\Sigma, Ax)$ une spécification modale conditionnelle positive. Soit $\mathcal{S} = (E, \gamma) \in Mod(\Sigma)$ un système. Soit φ une formule modale sans variables. Le système $\mathcal{S}$ est terminal pour φ si et seulement si :

$$\forall e \in E, \forall \nu : V \to T_\Sigma, \mathcal{S} \models_{\nu,e} \varphi \Rightarrow \begin{cases} \exists \alpha_1, \dots, \alpha_m \in T_{\Sigma_s}, s \in T, \\ \exists e_0, e_1, \dots, e_{m-1} \in E, e = \nu_{e_{m-1}}(\alpha_m), \dots, e_1 = \nu_{e_0}(\alpha_1) \\ \wedge\ Sp \models [\alpha_1] \dots [\alpha_m]\varphi \end{cases}$$

Pour établir l'exhaustivité de $Sp^\bullet \cap Obs$, on va imposer cette condition au système pour toutes les instances sans variables des prémisses des axiomes de la spécification. On assure ainsi qu'un système validant toutes les conséquences sémantiques observables de la spécification valide les axiomes. En effet, pour un axiome donné, si une des prémisses φ n'est pas accessible selon la spécification, c'est-à-dire qu'il n'existe pas de formule dans $Sp^\bullet$ de la forme $[\alpha_1] \dots [\alpha_m]\varphi$, alors le système sous test étant terminal, il ne comportera pas d'état qui valide φ, il vérifiera donc l'axiome.

En pratique, la propriété φ est la plupart du temps une équation portant sur un attribut donné. Un attribut (par exemple $montant$) est généralement associé à au moins une méthode dont le rôle est précisément de faire évoluer la composante de l'état correspondant à cet attribut (par exemple $2euros$). On trouve alors dans la spécification les axiomes spécifiant la manière dont cette méthode modifie l'attribut, par exemple $montant = 0 \Rightarrow [2euros]\, montant = 2$.

THÉORÈME VII.1. *Soit $Sp = (\Sigma, Ax)$ une spécification. L'ensemble $Sp^\bullet \cap Obs$ des conséquences observables de la spécification est exhaustif pour tout système $\mathcal{S}$ dont le modèle du premier ordre sous-jacent est finiment engendré, et qui est terminal pour toutes les instances sans variables des formules modales qui apparaissent dans les prémisses des axiomes de Ax.*

Preuve. Soit $\mathcal{S}$ un système sous test, i.e. $\mathcal{S} \in Mod(\Sigma)$. On suppose que $\mathcal{S} \models Sp^\bullet \cap Obs$. Montrons que $Correct_{obs}(\mathcal{S}, Sp)$.

Puisque $\mathcal{S} \in Mod(\Sigma)$, $\mathcal{S}$ est une $\mathcal{F}$-coalgèbre (E, α), construite au-dessus d'une structure du premier

ordre $\mathcal{A}$, où $\mathcal{F}$ est le foncteur défini à la section 3. D'après le théorème III.5, $\mathcal{F}$ étant polynomial, $Mod(\Sigma) = \mathsf{Coalg}(\mathcal{F})$ admet une coalgèbre terminale, notée $\mathcal{T}$.

Définissons alors l'ensemble de formules sans variables $Th(\mathcal{S}) = \{\varphi \mid \mathcal{S} \models \varphi\}$. On note $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ la sous-catégorie pleine de $\mathsf{Coalg}(\mathcal{F})$, dont les objets sont tous les $Th(\mathcal{S})$-modèles. On voit aisément que $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ est une covariété. D'après le théorème III.6, la catégorie $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ admet alors un modèle terminal. On le note $\mathcal{T}/_{Th(\mathcal{S})}$. La catégorie $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ étant close par sous-coalgèbre, on note $\mathcal{T}/_{\mathcal{S}} = (E', \alpha')$, où $E' = h(E)$ et où h est l'unique homomorphisme de $\mathcal{S}$ dans $\mathcal{T}/_{Th(\mathcal{S})}$. Par construction, on a $\mathcal{S} \equiv_{Obs} \mathcal{T}/_{\mathcal{S}}$. En fait, on a un résultat plus fort. En effet, si on note q l'homomorphisme de $\mathcal{S}$ dans $\mathcal{T}/_{\mathcal{S}}$, alors l'ensemble $\{(e, q(e)) \mid e \in E\}$ est une bisimulation. Par conséquent, q est un homomorphisme élémentaire, i.e. pour toute formule sans variables φ et pour tout $e \in E$, $\mathcal{S} \models_e \varphi \iff \mathcal{T}/_{\mathcal{S}} \models_{q(e)} \varphi$.

Soit $\varphi_1 \wedge \ldots \wedge \varphi_n \Rightarrow \varphi$ une formule de Ax. Soit $\iota : V \to M$ une interprétation et soit e un état de $\mathcal{T}/_{\mathcal{S}}$ tel que pour tout i, $1 \leq i \leq n$, $\mathcal{T}/_{\mathcal{S}} \models_{\iota,e} \varphi_i$. Comme le modèle du premier ordre sous-jacent à $\mathcal{S}$ est finiment engendré, alors celui de $\mathcal{T}/_{\mathcal{S}}$ l'est également, et il existe ψ_i et ψ tels que $\mathcal{T}/_{\mathcal{S}} \models_{\iota,e} \varphi_i \iff \mathcal{T}/_{\mathcal{S}} \models_e \psi_i$ et $\mathcal{T}/_{\mathcal{S}} \models_{\iota,e} \varphi \iff \mathcal{T}/_{\mathcal{S}} \models_e \psi$. Par définition, $\mathcal{T}/_{\mathcal{S}}$ est un quotient de $\mathcal{S}$ et on a vu que q est un homomorphisme élémentaire. Par conséquent, $\mathcal{S} \models_{e'} \psi_i$ pour tout $e' \in q^{-1}(e)$. Comme $\mathcal{S}$ est terminal pour φ_i, il existe $\alpha^1_{e',\psi_i}, \ldots, \alpha^{m_i}_{e',\psi_i}$ et il existe $e'' \in \mathcal{E}$ tel que $e'' \; \mathcal{R}_{\alpha^1_{e',\psi_i}} \bullet \ldots \bullet \mathcal{R}_{\alpha^{m_i}_{e',\psi_i}} \; e'$ et $Sp \models [\alpha^1_{e',\psi_i}] \ldots [\alpha^{m_i}_{e',\psi_i}]\psi_i$. Donc $Sp \models [\alpha^1_{e',\psi_i}] \ldots [\alpha^{m_i}_{e',\psi_i}]\psi$, et $\mathcal{S} \models [\alpha^1_{e',\psi_i}] \ldots [\alpha^{m_i}_{e',\psi_i}]\psi$. On en déduit que $\mathcal{T}/_{\mathcal{S}} \models [\alpha^1_{e',\psi_i}] \ldots [\alpha^{m_i}_{e',\psi_i}]\psi$, et alors $\mathcal{T}/_{\mathcal{S}} \models_e \psi$. On a donc $\mathcal{T}/_{\mathcal{S}} \models_e \varphi$. □

Sous ces hypothèses, nous disposons maintenant d'un jeu de tests exhaustif de référence à partir duquel commencer la procédure de sélection.

2 Dépliage

Nous adaptons ici la méthode de sélection de tests par dépliage des axiomes, définie dans la partie précédente à partir de spécifications du premier ordre, aux spécifications modales conditionnelles positives. Comme nous l'avons déjà expliqué, cette méthode consiste à partitionner le jeu de tests initial $Sp^\bullet \cap Obs$, dont nous avons montré l'exhaustivité à la section précédente, en sous-jeux de tests, à l'aide des axiomes de la spécification. Comme dans le cadre de la sélection à partir de spécifications équationnelles conditionnelles positives, le jeu de test initial étant défini de la façon suivante :

$$\begin{aligned} Sp^\bullet \cap Obs = \{[a_1] \ldots [a_m] f(t_1, \ldots, t_n) = t \mid\; & f : s_1 \times \ldots \times s_n \times s \to s' \in F_s, s' \in S_{obs}, \\ & a_1, \ldots, a_m, t_1, \ldots, t_n, t \in T_\Sigma, \\ & Sp \models [a_1] \ldots [a_m] f(t_1, \ldots, t_n) = t\} \end{aligned}$$

diviser cet ensemble revient à diviser chacun des sous-ensembles associé à une opération f donnée, où f est ici un attribut, ou observateur de sorte observable. Le sous-jeu de tests exhaustif associé à un attribut est appelé son domaine.

DÉFINITION VII.2 — *Domaine d'un attribut.*

Soit $Sp = (\Sigma, Ax)$ une spécification modale conditionnelle positive. Soit $f : s_1 \times \ldots \times s_n \times s \to s' \in F_s$

un attribut. Le *domaine* de f, noté $T(Sp)_{|f}$, est l'ensemble défini par :

$$T(Sp)_{|f} = \{\, [a_1]\dots[a_m]f(t_1,\dots,t_n) = t \mid\ a_1,\dots,a_m,t_1,\dots,t_n,t \in T_\Sigma,\ Sp \models [a_1]\dots[a_m]f(t_1,\dots,t_n) = t \,\}$$

C'est cet ensemble qui va être partitionné, pour chaque attribut, à l'aide des axiomes qui le spécifient.

EXEMPLE VII.1 — *Distributeur automatique de billets.*

Nous prenons ici l'exemple d'un distributeur automatique de billets qui permet à des clients d'accéder à leur compte bancaire pour y faire des retraits ou pour vérifier leur solde. Le client introduit tout d'abord sa carte, puis saisit son code (PIN pour *Personal Identification Number*). Si le code entré est bon, le client peut effectuer une transaction, qui consiste soit à consulter son solde soit à retirer de l'argent liquide. Si le code saisi est erroné trois fois de suite, la carte n'est pas rendue au client. Si le client demande à effectuer un retrait, il saisit un montant qui ne doit pas aller au-delà du seuil autorisé pour ce compte. Si ce seuil est dépassé, le retrait n'est pas autorisé. Sinon, si le montant demandé est disponible dans la machine, le client reçoit l'argent qu'il a demandé, et peut récupérer sa carte.

Il est possible d'associer à un distributeur la signature donnée à la page suivante.

types $Bool ::= true \mid false$

$Nat ::= 0 \mid 1 \mid 2$

cotype $\acute{E}tat$

ops $PIN : Nat \to Nat$

$solde : Nat \to Nat$

$seuil : Nat \to Nat$

obs

$carte : \acute{E}tat \to Nat$	$code : \acute{E}tat \to Nat$
$montant : \acute{E}tat \to Nat$	$essais : \acute{E}tat \to Nat$
$écran : \acute{E}tat \to Nat$	$montantDAB : \acute{E}tat \to Nat$
$Carte? : Nat \times \acute{E}tat \to \acute{E}tat$	$Code? : Nat \times \acute{E}tat \to \acute{E}tat$
$Consulte? : \acute{E}tat \to \acute{E}tat$	$Retrait? : \acute{E}tat \to \acute{E}tat$
$Montant? : Nat \times \acute{E}tat \to \acute{E}tat$	$RendreCarte? : \acute{E}tat \to \acute{E}tat$
$Solde! : \acute{E}tat \to \acute{E}tat$	$MauvaisCode! : \acute{E}tat \to \acute{E}tat$
$CarteGardée! : \acute{E}tat \to \acute{E}tat$	$Billets! : \acute{E}tat \to \acute{E}tat$
$PasAssez! : \acute{E}tat \to \acute{E}tat$	$Seuil! : \acute{E}tat \to \acute{E}tat$

En fonction d'un numéro de carte A, le distributeur peut savoir le code de la carte $PIN(A)$, le solde du compte associé à la carte $solde(A)$ et le montant maximum de retrait autorisé $seuil(A)$.

L'état de la machine est connu au travers de six attributs : $carte$ donne le numéro de la carte insérée s'il y en a une, 0 sinon ; $code$ donne le code saisi s'il existe et 0 sinon ; $montant$ donne le montant demandé si un montant a été entré et 0 sinon ; $montantDAB$ donne le montant total disponible dans la machine ; $essais$ donne le nombre de codes erronés entrés à la suite depuis qu'une nouvelle carte a été insérée ; $écran$ donne

l'affichage courant à l'écran, 0 si rien n'est affiché [1].

Comme dans le cadre des systèmes de transitions étiquetés à entrées et sorties, l'état du distributeur évolue grâce aux communications entre le client et la machine, qui sont des émissions et des réceptions à travers des canaux de communication. Afin de garder les notations habituelles, bien que cela n'ait aucune influence sur la sémantique, les noms de méthodes représentant des réceptions seront suivies d'un point d'interrogation tandis que les noms de méthodes représentant des émissions seront suivies d'un point d'exclamation.

- Les réceptions, du point de vue du distributeur, sont des actions effectuées par le client : l'insertion d'une carte dans la machine $Carte?$; la saisie d'un code $Code?$; la demande de consultation du solde d'un compte $Consulte?$; la demande de retrait $Retrait?$; la saisie d'un montant à retirer $Montant?$; la demande de récupération de la carte $RendreCarte?$.
- Les émissions sont les actions effectuées par la machine, qui sont principalement des messages au client, à l'exception de l'émission des billets : $Solde!$ affiche le solde du compte ; $MauvaisCode!$ dit au client que le code saisi est erroné ; $CarteGardée!$ signale que la carte a été avalée après trois essais infructueux de saisie d'un code ; $Billets!$ donne au client l'argent qu'il a demandé ; $PasAssez!$ signale que la machine ne dispose pas de suffisamment d'argent pour donner au client le montant qu'il a demandé ; $Seuil!$ signale au client qu'il n'est pas autorisé à retirer le montant qu'il a demandé car il dépasse son seuil autorisé.

Une spécification de ce distributeur dans notre formalisme peut alors être la suivante [2]. Ne nous intéressant qu'au test de systèmes dynamiques, nous ne donnons ici que les axiomes spécifiant les attributs et les méthodes. Nous supposons que les fonctions sur les données ont été spécifiées séparément.

- $carte = 0 \Rightarrow [Carte?A]\ \ carte = A$
- $carte = A \wedge A \neq 0 \Rightarrow [Carte?C]\ \ carte = A$
- $[RendreCarte?]\ \ carte = 0$
- $[CarteGardée!]\ \ carte = 0$
- $[Carte?A]\ \ carte = A \Rightarrow [Carte?A]\ \ code = 0$
- $code = 0 \Rightarrow [Code?c]\ \ code = c$
- $code = c \wedge c \neq 0 \Rightarrow [Code?d]\ \ code = c$
- $[RendreCarte?]\ \ code = 0$
- $code \neq PIN(carte) \Rightarrow [MauvaisCode!]\ \ code = 0$
- $[CarteGardée!]\ \ code = 0$
- $[Retrait?]\ \ montant = 0$
- $[CarteGardée!]\ \ montant = 0$
- $[Carte?A]\ \ carte = A \Rightarrow [Carte?A]\ \ montant = 0$
- $montant = 0 \wedge code = PIN(carte) \Rightarrow [Montant?M]\ \ montant = M$
- $montant = M \wedge M \neq 0 \Rightarrow [Montant?N]\ \ montant = M$
- $[RendreCarte?]\ \ montant = 0$
- $montant \leq seuil(A) \wedge montant \leq montantDAB \Rightarrow [Billets!]\ \ montant = 0$
- $montant \leq seuil(A) \wedge montant > montantDAB \Rightarrow [PasAssez!]\ \ montant = 0$

[1] L'écran est très simplifié ici, puisqu'il ne peut afficher qu'un entier. L'utilisation de chaînes de caractères n'aurait fait qu'alourdir inutilement l'exemple.

[2] Pour ne pas rendre la spécification trop lourde, les fonctions booléennes sont utilisées comme des prédicats. De façon évidente, une formule de la forme $c \neq 0$ devrait être écrite $(c \neq 0) = true$.

- $montant > seuil(A) \Rightarrow [Seuil!]\ montant = 0$
- $montant \leq seuil(A) \wedge montant \leq montantDAB$
  $\Rightarrow [Billets!]\ montantDAB = montantDAB - montant$
- $[Carte?A]\ carte = A \Rightarrow [Carte?A]\ essais = 0$
- $code \neq PIN(carte) \Rightarrow [MauvaisCode!]\ essais = essais + 1$
- $essais > 2 \Rightarrow [CarteGardée!]\ essais = 0$
- $[RendreCarte?]\ essais = 0$
- $[Carte?A]\ carte = A \Rightarrow [Carte?A]\ écran = 0$
- $[Retrait?]\ écran = 0$
- $code = PIN(carte) \Rightarrow [Consulte?][Solde!]\ écran = solde(carte)$

Les axiomes de la spécification donne, pour chaque attribut, les actions qui le modifient. De façon à ce que la spécification ne soit pas trop lourde, nous ne faisons pas figurer les axiomes exprimant le fait que certaines actions ne modifient pas un attribut. Dans le cas de l'attribut $montant$ par exemple, les axiomes suivants devraient apparaître dans la spécification :

- $montant = M \Rightarrow [Code?c]\ montant = M$
- $montant = M \Rightarrow [MauvaisCode!]\ montant = M$
- $montant = M \Rightarrow [Consulte?]\ montant = M$
- $montant = M \Rightarrow [Solde!]\ montant = M$ ◇

Comme nous l'avons déjà vu lors de la présentation du dépliage dans le cadre des spécifications conditionnelles positives, partitionner ce jeu de tests revient en fait à partitionner l'ensemble des substitutions closes qui, appliquées à l'objectif de test $[\alpha_1] \ldots [\alpha_m] f(x_1, \ldots, x_n) = y$, forment le domaine de l'attribut f. Cependant, l'interprétation d'un terme construit sur un attribut dépendant de l'état du système, une substitution close ne caractérise pas à elle seule un test, l'état ayant son rôle à jouer. Un cas de test sera caractérisé également par un chemin, menant à un état dans lequel l'objectif de test, clos par la substitution, est vérifié. Découper le jeu de tests initial revient alors à découper l'ensemble des substitutions closes mais également l'ensemble des états du système, selon l'instance sans variables de l'objectif de test qu'ils valident.

Par exemple, le domaine de l'attribut $écran$ est le jeu de tests

$$\{[\sigma(\alpha_1)] \ldots [\sigma(\alpha_m)]\ écran = \sigma(n) \mid \sigma : V \to T_\Sigma, Sp \models [\sigma(\alpha_1)] \ldots [\sigma(\alpha_m)]\ écran = \sigma(n)\}$$

C'est l'ensemble des suites d'actions $\alpha_1 \ldots \alpha_m$, également appelées chemins ou traces, et des substitutions closes σ, telles que l'équation sans variables $écran = \sigma(n)$ soit validée dans l'état atteint par l'instance $\sigma(\alpha_1) \ldots \sigma(\alpha_m)$ de ce chemin, si un tel état existe. Chaque instance sans variables de l'équation peut être validée en plusieurs états, accessibles par différents chemins. Tester l'attribut $écran$ revient alors à chercher les couples composés d'un chemin et d'une substitution close tels que les états atteints par le chemin valident l'équation à laquelle a été appliquée la substitution.

Les axiomes spécifiant l'attribut $écran$ sont les trois derniers axiomes de la spécification donnée plus haut. Ils nous permettent de dériver les jeux de tests suivants :

– le jeu de tests correspondant à la substitution $\sigma_1 : n \mapsto 0$ et au chemin $Carte?A_0$,

$$\{[Carte?A_0]\ écran = 0 \mid [Carte?A_0]\ carte = A_0\}$$

– le jeu de tests correspondant à la substitution $\sigma_2 : n \mapsto 0$ et au chemin $Retrait?$,

$$\{[Retrait?]\ écran = 0\}$$

– le jeu de tests correspondant à la substitution $\sigma_3 : n \mapsto solde(carte)$ et au chemin $Consulte?\ Solde!$,

$$\{[Consulte?][Solde!]\ écran = solde(carte) \mid code = PIN(carte)\}$$

Nous ne considérons pas, dans cet exemple, les autres axiomes spécifiant l'attribut $écran$, ces axiomes exprimant le fait que les autres méthodes n'ont pas d'influence sur la composante de l'état représentée par cet attribut.

On voit alors qu'un jeu de tests est caractérisé non seulement par une subtitution et un chemin, mais également par une autre contrainte sous forme d'un ensemble de formules, provenant, comme on le verra par la suite de façon rigoureuse, des prémisses de l'axiome à partir duquel a été dérivé le jeu de tests. Ce triplet formé d'un ensemble de contraintes, d'une substitution et d'un chemin caractérise ce qu'on appelle un jeu de tests contraint.

DÉFINITION VII.3 — *Jeu de tests contraint.*

Soit $Sp = (\Sigma, Ax)$ une spécification modale conditionnelle positive. Soit $f : s_1 \times \ldots \times s_n \times s \to s' \in F_s$ un attribut et $x_1, \ldots, x_n, y \in V$. Soit $\mathcal{C}$ un ensemble de formules modales appelées Σ-*contraintes*, soit $\sigma : V \to T_\Sigma(V)$ une substitution, et soit $\Delta = \alpha_1 \ldots \alpha_m$ une suite de termes de sorte non-observable appelée *trace*.

Le *jeu de tests pour* f *contraint par* $\mathcal{C}$, σ *et* Δ, noté $T_{(\mathcal{C},\sigma,\Delta),f}$, est l'ensemble de formules sans variables défini par :

$$T_{(\mathcal{C},\sigma,\Delta),f} = \{[\gamma_1] \ldots [\gamma_q][\rho(\sigma(\alpha_1))] \ldots [\rho(\sigma(\alpha_m))]\rho(\sigma(f(x_1, ..., x_n)) = \rho(\sigma(y)) \mid \gamma_1, \ldots, \gamma_q \in T_\Sigma, \rho : V \to T_\Sigma, \forall c \in \mathcal{C}, Sp \models [\gamma_1] \ldots [\gamma_q]\rho(c)\}$$

Le couple $((\mathcal{C}, \sigma, \Delta), [\alpha_1] \ldots [\alpha_m] f(x_1, ..., x_n) = y)$ est appelé *objectif de test contraint*.

Il est à noter que le domaine d'un attribut f peut être vu comme un jeu de tests contraint par l'unique contrainte $[\alpha_1] \ldots [\alpha_m] f(x_1, \ldots, x_n) = y$, la substitution identité et un chemin vide [3] :

$$T(Sp)_{|f} = T_{(\{[\alpha_1]\ldots[\alpha_m]f(x_1,\ldots,x_n)=y\},\mathrm{Id},\_),f}$$

De la même manière que dans le cadre du premier ordre, les trois jeux de tests donnés plus haut peuvent être déduits des différents arbres de preuve permettant de prouver l'objectif de test. Chaque arbre sera en fait la preuve d'une instance de l'objectif de test, c'est-à-dire, dans notre cadre, d'une formule sans variables de la forme $[a_1] \ldots [a_m]\ écran = t$. On aura ainsi prouvé qu'il existe un chemin menant à un état dans lequel la formule sans variables $écran = t$ est vérifiée.

La première instance de l'objectif de test peut être prouvée à partir de l'axiome $[Carte?A]\ carte = A \Rightarrow [Carte?A]\ écran = 0$ de la manière suivante :

[3] On note _ la trace vide et _._ la concaténation de deux traces.

$$\dfrac{\dfrac{\dfrac{}{[Carte?A]\ \ carte = A \Rightarrow [Carte?A]\ \ écran = 0}\mathsf{Ax}}{[Carte?A_0]\ \ carte = A_0 \Rightarrow [Carte?A_0]\ \ écran = 0}\mathsf{Subs} \quad \dfrac{\vdots}{[Carte?A_0]\ \ carte = A_0}}{[Carte?A_0]\ \ écran = 0}\mathsf{MP}$$

La substitution vient de l'unification de l'objectif de test avec la conclusion de l'axiome, le chemin est la partie de la conclusion de l'axiome qui n'a pas pu être unifiée et l'ensemble de contraintes est l'ensemble des lemmes intermédiaires qu'il reste à prouver pour compléter l'arbre de preuve. Autrement dit, il est possible d'atteindre un état qui valide $écran = 0$ par un chemin constitué de l'unique transition $Carte?A_0$ si l'état courant valide la formule $[Carte?A_0]\ \ carte = A_0$. Le deuxième jeu de tests est associé à l'arbre de preuve trivial suivant :

$$\dfrac{}{[Retrait?]\ \ écran = 0}\mathsf{Ax}$$

Il est donc possible en tout état d'atteindre un état dans lequel $écran = 0$ si l'action $Retrait?$ est possible. Ce jeu de tests ne pourra plus être déplié avec les axiomes dont on dispose. Le troisième jeu de tests est dérivé de l'arbre de preuve suivant :

$$\dfrac{\dfrac{}{code = PIN(carte) \Rightarrow [Consulte?][Solde!]\ \ écran = solde(carte)}\mathsf{Ax} \quad \dfrac{\vdots}{code = PIN(carte)}}{[Consulte?][Solde!]\ \ écran = solde(carte)}\mathsf{MP}$$

Si dans l'état courant, $code = PIN(carte)$, alors après avoir effectué les actions $Consulte?$ puis $Solde!$, $écran = solde(carte)$ est vérifié.

Le premier et le troisième jeu de tests faisant apparaître des contraintes, il est possible de les déplier de nouveau, de façon à essayer de compléter l'arbre de preuve correspondant. En dépliant le premier jeu de tests par rapport au premier axiome de la spécification $carte = 0 \Rightarrow [Carte?A]\ \ carte = A$, on obtient le jeu de tests suivant :

$$\{[Carte?A_0]\ \ écran = 0 \mid\ carte = 0\}$$

L'arbre de preuve de la contrainte $[Carte?A_0]\ \ carte = A_0$ est celui-ci :

$$\dfrac{\dfrac{\dfrac{}{carte = 0 \Rightarrow [Carte?A]\ \ carte = A}\mathsf{Ax}}{carte = 0 \Rightarrow [Carte?A_0]\ \ carte = A_0}\mathsf{Subs} \quad \dfrac{\vdots}{carte = 0}}{[Carte?A_0]\ \ carte = A_0}\mathsf{MP}$$

On voit alors qu'on a complété l'arbre de preuve associé au premier jeu de tests :

$$\dfrac{\dfrac{\dfrac{}{[Carte?A]\ \ carte{=}A \Rightarrow [Carte?A]\ \ écran{=}0}\mathsf{Ax}}{[Carte?A_0]\ \ carte{=}A_0 \Rightarrow [Carte?A_0]\ \ écran{=}0}\mathsf{Subs} \quad \dfrac{\dfrac{\dfrac{}{carte{=}0 \Rightarrow [Carte?A]\ \ carte{=}A}\mathsf{Ax}}{carte{=}0 \Rightarrow [Carte?A_0]\ \ carte{=}A_0}\mathsf{Subs} \quad \dfrac{\vdots}{carte{=}0}}{[Carte?A_0]\ \ carte{=}A_0}\mathsf{MP}}{[Carte?A_0]\ \ écran{=}0}\mathsf{MP}$$

On remarque que ce nouveau dépliage fait apparaître la contrainte $carte = 0$ mais ne modifie ni le chemin ni la substitution. Le chemin n'est pas modifié puisque la contrainte $[Carte?A_0]\ \ carte = A_0$ a pu être unifiée avec la conclusion complète de l'axiome. Cet axiome étant le seul avec lequel la contrainte peut s'unifier, la deuxième étape de dépliage pour ce jeu de tests est terminée. Il est bien sûr possible de procéder à un troisième dépliage, la contrainte $carte = 0$ s'unifiant avec deux axiomes de la spécification.

Une deuxième étape de dépliage pour le troisième jeu de tests donne les deux jeux de tests suivants :

- le jeu de tests correspondant à l'unification de la contrainte $code = PIN(carte)$ avec l'axiome $code = 0 \Rightarrow [Code?c]\ \ code = c$

$$\{[Code?\ PIN(carte)][Consulte?][Solde!]\ \ écran = solde(carte) \mid\ code = 0\}$$

- le jeu de tests correspondant à l'unification avec l'axiome $code = c \wedge c \neq 0 \Rightarrow [Code?d]\ \ code = c$

$$\{[Code?d_0][Consulte?][Solde!]\ écran = solde(carte) \mid\ code = PIN(carte), PIN(carte) \neq 0\}$$

L'arbre de preuve associé au dépliage de la contrainte à l'aide du premier axiome est le suivant :

$$\dfrac{\dfrac{}{code = 0 \Rightarrow [Code?c]\ \ code = c}\,\mathsf{Ax} \qquad \vdots}{\dfrac{code = 0 \Rightarrow [Code?\ PIN(carte)]\ \ code = PIN(carte)}{[Code?\ PIN(carte)]\ \ code = PIN(carte)}\,\mathsf{MP}}\,\mathsf{Subs} \qquad \dfrac{\vdots}{code = 0}$$

ce qui donne l'arbre de preuve suivant, associé aux deux dépliages consécutifs :

$$\dfrac{\dfrac{\dfrac{}{code{=}PIN(carte) \Rightarrow [Consulte?][Solde!]\ \ écran{=}solde(carte)}\,\mathsf{Ax}}{[Code?\ PIN(carte)]\ \ code{=}PIN(carte) \Rightarrow [Code?\ PIN(carte)][Consulte?][Solde!]\ \ écran{=}solde(carte)}\,\mathsf{Nec} \qquad SA_1}{[Code?\ PIN(carte)][Consulte?][Solde!]\ \ écran{=}solde(carte)}\,\mathsf{MP}$$

où SA_1 est l'arbre précédent. On remarque qu'il faut appliquer la règle de nécessitation pour pouvoir allonger le chemin au fur et à mesure des dépliages successifs.

L'arbre de preuve de la contrainte $code = PIN(carte)$, associé au deuxième jeu de tests obtenu, est le suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{code{=}c \wedge c{\neq}0 \Rightarrow [Code?d]\ \ code{=}c}\,\mathsf{Ax} \qquad \vdots}{code{=}PIN(carte) \wedge PIN(carte){\neq}0 \Rightarrow [Code?d_0]\ \ code{=}PIN(carte)}\,\mathsf{Subs} \qquad \dfrac{\vdots}{code{=}PIN(carte)}}{PIN(carte){\neq}0 \Rightarrow [Code?d_0]\ \ code{=}PIN(carte)}\,\mathsf{MP} \qquad \dfrac{\vdots}{PIN(carte){\neq}0}}{[Code?d_0]\ \ code{=}PIN(carte)}\,\mathsf{MP}$$

L'arbre de preuve complet après les deux dépliages consécutifs est alors le suivant :

$$\dfrac{\dfrac{\dfrac{}{code{=}PIN(carte) \Rightarrow [Consulte?][Solde!]\ \ écran{=}solde(carte)}\,\mathsf{Ax}}{[Code?d_0]\ \ code{=}PIN(carte) \Rightarrow [Code?d_0][Consulte?][Solde!]\ \ écran{=}solde(carte)}\,\mathsf{Nec} \qquad SA_2}{[Code?d_0][Consulte?][Solde!]\ \ écran{=}solde(carte)}\,\mathsf{MP}$$

où SA_2 est l'arbre précédent.

On constate ici que, comme dans le cadre du premier ordre, les différents arbres de preuve associés aux jeux de tests construits respectent une certaine structure. On trouve aux feuilles les instances d'axiomes, puis les instances de la règle de substitution, suivies de celles de la règle de nécessitation, pour finir par les instances de la règle de modus ponens. La procédure de dépliage va alors consister à construire, étape par étape, des arbres ayant cette structure, pour en dériver les jeux de tests.

En termes d'algorithme, la procédure prend alors en entrée :

- une spécification modale conditionnelle positive $Sp = (\Sigma, Ax)$;
- un attribut $f : s_1 \times \ldots \times s_n \times s \to s' \in F_s$;
- un ensemble Ψ de triplets $(\mathcal{C}, \sigma, \Delta)$ où $\mathcal{C}$ est un ensemble de Σ-contraintes, σ est une substitution des variables et Δ est une trace.

Les jeux de tests pour les attributs sont prolongés aux ensembles de triplets Ψ de la façon suivante :

$$T_{\Psi,f} = \bigcup_{(\mathcal{C},\sigma,\Delta)\in\Psi} T_{(\mathcal{C},\sigma,\Delta),f}$$

Le premier ensemble Ψ_0 contient l'unique triplet, composé d'un ensemble de contraintes, d'une substitution et d'une trace, associé à l'objectif de test contraint initial :

$$\Psi_0 = \{(\{f(x_1, \ldots, x_n) = y\}, \mathrm{Id}, \_)\}$$

Intuitivement, le principe de la procédure est de chercher à unifier une des contraintes à la conclusion d'un axiome de la spécification, plus exactement à une partie de la conclusion de cet axiome. Si on dispose d'un axiome de la forme $\varphi_1 \wedge \ldots \wedge \varphi_m \Rightarrow [\gamma_1] \ldots [\gamma_p]u = v$, et d'une contrainte $[\beta_1] \ldots [\beta_k]t = r$, on cherche à unifier la contrainte avec la partie $[\gamma_{p-k+1}] \ldots [\gamma_p]u = v$ de la conclusion de l'axiome, c'est-à-dire avec la formule constituée des k dernières modalités et de l'équation. On construit alors le jeu de tests correspondant en ajoutant $\varphi_1, \ldots, \varphi_m$ à l'ensemble de contraintes, en allongeant le chemin des termes $\gamma_1 \ldots \gamma_{p-k}$, et en composant la substitution ayant permis l'unification avec la substitution existante. En effet, si la contrainte a pu être unifiée avec la partie de l'axiome correspondante, il existe une preuve de la contrainte, précédée du chemin $\gamma_1 \ldots \gamma_{p-k}$, à partir de l'axiome et des preuves de chacune des prémisses de l'axiome.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi_1 \wedge \ldots \wedge \varphi_m \Rightarrow [\gamma_1] \ldots [\gamma_p]u = v}\,\mathsf{Ax}}{\sigma(\varphi_1) \wedge \ldots \wedge \sigma(\varphi_m) \Rightarrow [\sigma(\gamma_1)] \ldots [\sigma(\gamma_p)]\sigma(u) = \sigma(v)}\,\mathsf{Subs} \quad \dfrac{\vdots}{\sigma(\varphi_1)}}{\sigma(\varphi_2) \wedge \ldots \wedge \sigma(\varphi_m) \Rightarrow [\sigma(\gamma_1)] \ldots [\sigma(\gamma_p)]\sigma(u) = \sigma(v)}\,\mathsf{MP}}{\vdots}\,\mathsf{MP}}{\sigma(\varphi_m) \Rightarrow [\sigma(\gamma_1)] \ldots [\sigma(\gamma_p)]\sigma(u) = \sigma(v)}\,\mathsf{MP} \quad \dfrac{\vdots}{\sigma(\varphi_m)}}{[\sigma(\gamma_1)] \ldots [\sigma(\gamma_p)]\sigma(u) = \sigma(v)}\,\mathsf{MP}$$

La conclusion $[\sigma(\gamma_1)] \ldots [\sigma(\gamma_p)]\sigma(u) = \sigma(v)$ est bien égale à l'instance sans variables de la contrainte précédée du chemin $\gamma_1 \ldots \gamma_{p-k}$ clos :

$$[\sigma(\gamma_1)] \ldots [\sigma(\gamma_{p-k})][\sigma(\gamma_{p-k+1})] \ldots [\sigma(\gamma_p)]\sigma(t) = \sigma(r)$$

La procédure de dépliage s'exprime alors au travers des deux règles suivantes :

$$\textbf{Réduction}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{[\beta_1]\dots[\beta_k]t = r\}, \sigma', \Delta)\}}{\Psi \cup \{(\sigma(\mathcal{C}), \sigma \circ \sigma', \Delta)\}} \quad \sigma(t) = \sigma(r)$$

$$\textbf{Dépliage}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\phi\}, \sigma', \Delta')\}}{\Psi \cup \bigcup_{(c,\sigma,\Delta) \in Dep(\phi)} \{(\mathcal{C}' \cup c, \sigma \circ \sigma', \Delta.\Delta')\}}$$

où $Dep(\phi)$ pour $\phi = [\beta_1]\dots[\beta_k]t = r$ est l'ensemble défini par :

$$\left\{ \left(\left\{ \begin{array}{c} [\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t[v]_\omega) = \sigma(r), \\ \sigma(\varphi_1), \dots, \sigma(\varphi_m) \end{array} \right\}, \sigma, \sigma(\gamma_1)\dots\sigma(\gamma_{p-k}) \right) \middle| \begin{array}{c} t_{|_\omega} = g(v_1, \dots, v_n) \\ \sigma(t_{|_\omega}) = \sigma(g(v_1, \dots, v_n)), \\ p \geq k, \forall l, 1 \leq l \leq k, \sigma(\beta_l) = \sigma(\gamma_{(p-k)+l}) \\ \Big(\bigwedge_{1 \leq i \leq m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]g(v_1, \dots, v_n) = v \in Ax \\ \text{ou} \\ \bigwedge_{1 \leq i \leq m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]v = g(v_1, \dots, v_n) \in Ax \Big) \end{array} \right\}$$

$$\bigcup$$

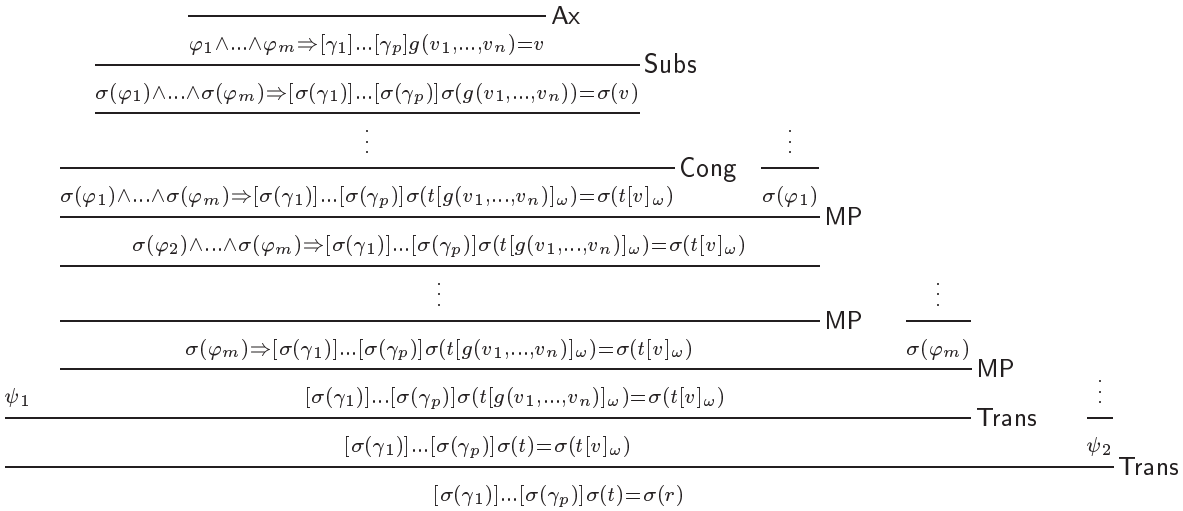
$$\left\{ \left(\left\{ \begin{array}{c} [\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t) = \sigma(r[v]_\omega), \\ \sigma(\varphi_1), \dots, \sigma(\varphi_m) \end{array} \right\}, \sigma, \sigma(\gamma_1)\dots\sigma(\gamma_{p-k}) \right) \middle| \begin{array}{c} r_{|_\omega} = g(v_1, \dots, v_n) \\ \sigma(r_{|_\omega}) = \sigma(g(v_1, \dots, v_n)), \\ p \geq k, \forall l, 1 \leq l \leq k, \sigma(\beta_l) = \sigma(\gamma_{(p-k)+l}) \\ \Big(\bigwedge_{1 \leq i \leq m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]g(v_1, \dots, v_n) = v \in Ax \\ \text{ou} \\ \bigwedge_{1 \leq i \leq m} \varphi_i \Rightarrow [\gamma_1]\dots[\gamma_p]v = g(v_1, \dots, v_n) \in Ax \Big) \end{array} \right\}$$

et pour tout $(c, \sigma, \Delta) \in Dep(\phi)$, l'ensemble de contraintes $\mathcal{C}'$ est l'ensemble de contraintes $\mathcal{C}$ dont toutes les contraintes sont précédées du chemin $\sigma(\Delta)$:

$$\mathcal{C}' = \{[\sigma(\gamma_1)]\dots[\sigma(\gamma_{p-k})]\sigma(\varepsilon) \mid \varepsilon \in \mathcal{C}\}$$

La règle de réduction permet d'éliminer les tautologies des ensembles de contraintes, qui ne sont d'aucune utilité pour le test. La règle de dépliage cherche à unifier un sous-terme d'un des deux termes apparaissant dans l'équation d'une contrainte, ainsi que les modalités qui précédent cette équation, avec la partie correspondante de la conclusion d'un axiome de la spécification. Si l'unification est possible avec un axiome, les prémisses de cet axiome sont ajoutées à l'ensemble des contraintes du jeu de tests courant (ainsi qu'un lemme

intermédiaire permettant de mener la preuve), le chemin correspondant à la partie des modalités qui n'a pas pu être unifiée est ajouté en tête du chemin courant, et la substitution qui permet l'unification est composée avec la substitution courante. Par exemple, si un sous-terme de t a pu être unifié avec le membre gauche de la conclusion d'un axiome, l'arbre suivant est la preuve de $[\rho\sigma(\gamma_1)]\dots[\rho\sigma(\gamma_p)]\rho\sigma(t) = \rho\sigma(r)$.



où ψ_1 est la formule $[\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t) = \sigma(t[g(v_1,\dots,v_n)]_\omega)$ et ψ_2 est la formule $[\sigma(\gamma_1)]\dots[\sigma(\gamma_p)]\sigma(t[v]_\omega) = \sigma(r)$.

On écrit $(\Psi, f) \vdash_D (\Psi', f)$ si Ψ' a pu être dérivé de Ψ en appliquant la règle **Réduction** ou **Dépliage**. La procédure de dépliage prend donc en entrée une spécification modale conditionnelle positive et un attribut f, et applique successivement les règles **Réduction** et **Dépliage** pour produire la suite de dérivations suivante :

$$(\Psi_0, f) \vdash_D (\Psi_1, f) \vdash_D (\Psi_2, f)\dots$$

Nous prouvons maintenant les deux propriétés qui assurent la pertinence de la méthode de dépliage pour des spécifications modales conditionnelles positives, la correction et la complétude. Nous posons de nouveau l'hypothèse selon laquelle toutes les variables utilisées sont différentes : pour tout Ψ résultant de la procédure de dépliage, pour tout $(\mathcal{C}, \sigma, \Delta) \in \Psi$, pour toute contrainte $c \in \mathcal{C}$ et pour tout axiome $\varphi \in Ax$, $Var(c) \cap Var(\varphi) = \emptyset$. Le renommage des variables apparaissant dans les contraintes à chaque étape assure le respect de cette condition.

THÉORÈME VII.2. *Si* $(\Psi, f) \vdash_D (\Psi', f)$, *alors* $T_{\Psi,f} = T_{\Psi',f}$.

Preuve.
(Correction) Montrons que si $(\Psi, f) \vdash_D (\Psi', f)$, alors $T_{\Psi',f} \subseteq T_{\Psi,f}$.

Si la règle appliquée est **Réduction**, c'est trivial. S'il s'agit de la règle de **Dépliage**, par définition, on doit montrer que pour tout $(c, \sigma, \Delta) \in Dep(\phi)$, où ϕ est la formule $[\beta_1]\dots[\beta_k]t = r$,

$$T_{(\mathcal{C}' \cup c, \sigma\circ\sigma', \Delta.\Delta')} \subseteq T_{(\mathcal{C}\cup\{\phi\}, \sigma', \Delta')}$$

On doit alors montrer que pour chaque substitution close $\rho : V \to T_\Sigma$ telle que $Sp \models \rho(\xi')$ pour tout $\xi' \in \sigma(\mathcal{C}') \cup c$, il existe $\rho' : V \to T_\Sigma$ telle que $Sp \models [\rho'(\gamma_1)]\dots[\rho'(\gamma_{p-k})]\rho'(\xi)$ pour tout $\xi \in \mathcal{C} \cup \{\phi\}$.

Sans perte de généralité, on suppose que pour tout $\phi = [\beta_1] \dots [\beta_k] t = r \in \mathcal{C}$, chaque c tel que $(c, \sigma, \Delta) \in Dep(\phi)$ est de la forme :

$$\{[\sigma(\gamma_1)] \dots [\sigma(\gamma_p)]\sigma(t[v]_\omega) = \sigma(r), \sigma(\varphi_1), \dots, \sigma(\varphi_m)\}$$

avec $t_{|_\omega} = g(v_1, \dots, v_n)$ et $\bigwedge_{1 \leq i \leq m} \varphi_i \Rightarrow [\gamma_1] \dots [\gamma_p] g(v_1, \dots, v_n) = v \in Ax$. Soit $\rho : V \to T_\Sigma$ une substitution close telle que $Sp \models \rho(\xi')$, pour tout $\xi' \in \mathcal{C}' \cup c$, c'est-à-dire, en particulier, $Sp \models \rho([\sigma(\gamma_1)] \dots [\sigma(\gamma_p)]\sigma(t[v]_\omega) = \sigma(r))$ et $Sp \models \rho\sigma(\varphi_i)$, pour tout i, $1 \leq i \leq m$.

On a alors l'arbre de preuve suivant, dans lequel on note $\rho\sigma$ la composition $\rho \circ \sigma$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi_1 \wedge \dots \wedge \varphi_m \Rightarrow [\gamma_1] \dots [\gamma_p] g(v_1, \dots, v_n) = v}\,\mathsf{Ax}}{\rho\sigma(\varphi_1) \wedge \dots \wedge \rho\sigma(\varphi_m) \Rightarrow [\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(g(v_1, \dots, v_n)) = \rho\sigma(v)}\,\mathsf{Subs}}{\vdots}\,\mathsf{Cong}}{\rho\sigma(\varphi_1) \wedge \dots \wedge \rho\sigma(\varphi_m) \Rightarrow [\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t[g(v_1, \dots, v_n)]_\omega) = \rho\sigma(t[v]_\omega)}\,\mathsf{MP} \quad \dfrac{\vdots}{\rho\sigma(\varphi_1)}}{\rho\sigma(\varphi_2) \wedge \dots \wedge \rho\sigma(\varphi_m) \Rightarrow [\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t[g(v_1, \dots, v_n)]_\omega) = \rho\sigma(t[v]_\omega)}\,\mathsf{MP} \quad \dfrac{\vdots}{\rho\sigma(\varphi_m)}}{\rho\sigma(\varphi_m) \Rightarrow [\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t[g(v_1, \dots, v_n)]_\omega) = \rho\sigma(t[v]_\omega)}\,\mathsf{MP}$$

$$\dfrac{\dfrac{SA \qquad [\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t[g(v_1, \dots, v_n)]_\omega) = \rho\sigma(t[v]_\omega)}{[\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t) = \rho\sigma(t[v]_\omega)}\,\mathsf{Trans} \quad \dfrac{\vdots}{\psi_2}}{[\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t) = \rho\sigma(r)}\,\mathsf{Trans}$$

où SA est le sous-arbre suivant :

$$\dfrac{\dfrac{\dfrac{\sigma(t_{|_\omega}) = \sigma(g(v_1, \dots, v_n))}{\rho\sigma(t) = \rho\sigma(g(v_1, \dots, v_n))}\,\mathsf{Subs}}{\vdots}\,\mathsf{Cong}}{[\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t) = \rho\sigma(t[g(v_1, \dots, v_n)]_\omega)}\,\mathsf{Nec}$$

De plus, comme pour tout l, $1 \leq l \leq k$, on a $\sigma(\beta_l) = \sigma(\gamma_{(p-k)+l})$, alors $\rho\sigma(\beta_l) = \rho\sigma(\gamma_{(p-k)+l})$, et la conclusion $[\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_p)] \rho\sigma(t) = \rho\sigma(r)$ de l'arbre de preuve est égale à :

$$[\rho\sigma(\gamma_1)] \dots [\rho\sigma(\gamma_{p-k})] [\rho\sigma(\beta_1)] \dots [\rho\sigma(\beta_k)] \rho\sigma(t) = \rho\sigma(r)$$

(Complétude) Montrons que si $(\Psi, f) \vdash_D (\Psi', f)$, alors $T_{\Psi, f} \subseteq T_{\Psi', f}$.

Par définition de la règle **Dépliage**, on doit montrer que

$$T_{(\mathcal{C} \cup \{\phi\}, \sigma', \Delta'), f} \subseteq \bigcup_{(c, \sigma, \Delta) \in Dep(\phi)} T_{(\mathcal{C}' \cup c, \sigma \circ \sigma', \Delta . \Delta')}$$

On doit alors montrer que pour chaque substitution close $\rho : V \to T_\Sigma$ telle que $Sp \vdash [\rho(\gamma_1)] \dots [\rho(\gamma_{p-k})] \rho(\xi)$ pour tout $\xi \in \mathcal{C} \cup \{\phi\}$, il existe $(c, \sigma, \Delta) \in Dep(\phi)$ tel qu'il existe $\rho' : V \to T_\Sigma$

telle que $Sp \vdash \rho'(\xi')$ pour tout $\xi' \in \mathcal{C}' \cup c$. En d'autres termes, on doit prouver que chaque $[\rho(\gamma_1)] \dots [\rho(\gamma_{p-k})]\rho(\xi)$ où $\rho : V \to T_\Sigma$, $\xi \in \mathcal{C} \cup \{\phi\}$ et $\gamma_1, \dots, \gamma_{p-k} \in T_{\Sigma_s}$ où $s \in T$, peut être déduit de la spécification Sp s'il existe $(c, \sigma, \Delta) \in Dep(\phi)$ et $\rho' : V \to T_\Sigma$ tels que $Sp \vdash \rho'(\xi')$ pour tout $\xi' \in \mathcal{C}' \cup c$.

Remarquons tout d'abord que la procédure de dépliage définit une stratégie qui limite l'espace de recherche d'arbres de preuve à une classe d'arbres ayant une certaine structure. La procédure de dépliage définit une stratégie de recherche de preuve qui sélectionne des arbres de preuve tels que :

- aucune instance de la règle de transitivité n'apparaît au-dessus d'instances des règles de congruence, de nécessitation, de substitution et de modus ponens ;
- aucune instance de la règle de modus ponens n'apparaît au-dessus d'instances des règles de congruence, de nécessitation et de substitution ;
- aucune instance des règles de congruence et de nécessitation n'apparaissent au-dessus d'instances de la règle de substitution.

On doit alors montrer qu'il existe un arbre de preuve de conclusion $[\rho(\gamma_1)] \dots [\rho(\gamma_{p-k})]\rho(\xi)$ respectant cette structure. On va en fait montrer un résultat plus fort : on va définir des transformations élémentaires d'arbres de preuve, qui permettent de réécrire des combinaisons élémentaires de règles d'inférence, nous allons ensuite montrer que la transformation globale résultante termine. On donne ici quelques exemples de ces transformations élémentaires. Les autres combinaisons de règles suivent des transformations similaires.

Le cas où une instance du modus-ponens se trouve au-dessus d'une instance de la règle de nécessitation :

$$\dfrac{\dfrac{\bigwedge_{i\le n} \varphi_i \wedge \psi \Rightarrow \varphi \qquad \bigwedge_{i\le n} \varphi_i \Rightarrow \psi}{\bigwedge_{i\le n} \varphi_i \Rightarrow \varphi}\,\mathsf{MP}}{\bigwedge_{i\le n} [\alpha]\varphi_i \Rightarrow [\alpha]\varphi}\,\mathsf{Nec} \quad \rightsquigarrow \quad \dfrac{\dfrac{\bigwedge_{i\le n} \varphi_i \wedge \psi \Rightarrow \varphi}{\bigwedge_{i\le n} [\alpha]\varphi_i \wedge [\alpha]\psi \Rightarrow [\alpha]\varphi}\,\mathsf{Nec} \qquad \dfrac{\bigwedge_{i\le n} \varphi_i \Rightarrow \psi}{\bigwedge_{i\le n} [\alpha]\varphi_i \Rightarrow [\alpha]\psi}\,\mathsf{Nec}}{\bigwedge_{i\le n} [\alpha]\varphi_i \Rightarrow [\alpha]\varphi}\,\mathsf{MP}$$

Le cas où une instance de la règle de nécessitation se trouve au-dessus d'une instance de la règle de substitution :

$$\dfrac{\dfrac{\bigwedge_{i\le n} \varphi_i \Rightarrow \varphi}{\bigwedge_{i\le n} [\alpha]\varphi_i \Rightarrow [\alpha]\varphi}\,\mathsf{Nec}}{\bigwedge_{i\le n} [\sigma(\alpha)]\sigma(\varphi_i) \Rightarrow [\sigma(\alpha)]\sigma(\varphi)}\,\mathsf{Subs} \quad \rightsquigarrow \quad \dfrac{\dfrac{\bigwedge_{i\le n} \varphi_i \Rightarrow \varphi}{\bigwedge_{i\le n} \sigma(\varphi_i) \Rightarrow \sigma(\varphi)}\,\mathsf{Subs}}{\bigwedge_{i\le n} [\sigma(\alpha)]\sigma(\varphi_i) \Rightarrow [\sigma(\alpha)]\sigma(\varphi)}\,\mathsf{Nec}$$

Le cas où une instance de la règle de monotonie se trouve au-dessus d'une instance de la règle de nécessitation :

$$\dfrac{\dfrac{\bigwedge_{i\le n} \varphi_i \Rightarrow \varphi}{\bigwedge_{i\le n} \varphi_i \wedge \psi \Rightarrow \varphi}\,\mathsf{Mono}}{\bigwedge_{i\le n} [\alpha]\varphi_i \wedge [\alpha]\psi \Rightarrow [\alpha]\varphi}\,\mathsf{Nec} \quad \rightsquigarrow \quad \dfrac{\dfrac{\bigwedge_{i\le n} \varphi_i \Rightarrow \varphi}{\bigwedge_{i\le n} [\alpha]\varphi_i \Rightarrow [\alpha]\varphi}\,\mathsf{Nec}}{\bigwedge_{i\le n} [\alpha]\varphi_i \wedge [\alpha]\psi \Rightarrow [\alpha]\varphi}\,\mathsf{Mono}$$

Le cas où une instance de la règle de modus ponens apparaît au-dessus d'une instance de la règle de monotonie :

$$\dfrac{\dfrac{\bigwedge_{i\leq n}\varphi_i\wedge\psi\Rightarrow\varphi \qquad \bigwedge_{i\leq n}\varphi_i\Rightarrow\psi}{\bigwedge_{i\leq n}\varphi_i\Rightarrow\varphi}\,\mathsf{MP}}{\chi\wedge\bigwedge_{i\leq n}\varphi_i\Rightarrow\varphi}\,\mathsf{Mono} \quad\rightsquigarrow\quad \dfrac{\dfrac{\bigwedge_{i\leq n}\varphi_i\wedge\psi\Rightarrow\varphi \qquad \bigwedge_{i\leq n}\varphi_i\Rightarrow\psi}{\chi\wedge\bigwedge_{i\leq n}\varphi_i\wedge\psi\Rightarrow\varphi\chi\wedge\bigwedge_{i\leq n}\varphi_i\Rightarrow\psi}\,\mathsf{Mono}}{\chi\wedge\bigwedge_{i\leq n}\varphi_i\Rightarrow\varphi}\,\mathsf{MP}$$

Considérons l'ordre bien-fondé suivant sur les instances de règles d'inférence :

$$\mathsf{Sym}\sim\mathsf{Subs}\succ\mathsf{Cong}\sim\mathsf{Nec}\succ\mathsf{Mono}\succ\mathsf{MP}\succ\mathsf{Trans}$$

Il est alors facile de voir que ces règles de transformation vérifient la condition 1 de l'annexe B. D'après le théorème B.1, la transformation globale induite est donc fortement normalisante.

Puisque par hypothèse, $Sp \models [\rho'(\gamma_1)]\dots[\rho'(\gamma_{p-k})][\rho'(\beta_1)]\dots[\rho'(\beta_k)]\rho'(t) = \rho'(r)$, où $[\beta_1]\dots[\beta_k]t = r \in \mathcal{C}$, comme $[\rho'(\gamma_1)]\dots[\rho'(\gamma_{p-k})][\rho'(\beta_1)]\dots[\rho'(\beta_k)]\rho'(t) = \rho'(r)$ n'est pas une tautologie, il existe nécessairement un axiome $\bigwedge_{i\leq m}\alpha_i \Rightarrow [\gamma_1]\dots[\gamma_p]g(v_1,\dots,v_n) = v$, une position ω dans $\rho'(t)$ ou $\rho'(r)$ et une substitution close ρ telle que $\rho'(t)_{|\omega} = \rho'(g(v_1,\dots,v_n))$ ou $\rho'(r)_{|\omega} = \rho'(g(v_1,\dots,v_n))$, et telle que $\rho'(\beta_l) = \rho'(\gamma_{(p-k)+l})$ pour tout l, $1 \leq l \leq k$. La substitution ρ' est alors un unificateur de $t_{|\omega}$ ou $r_{|\omega}$ et de $g(v_1,\dots,v_n)$, et de chaque β_l et $\gamma_{(p-k)+l}$. Il existe donc un arbre de preuve résultant de la transformation globale définie plus haut, de conclusion $[\rho'(\gamma_1)]\dots[\rho'(\gamma_{p-k})][\rho'(\beta_1)]\dots[\rho'(\beta_k)]\rho'(t) = \rho'(r)$, et de la forme :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi_1\wedge\dots\wedge\varphi_m\Rightarrow[\gamma_1]\dots[\gamma_p]g(v_1,\dots,v_n)=v}\,\mathsf{Ax}}{\rho'(\varphi_1)\wedge\dots\wedge\rho'(\varphi_m)\Rightarrow[\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(g(v_1,\dots,v_n))=\rho'(v)}\,\mathsf{Subs}}{\vdots}}{\rho'(\varphi_1)\wedge\dots\wedge\rho'(\varphi_m)\Rightarrow[\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(t)=\rho'(t[v]_\omega)}\,\mathsf{Cong} \quad \dfrac{\vdots}{\rho'(\varphi_1)}}{\rho'(\varphi_2)\wedge\dots\wedge\rho'(\varphi_m)\Rightarrow[\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(t)=\rho'(t[v]_\omega)}\,\mathsf{MP}}{\vdots}}{\rho'(\varphi_m)\Rightarrow[\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(t)=\rho'(t[v]_\omega)}\,\mathsf{MP} \quad \dfrac{\vdots}{\rho'(\varphi_m)}}{[\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(t)=\rho'(t[v]_\omega)}\,\mathsf{MP} \quad \dfrac{\vdots}{\psi}}{[\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(t)=\rho'(r)}\,\mathsf{Trans}$$

où $\psi = [\rho'(\gamma_1)]\dots[\rho'(\gamma_p)]\rho'(t[v]_\omega) = \rho'(r)$. □

CHAPITRE VIII

Sélection à partir de spécifications modales sans quantificateurs

L'adaptation du cadre de sélection de tests aux spécifications modales conditionnelles positives ne constitue qu'une étape vers la généralisation aux spécifications modales telles que permet de les exprimer la logique sous-jacente au langage de spécifications COCASL. Nous présentons alors dans ce chapitre la généralisation à la fois des résultats d'exhaustivité et de la procédure de dépliage à des spécifications modales du premier ordre sans quantificateurs. En ce qui concerne la procédure de dépliage cependant, cette généralisation est plus proche de la procédure généralisée aux spécifications du premier ordre sans quantificateurs que de celle présentée au chapitre précédent. En effet, la forme générale des formules, une fois transformées en séquents et normalisées de la façon que nous verrons, est similaire à celle des séquents normalisés manipulés dans le cadre du premier ordre sans quantificateurs. Étendre simplement la procédure définie pour les spécifications modales conditionnelles positives aurait été de fait trop restrictif, la forme de ces formules étant très particulière et induisant un type de dépliage spécifique.

De nouveau, la complétude du calcul des séquents n'ayant pas été montrée, nous montrons l'exhaustivité de l'ensemble des conséquences sémantiques observables de la spécification mais nous procédons à la sélection à partir de l'ensemble des théorèmes observables, celui-ci étant inclus dans le jeu de tests exhaustif.

1 Normalisation des séquents

Afin de manipuler des formules aussi simples que possibles au cours de la procédure de dépliage, nous allons de nouveau tirer avantage du fait que les règles associées aux connecteurs booléens dans le calcul des séquents sont inversibles. Cela permet en effet de définir un processus qui transforme tout séquent en un séquent normalisé. Dans le cadre de la logique du premier ordre sans quantificateurs, nous avions défini un séquent normalisé comme étant un séquent dans lequel toute formule est atomique, c'est-à-dire de la forme $r(t_1, \ldots, t_n)$ avec r un prédicat et $t_1, \ldots, t_n$ des termes du premier ordre. Il était possible de transformer tout séquent en un séquent de cette forme car toutes les règles associées aux connecteurs booléens pouvaient « descendre » dans l'arbre de preuve en-dessous de toutes les autres règles, ce qui permettaient de laisser pour la fin de la preuve l'application de ces règles, et donc de ne travailler qu'avec des séquents normalisés.

Ici cependant, la règle de nécessitation complique le processus de normalisation des séquents. En effet, cette règle n'est pas inversible, il n'est donc pas envisageable de chercher à l'éliminer de la procédure de

dépliage comme les règles booléennes. Mais, contrairement aux règles de substitution et de coupure, elle ne remonte sur aucune des règles associées aux connecteurs booléens. Tout au plus annule-t-elle la règle associée à la négation. La notion de séquent normalisé doit alors être modifiée de façon à prendre en compte le fait que la nécessitation peut être appliquée après l'introduction de connecteurs booléens.

DÉFINITION VIII.1 — *Séquent normalisé.*

Un *séquent normalisé* est un séquent $\Gamma \mid\sim \Delta$ tel que pour toute formule $\varphi \in \Gamma \cup \Delta$, φ est de la forme $\alpha_1 \dots \alpha_m \psi$ où $\alpha_1, \dots, \alpha_m$ sont des modalités et ψ est une formule modale sans quantificateurs qui ne commence pas par une modalité.

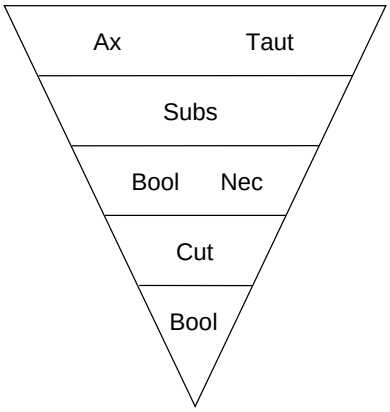
Dans une formule d'un séquent normalisé, la formule apparaissant derrière les modalités doit avoir pour opérateur de tête un connecteur booléen, mais peut être de n'importe quelle forme. Par exemple $\mid\sim [t] \langle u* \rangle (\varphi \Rightarrow [w*]\psi)$ est un séquent normalisé. Il est alors possible de transformer tout séquent en une famille de séquents normalisés, en éliminant tous les connecteurs booléens qui ne sont pas sous la portée d'une modalité. On donne ci-dessous l'exemple de la normalisation du séquent $\vdash [t] \langle u* \rangle (\varphi \Rightarrow [w*]\psi) \Rightarrow \neg[\{t,u\}]\varphi \vee [w]\varphi \vee (\varphi \wedge \neg\psi)$.

$$\dfrac{\dfrac{\dfrac{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi),[\{t,u\}]\varphi \vdash}{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi) \vdash \neg[\{t,u\}]\varphi}\neg\text{-droit} \quad \dfrac{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi) \vdash [w]\varphi \quad \dfrac{\dfrac{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi),\psi \vdash \varphi}{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi) \vdash \varphi,\neg\psi}\neg\text{-droit}}{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi) \vdash \varphi\wedge\neg\psi}\wedge\text{-droit}}{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi) \vdash [w]\varphi\vee(\varphi\wedge\neg\psi)}\vee\text{-droit}}{[t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi) \vdash \neg[\{t,u\}]\varphi\vee[w]\varphi\vee(\varphi\wedge\neg\psi)}\vee\text{-droit}}{\vdash [t]\langle u*\rangle(\varphi\Rightarrow[w*]\psi)\Rightarrow\neg[\{t,u\}]\varphi\vee[w]\varphi\vee(\varphi\wedge\neg\psi)}\Rightarrow\text{-droit}$$

On obtient alors la famille de séquents normalisés suivante :

$$\{[t]\langle u*\rangle(\varphi \Rightarrow [w*]\psi), [\{t,u\}]\varphi \vdash , [t]\langle u*\rangle(\varphi \Rightarrow [w*]\psi) \vdash [w]\varphi, [t]\langle u*\rangle(\varphi \Rightarrow [w*]\psi), \psi \vdash \varphi\}$$

De façon à s'assurer que cette transformation des séquents n'affectera pas la procédure de dépliage, il faut montrer que toute preuve peut être menée en deux temps : ne manipuler que des séquents normalisés d'abord, puis ne consister qu'en l'application des règles booléennes ensuite. Chercher la preuve d'un séquent quelconque ou les preuves des séquents normalisés qui lui sont équivalents reviendra alors au même. Nous allons en fait montrer que tout arbre de preuve peut être transformé en un arbre de même conclusion et de mêmes feuilles dans lequel les instances de règles associées aux connecteurs booléens se trouvent soit en-dessous des instances de la règle de coupure, soit au même niveau que les instances de la règle de nécessitation. Schématiquement, un tel arbre admet la forme suivante :



Il faut pour cela poser la même hypothèse que dans le cadre de la logique du premier ordre : les axiomes de la spécification introduits par la règle Ax et les tautologies doivent être sous la forme de séquents normalisés. De cette façon, il ne sera manipulé que des séquents normalisés en haut de l'arbre de preuve.

La transformation d'un séquent quelconque en une famille de séquents normalisés s'opère par une suite de transformations élémentaires d'arbres de preuve qui termine (voir l'annexe B), dont nous allons donner quelques exemples. Les règles de transformation faisant remonter la coupure et la substitution ayant déjà été données au chapitre VI, nous ne présentons ici que les règles faisant intervenir la nécessitation. Tout d'abord, la règle de substitution remonte sur la règle de nécessitation de la façon suivante :

$$\dfrac{\dfrac{\Gamma \mid\sim \Delta}{[t]\Gamma \mid\sim [t]\Delta}\,\mathsf{Nec}}{[\sigma(t)]\sigma(\Gamma) \mid\sim [\sigma(t)]\sigma(\Delta)}\,\mathsf{Subs} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \mid\sim \Delta}{\sigma(\Gamma) \mid\sim \sigma(\Delta)}\,\mathsf{Subs}}{[\sigma(t)]\sigma(\Gamma) \mid\sim [\sigma(t)]\sigma(\Delta)}\,\mathsf{Nec}$$

La règle de transformation qui permet de faire remonter la règle de nécessitation sur la règle de coupure est la suivante :

$$\dfrac{\dfrac{\Gamma \mid\sim \varphi, \Delta \qquad \Gamma', \varphi \mid\sim \Delta'}{\Gamma, \Gamma' \mid\sim \Delta, \Delta'}\,\mathsf{Cut}}{[t]\Gamma, [t]\Gamma' \mid\sim [t]\Delta, [t]\Delta'}\,\mathsf{Nec} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \mid\sim \varphi, \Delta}{[t]\Gamma \mid\sim [t]\varphi, [t]\Delta}\,\mathsf{Nec} \qquad \dfrac{\Gamma', \varphi \mid\sim \Delta'}{[t]\Gamma', [t]\varphi \mid\sim [t]\Delta'}\,\mathsf{Nec}}{[t]\Gamma, [t]\Gamma' \mid\sim [t]\Delta, [t]\Delta'}\,\mathsf{Cut}$$

Celles qui apparaissent en-dessous, si elles sont au-dessus de la règle de coupure, descendent, et se retrouvent ainsi toutes sous la règle de coupure. L'arbre de preuve final a bien alors la forme voulue, donnée ci-dessus. Plus formellement, toutes les règles de transformations élémentaires vérifient la condition 1 donnée dans l'annexe B, pour l'ordre bien-fondé suivant sur les instances de règles d'inférence :

$$\mathsf{Ax} \sim \mathsf{Taut} \succ \mathsf{Subs} \succ \mathsf{Bool} \sim \mathsf{Nec} \succ \mathsf{Cut}$$

Par conséquent, d'après le théorème B.1, la transformation globale induite est donc fortement normalisante. Tout arbre de preuve pouvant prendre la forme voulue, raisonner sur des séquents normalisés ne restreint pas les capacités d'application de la procédure de dépliage que nous allons présenter par la suite. Nous ne considérerons donc dorénavant que des séquents normalisés.

EXEMPLE VIII.1 — *Listes infinies.*

On transforme la spécification des listes infinies donnée à l'exemple III.19 en séquents normalisés. Chacun des axiomes étant une équivalence, on obtient deux formules, une pour chaque implication. Ensuite, chaque implication est transformée de façon triviale en séquent normalisé grâce à la règle $\Rightarrow$-droit.

1. $tête = n \mid\sim [impair]\ tête = n$
2. $[impair]\ tête = n \mid\sim\ tête = n$
3. $[impair][queue]\varphi \mid\sim [queue][queue][impair]\varphi$
4. $[queue][queue][impair]\varphi \mid\sim [impair][queue]\varphi$
5. $[pair]\varphi \mid\sim [queue][impair]\varphi$
6. $[queue][impair]\varphi \mid\sim [pair]\varphi$

◇

2 Exhaustivité

Il s'agit ici de définir la notion d'observabilité dans le cadre du test à partir de spécifications modales sans quantificateurs, afin de préciser le jeu de tests dont doit être montrée l'exhaustivité. Ici, comme dans le cadre du test à partir de spécifications du premier ordre sans quantificateurs, sont considérées comme observables toutes les formules sans variables, quelle que soit leur forme. Il nous faut alors montrer que l'ensemble des conséquences sémantiques sans variables de la spécification est un jeu de tests exhaustif. Contrairement au chapitre précédent, mais de la même manière que dans le cadre du premier ordre au chapitre VI, aucune hypothèse forte sur le système sous test n'est nécessaire pour obtenir l'exhaustivité du jeu de tests $Sp^\bullet \cap Obs$. La seule restriction porte sur le modèle associé aux données, qui doit être finiment engendré. Cette restriction est mineure, cette propriété étant vérifiée la plupart du temps. En effet, il est naturel de considérer que les valeurs manipulées par le système sous test peuvent être dénotées par un terme clos construit sur la signature du système.

THÉORÈME VIII.1. *Soient $Sp = (\Sigma, Ax)$ une spécification modale sans quantificateurs. Le jeu de tests $Sp^\bullet \cap Obs$ est exhaustif pour tout modèle de $Mod(\Sigma)$ dont le modèle du premier ordre sous-jacent est finiment engendré.*

Preuve. Soit $\mathcal{S}$ un système sous test, i.e. $\mathcal{S} \in Mod(\Sigma)$. On suppose que $\mathcal{S} \models Sp^\bullet \cap Obs$. Montrons que $Correct_{obs}(\mathcal{S}, Sp)$.

Puisque $\mathcal{S} \in Mod(\Sigma)$, $\mathcal{S}$ est une $\mathcal{F}$-coalgèbre (E, α), construite au-dessus d'une structure du premier ordre $\mathcal{A}$, où $\mathcal{F}$ est le foncteur défini à la section 2.2. D'après le théorème III.5, $\mathcal{F}$ étant polynomial, $Mod(\Sigma) = \mathsf{Coalg}(\mathcal{F})$ admet une coalgèbre terminale.

Posons l'ensemble de formules modales sans variables $Th(\mathcal{S}) = \{\varphi \in Obs \mid \mathcal{S} \models \varphi\}$. Notons $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ la sous-catégorie pleine de $\mathsf{Coalg}(\mathcal{F})$ dont les objets sont les Σ-modèles qui valident $Th(\mathcal{S})$. Il est facile de vérifier que $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ est une covariété. D'après le théorème III.6, $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ admet un modèle terminal, que nous noterons $\mathcal{T}/_{Th(\mathcal{S})}$, qui est une sous-coalgèbre de $\mathcal{T}$. $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$ étant fermée par sous-coalgèbres, notons $\mathcal{T}/_{\mathcal{S}}$ la $\mathcal{F}$-coalgèbre (E', α') sur la structure du premier ordre $\mathcal{A}$, où $E' = h(E)$ et h est l'unique morphisme de $\mathcal{S}$ dans $\mathcal{T}/_{Th(\mathcal{S})}$. Par construction, nous avons $\mathcal{S} \equiv_{obs} \mathcal{T}/_{\mathcal{S}}$, puisque $\mathcal{S}$ et $\mathcal{T}/_{\mathcal{S}}$ sont dans $\mathsf{Coalg}(\mathcal{F})_{|_{Th(\mathcal{S})}}$. En fait, nous avons un résultat plus fort : le morphisme h

admet une factorisation $h = i \circ q$ où q est surjective de $\mathcal{S}$ dans $\mathcal{T}/_{\mathcal{S}}$. Puisque q est un morphisme, l'ensemble $\{(e, q(e)) \mid e \in E\}$ est une bisimulation, et donc, q est un morphisme élémentaire. On conclut alors que pour toute formule sans variables φ et pour tout état $e \in E$, $\mathcal{S} \models_e \varphi \Leftrightarrow \mathcal{T}/_{\mathcal{S}} \models_{q(e)} \varphi$.

Montrons maintenant que $\mathcal{T}/_{\mathcal{S}} \in Mod(Sp)$. Soit φ un axiome de Ax, soit $e' \in E'$ un état et $\nu' : V \to A$ une interprétation. Montrons que $\mathcal{T}/_{\mathcal{S}} \models_{\nu',e'} \varphi$. Comme le modèle du premier ordre sous-jacent à $\mathcal{S}$ est finiment engendré, pour tout état $e \in E$ et toute interprétation $\nu : V \to A$, $\mathcal{S} \models_{\nu,e} \varphi$. En particulier, c'est vrai pour tout $e \in q^{-1}(e')$ et pour tout $(\nu_e)_{e \in E}$ tels que pour tout $e' \in E'$, pour tout $e \in q^{-1}(e')$, $\nu_e = \nu'_{e'}$. Puisque $\mathcal{S}$ et $\mathcal{T}/_{\mathcal{S}}$ sont élémentairement équivalents sur les formules sans variables, et puisque le modèle du premier ordre sous-jacent à chacun d'entre eux est finiment engendré, nous avons $\mathcal{T}/_{\mathcal{S}} \models_{\nu',e'} \varphi$ et donc $\mathcal{T}/_{\mathcal{S}} \models \varphi$. Par conséquent, comme $\mathcal{T}/_{\mathcal{S}} \in Mod(Sp)$ et $\mathcal{S} \equiv_{obs} \mathcal{T}/_{\mathcal{S}}$, nous avons $Correct_{obs}(\mathcal{S}, Sp)$.

On suppose que $Correct_{obs}(\mathcal{S}, Sp)$, i.e. il existe $\mathcal{T} \in Mod(Sp)$ tel que $\mathcal{T} \equiv_{obs} \mathcal{S}$. Soit $\varphi \in Sp^\bullet \cap Obs$. Par hypothèse, $\mathcal{T} \models \varphi$, donc $\mathcal{S} \models \varphi$ également. Alors $\mathcal{S} \models Sp^\bullet \cap Obs$. □

3 Dépliage

Nous présentons ici une généralisation de la méthode de sélection de tests par dépliage des axiomes, que nous avons présentée tout au long de cette thèse, aux spécifications modales sans quantificateurs. Comme nous l'avons annoncé dans l'introduction de ce chapitre, la procédure que nous allons présenter est beaucoup plus une généralisation de la procédure dédiée aux spécifications du premier ordre sans quantificateurs présentée au chapitre VI que de celle, présentée au chapitre précédent, pour des spécifications modales conditionnelles positives. Nous verrons par la suite pour quelles raisons cette dernière procédure n'a pas trouvé de généralisation satisfaisante, et en quoi la procédure présentée ici permet quand même de recouvrir les mêmes jeux de tests.

Le but de la procédure de dépliage est de diviser le jeu de tests exhaustif de référence, $Sp^\bullet \cap Obs$. Découper ce jeu de tests revient à découper chacun des jeux de tests associé à une formule particulière prise comme objectif de test. Les sous-jeux de test obtenus par la division du jeu de tests associé à une formule sont des jeux de tests contraints par un ensemble de formules et une substitution. Les notions de jeu de tests pour une formule et de jeu de tests contraint étant définies de manière générale à la section 3 du chapitre VI, indépendamment de la forme des formules, nous reprenons ici ces définitions.

DÉFINITION VIII.2 — *Jeu de tests associé à une formule.*

Soient $Sp = (\Sigma, Ax)$ une spécification modale sans quantificateurs et $Obs \subseteq For(\Sigma)$ l'ensemble des formules observables. Soit φ une formule modale sans quantificateurs sur Σ, appelée *objectif de test*. Le *jeu de tests associé à* φ, noté T_φ, est l'ensemble défini par :

$$T_\varphi = \{\rho(\varphi) \mid \rho : V \to T_\Sigma, \rho(\varphi) \in Sp^\bullet \cap Obs\}$$

DÉFINITION VIII.3 — *Jeu de tests contraint.*

Soient $Sp = (\Sigma, Ax)$ une spécification modale sans quantificateurs et $Obs \subseteq For(\Sigma)$ l'ensemble des formules sans variables sur Σ. Soit φ une formule modale sans quantificateurs sur Σ. Soient $\mathcal{C} \subseteq For(\Sigma)$ un ensemble de formules appelées Σ-*contraintes* et $\sigma : V \to T_\Sigma(V)$ une substitution des variables. Le *jeu de*

tests associé à φ contraint par $\mathcal{C}$ et σ, noté $T_{(\mathcal{C},\sigma),\varphi}$, est l'ensemble de formules sans variables défini par :

$$T_{(\mathcal{C},\sigma),\varphi} = \{\rho(\sigma(\varphi)) \mid \rho : V \to T_\Sigma, \forall c \in \mathcal{C}, Sp \models \rho(c)\}$$

Le couple $((\mathcal{C}, \sigma), \varphi)$ est appelé *objectif de test contraint*.

L'objectif de test initial donné en argument à la procédure n'est pas contraint. La procédure va découper le jeu de tests associé à cet objectif de test en jeux de tests contraints, à l'aide des axiomes de la spécification. Elle prend en entrée :

- une spécification modale sans quantificateurs $Sp = (\Sigma, Ax)$ dont les axiomes ont été transformés en séquents normalisés ;
- une formule modale sans quantificateurs φ, mise sous la forme d'une famille de séquents normalisés, et vue comme l'objectif de test contraint initial $\langle(\{\varphi\}, \mathrm{Id}), \varphi\rangle$;
- une famille Ψ de couples $(\mathcal{C}, \sigma)$ où $\mathcal{C}$ est un ensemble de Σ-contraintes sous la forme de séquents normalisés, et σ est une substitution des variables $V \to T_\Sigma(V)$.

Les jeux de tests pour une formule sont naturellement étendus aux ensembles de couples Ψ de la façon suivante :

$$T_{\Psi,\varphi} = \bigcup_{(\mathcal{C},\sigma)\in\Psi} T_{(\mathcal{C},\sigma),\varphi}$$

Le premier ensemble Ψ_0 contient uniquement le couple composé de la famille de séquents normalisés obtenue à partir de l'objectif de test φ et de la substitution identité.

Le principe de la procédure est de chercher un arbre de preuve des différentes instances de l'objectif de test à partir des axiomes de la spécification. Dans le cadre de la sélection de tests à partir de spécifications du premier ordre sans quantificateurs, il s'agit d'unifier une partie de l'objectif de test sous la forme d'un séquent normalisé à une partie d'un axiome. On montre alors qu'il existe une preuve ayant cet axiome à une de ses feuilles et l'objectif de test comme conclusion. Le principe est le même ici à une exception près : les modalités. Dans le cadre du premier ordre, toutes les formules d'un séquent normalisé sont atomiques. Si une unification est possible entre deux formules atomiques, elle unifie nécessairement les formules dans leur totalité, celles-ci n'ayant pas de sous-formules. Les formules des séquents normalisés dans le cadre de la logique modale sont constituées d'une suite de modalités puis d'une formule quelconque ne commençant pas par une modalité. Une formule apparaissant dans un séquent normalisé peut donc être unifiée avec une sous-formule d'une formule apparaissant dans un autre séquent normalisé. On va donc chercher à unifier des sous-formules des formules composant l'objectif de test sous forme de séquent normalisé avec les formules du séquent normalisé correspondant à un axiome. La règle de nécessitation permettant intuitivement d'ajouter des modalités devant toutes les formules d'un séquent, on va pouvoir faire porter l'unification sur une sous-formule qui ne soit précédée que de modalités. Il suffira alors d'appliquer la règle de nécessitation suffisamment de fois pour ajouter les modalités manquantes. On schématise le processus global de la façon suivante :

$$\begin{array}{lcccc}
\text{Axiome :} & \psi_1,\ldots,\psi_p, & \xi_1,\ldots,\xi_l & \vdash\sim & \varphi_1,\ldots,\varphi_q, \quad \zeta_1,\ldots,\zeta_k \\
 & & \overbrace{}^{\text{à éliminer}} & & \overbrace{}^{\text{à éliminer}} \\
\text{Après Nec :} & \underbrace{M_1\psi_1,\ldots,M_p\psi_p}_{\text{unification}}, & \overbrace{M'_1\xi_1,\ldots,M'_l\xi_l} & \vdash\sim & \underbrace{N_1\varphi_1,\ldots,N_q\varphi_q}_{\text{unification}}, \overbrace{N'_1\zeta_1,\ldots,N'_k\zeta_k} \\
\text{Objectif de test :} & \overbrace{M_1\chi_1,\ldots,M_p\chi_p}, & \underbrace{\gamma_{p+1},\ldots,\gamma_m}_{\text{à ajouter}} & \vdash\sim & \overbrace{N_1\theta_1,\ldots,N_q\theta_q}, \quad \underbrace{\delta_{q+1},\ldots,\delta_n}_{\text{à ajouter}}
\end{array}$$

On tente donc d'unifier chacun des χ_i avec chacun des ψ_i pour tout i, $1 \leq i \leq p$ d'une part, et d'unifier chacun des θ_i avec chacun des φ_i pour tout i, $1 \leq i \leq q$ d'autre part. De façon à ce que l'ajout des modalités $M_1,\ldots,M_p,M'_1,\ldots,M'_l$ et $N_1,\ldots,N_q,N'_1,\ldots,N'_k$ soit effectivement possible grâce à une ou plusieurs applications de la règle de nécessitation, ces modalités doivent avoir une certaine forme, que précise la définition suivante.

DÉFINITION VIII.4 — *Relation $\mathcal{N}$ sur les modalités.*

Soient $p,q \in \mathbb{N}$. La relation $\mathcal{N} \subseteq (M_\Sigma(V)^*)^p \times (M_\Sigma(V)^*)^q$ est définie pour tous $(M_1,\ldots,M_p) \in (M_\Sigma(V)^*)^p$ et $(N_1,\ldots,N_q) \in (M_\Sigma(V)^*)^q$ de la façon suivante :

$(M_1,\ldots,M_p)\mathcal{N}(N_1,\ldots,N_q)$ si et seulement si

1. Il existe $n \in \mathbb{N}$ tel que pour tous i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, il existe $\alpha^i_1,\ldots,\alpha^i_n,\beta^j_1,\ldots,\beta^j_n \in M_\Sigma(V)$ tels que $M_i = \alpha^i_1\ldots\alpha^i_n$ et $N_j = \beta^j_1\ldots\beta^j_n$.
2. Pour tout i, $1 \leq l \leq n$, $\alpha^1_l,\ldots,\alpha^p_l$ et $\beta^1_l,\ldots,\beta^q_l$ sont tels que :
   (a) Il existe $t \in T_\Sigma(V)$ tel que pour tous i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, α^i_l et β^j_l sont tous égaux à $[t]$ ou $\langle t\rangle$, ou bien α^i_l et β^j_l sont tous égaux à $[t*]$ ou $\langle t*\rangle$.
   (b) Pour tous i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, $\alpha^i_l = [t]$ et $\beta^j_l = \langle t\rangle$ (resp. $\alpha^i_l = [t*]$ et $\beta^j_l = \langle t*\rangle$), sauf peut-être soit pour un k, $1 \leq k \leq p$, tel que $\alpha^k_l = \langle t\rangle$ (resp. $\alpha^k_l = \langle t*\rangle$), soit pour un k, $1 \leq k \leq q$, tel que $\beta^k_l = [t]$ (resp. $\beta^k_l = [t*]$).

   ou bien

   (a') Il existe $t_1,\ldots,t_m \in T_\Sigma(V)$ tel que pour tous i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, α^i_l et β^j_l sont tous égaux à $[\{t_1,\ldots,t_m\}]$ ou $\langle\{t_1,\ldots,t_m\}\rangle$, ou bien α^i_l et β^j_l sont tous égaux à $[\{t_1,\ldots,t_m\}*]$ ou $\langle\{t_1,\ldots,t_m\}*\rangle$.
   (b') Pour tous i, j, $1 \leq i \leq p$, $1 \leq j \leq q$, $\alpha^i_l = [\{t_1,\ldots,t_m\}]$ et $\beta^j_l = \langle\{t_1,\ldots,t_m\}\rangle$ (resp. $\alpha^i_l = [\{t_1,\ldots,t_m\}*]$ et $\beta^j_l = \langle\{t_1,\ldots,t_m\}*\rangle$), sauf peut-être soit pour un k, $1 \leq k \leq p$, tel que $\alpha^k_l = \langle\{t_1,\ldots,t_m\}\rangle$ (resp. $\alpha^k_l = \langle\{t_1,\ldots,t_m\}*\rangle$), soit pour un k, $1 \leq k \leq q$, tel que $\beta^k_l = [\{t_1,\ldots,t_m\}]$ (resp. $\beta^k_l = [\{t_1,\ldots,t_m\}*]$).

Pour expliquer la nécessité de cette relation entre les modalités, revenons à la règle de nécessitation. Nous avons vu, lorsque nous avons présenté la règle de nécessitation pour le calcul des séquents, qu'elle pouvait prendre différentes formes. Nous avons en fait les trois règles suivantes pour les modalités indexées par un

terme t, ainsi que pour les modalités indexées par $t*$, par $\{t_1, \ldots, t_m\}$ et par $\{t_1, \ldots, t_m\}*$:

$$\frac{\Gamma \mathrel{|\!\sim} \Delta}{[t]\Gamma \mathrel{|\!\sim} \langle t\rangle\Delta} \qquad \frac{\Gamma \mathrel{|\!\sim} \varphi, \Delta}{[t]\Gamma \mathrel{|\!\sim} [t]\varphi, \langle t\rangle\Delta} \qquad \frac{\Gamma, \varphi \mathrel{|\!\sim} \Delta}{[t]\Gamma, \langle t\rangle\varphi \mathrel{|\!\sim} \langle t\rangle\Delta}$$

Pour qu'une formule $M_1\gamma_1, \ldots, M_p\gamma_p \mathrel{|\!\sim} N_1\delta_1, \ldots, N_q\delta_q$ soit la conclusion d'une ou plusieurs règles de nécessitation, il faut tout d'abord que chaque suite de modalités $M_1, \ldots, M_p, N_1, \ldots, N_q$ soit composée du même nombre n de modalités (alinéa 1. de la définition). Ensuite, si on prend le cas (non réducteur) où chaque suite de modalités est composée d'un unique modalité, on a la formule $\alpha_1\gamma_1, \ldots, \alpha_p\gamma_p \mathrel{|\!\sim} \beta_1\delta_1, \ldots, \beta_q\delta_q$. Pour que cette formule soit la conclusion d'une des trois règles de nécessitation ci-dessus, il faut que toutes les modalités $\alpha_1, \ldots, \alpha_p, \beta_1, \ldots, \beta_q$ soient indexées par un même terme t, donc qu'elles soient toutes de la forme $[t]$ ou $\langle t\rangle$ (point 2.a. de la définition). Ensuite, trois cas se présentent (point 2.b. de la définition) :

- pour que la formule soit la conclusion de la première règle, il faut que tous les α_i, $1 \leq i \leq p$ soient égaux à $[t]$ et que tous les β_i, $1 \leq i \leq q$ soient égaux à $\langle t\rangle$;
- pour qu'elle soit la conclusion de la deuxième règle, il faut que tous les α_i, $1 \leq i \leq p$ soient égaux à $[t]$ et que tous les β_i, $1 \leq i \leq q$ soient égaux à $\langle t\rangle$, sauf un β_k, $1 \leq k \leq q$ qui doit être égal à $[t]$;
- pour qu'elle soit la conclusion de la troisième règle, il faut que tous les α_i, $1 \leq i \leq p$ soient égaux à $[t]$, sauf un α_k, $1 \leq k \leq p$ qui doit être égal à $\langle t\rangle$, et que tous les β_i, $1 \leq i \leq q$ soient égaux à $\langle t\rangle$.

Le même raisonnement tient également avec les modalités indexées par $t*$, par $\{t_1, \ldots, t_m\}$ et par $\{t_1, \ldots, t_m\}*$.

Cette relation sur les modalités nous assure donc la proposition suivante :

PROPOSITION VIII.1. *Soit* $\gamma_1, \ldots, \gamma_p \mathrel{|\!\sim} \delta_1, \ldots, \delta_q$ *un séquent. Soit* $(M_1, \ldots, M_p) \in (M_\Sigma(V)^*)^p$ *et* $(N_1, \ldots, N_q) \in (M_\Sigma(V)^*)^q$ *tels que* $(M_1, \ldots, M_p)\mathcal{N}(N_1, \ldots, N_q)$. *Alors il existe un arbre de preuve de conclusion* $M_1\gamma_1, \ldots, M_p\gamma_p \mathrel{|\!\sim} N_1\delta_1, \ldots, N_q\delta_q$ *uniquement composé d'instances de la règle de nécessitation.*

La preuve de l'objectif de test à partir de l'axiome avec lequel il a pu en partie être unifié va alors être composée de trois étapes : tout d'abord l'application de la substitution unificatrice, puis l'application d'une ou plusieurs règles de nécessitation pour ajouter les modalités nécessaires, et enfin l'application d'autant de règles de coupure qu'il faut pour éliminer toutes les formules de l'axiome qui n'ont pas pu être unifiées avec l'objectif de test d'une part, et pour ajouter les formules de l'objectif de test qui n'ont pas pu être déduites de l'axiome d'autre part. L'arbre de preuve global est alors de la forme suivante : les deux premières étapes de la preuve forment le premier arbre, noté Ax, tandis que la partie de l'arbre consistant en l'application des règles de coupure est donné plus bas.

$$\dfrac{\dfrac{\dfrac{}{\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mathrel{|\!\sim} \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q}\,\mathsf{Ax}}{\sigma(\psi_1), \ldots, \sigma(\psi_p), \sigma(\xi_1), \ldots, \sigma(\xi_l) \mathrel{|\!\sim} \sigma(\zeta_1), \ldots, \sigma(\zeta_k), \sigma(\varphi_1), \ldots, \sigma(\varphi_q)}\,\mathsf{Subs}}{\vdots}$$

$$M_1\sigma(\psi_1), \ldots, M_p\sigma(\psi_p), M'_1\sigma(\xi_1), \ldots, M'_l\sigma(\xi_l) \mathrel{|\!\sim} N'_1\sigma(\zeta_1), \ldots, N'_k\sigma(\zeta_k), N_1\sigma(\varphi_1), \ldots, N_q\sigma(\varphi_q)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma' \mathrel{|\!\sim} M'_1\sigma(\xi_1), \Delta'} \quad \dfrac{\dfrac{Ax \qquad \dfrac{\vdots}{\Gamma', N'_1\sigma(\zeta_1) \mathrel{|\!\sim} \Delta'}}{\Gamma, \Gamma', M'_1\sigma(\xi_1), \ldots, M'_l\sigma(\xi_l) \mathrel{|\!\sim} N'_2\sigma(\zeta_2), \ldots, N'_k\sigma(\zeta_k), \Delta), \Delta')}\,\text{Cut}}{\vdots}\,\text{Cut}}{\Gamma, \Gamma', M'_1\sigma(\xi_1), \ldots, M'_l\sigma(\xi_l) \mathrel{|\!\sim} \Delta, \Delta'}\,\text{Cut}}{\Gamma, \Gamma', M'_2\sigma(\xi_2), \ldots, M'_l\sigma(\xi_l) \mathrel{|\!\sim} \Delta, \Delta'}\,\text{Cut}}{\vdots}\,\text{Cut}}{\Gamma, \Gamma' \mathrel{|\!\sim} \Delta, \Delta'}\,\text{Cut}$$

où $\Gamma = \{M_1\sigma(\psi_1), \ldots, M_p\sigma(\psi_p)\}$, $\Gamma' = \{\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m)\}$, $\Delta = \{N_1\sigma(\varphi_1), \ldots, N_q\sigma(\varphi_q)\}$ et $\Delta' = \{\sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}$.

La procédure de dépliage est alors exprimée au travers des deux règles suivantes :

$$\textbf{Réduction}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\Gamma \mathrel{|\!\sim} \Delta\}, \sigma')\}}{\Psi \cup \{(\sigma(\mathcal{C}), \sigma \circ \sigma')\}} \quad \exists \gamma \in \Gamma, \exists \delta \in \Delta \text{ t.q. } \sigma(\gamma) = \sigma(\delta)$$

$$\textbf{Dépliage}\,\frac{\Psi \cup \{(\mathcal{C} \cup \{\phi\}, \sigma')\}}{\Psi \cup \bigcup_{(c,\sigma) \in Dep(\phi)} \{(\sigma(\mathcal{C}) \cup c, \sigma \circ \sigma')\}}$$

où $Dep(\phi)$ pour $\phi = \gamma_1, \ldots, \gamma_m \mathrel{|\!\sim} \delta_1, \ldots, \delta_n$ est l'ensemble défini de la façon suivante :

$$\left\{ \left(\begin{array}{l} \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m), N'_i\sigma(\zeta_i) \mathrel{|\!\sim} \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1 \le i \le k} \\ \cup \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m) \mathrel{|\!\sim} M'_i\sigma(\xi_i), \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1 \le i \le l} \end{array}, \sigma \right) \right.$$
$$\left| \begin{array}{l} \psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mathrel{|\!\sim} \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q \in Ax, \\ 1 \le p \le m, \forall 1 \le i \le p, \exists M_i \in M_\Sigma(V)^* \text{ t.q. } M_i\sigma(\psi_i) = \sigma(\gamma_i), \\ 1 \le q \le n, \forall 1 \le i \le q, \exists N_i \in M_\Sigma(V)^* \text{ t.q. } N_i\sigma(\varphi_i) = \sigma(\delta_i), \\ \forall 1 \le i \le l, \forall 1 \le j \le k, M'_i, N'_j \in M_\Sigma(V)^* \\ (M_1, \ldots, M_p, M'_1, \ldots, M'_l)\mathcal{N}(N_1, \ldots, N_q, N'_1, \ldots, N'_k) \\ k, l \in \mathbb{N} \end{array} \right\}$$

La règle **Réduction** élimine les tautologies des ensembles de contraintes. La règle **Dépliage** est celle donnée au chapitre VI à laquelle est ajoutée la gestion des modalités. Cette règle consiste à remplacer la formule ϕ par l'ensemble de contraintes de $Dep(\phi)$, qui est constitué des formules qui ont dû être éliminées par la règle de coupure pour obtenir une preuve de ϕ. En effet, s'il existe un axiome tel qu'on puisse unifier une partie de ses sous-formules avec une partie des sous-formules de l'objectif de test, à des modalités près qui doivent respecter la forme imposée par la relation $\mathcal{N}$, alors il est possible de prouver ϕ à partir de cet axiome, en coupant une à une toutes les formules de l'axiome qui n'ont pas pu être unifiées avec l'objectif de test.

Chaque unification avec un axiome donne autant de couples (c, σ) qu'il existe de façons d'instancier $M'_1, \ldots, M'_l$ et $N'_1, \ldots, N'_k$ pour que $(M_1, \ldots, M_p, M'_1, \ldots, M'_l)$ et $(N_1, \ldots, N_q, N'_1, \ldots, N'_k)$ soient

dans la relation $\mathcal{N}$. La formule initiale ϕ est alors remplacée par au moins autant d'ensemble de formules qu'il existe d'axiomes à laquelle elle puisse être unifiée. La définition de $Dep(\phi)$ étant fondée sur l'unification, cet ensemble est calculable si la spécification Sp a un nombre fini d'axiomes. Par conséquent, étant donnée une formule ϕ, on a le critère de sélection C_ϕ qui associe à tout $T_{(\mathcal{C},\sigma'),\varphi}$ le jeu de tests $(T_{(\sigma(\mathcal{C}\setminus\{\phi\})\cup c,\sigma\circ\sigma'),\varphi})_{(c,\sigma)\in Dep(\phi)}$ si $\phi \in \mathcal{C}$, et $T_{\mathcal{C},\varphi}$ sinon.

On écrit $(\Psi,\varphi) \vdash_D (\Psi',\varphi)$ si Ψ' peut être dérivé de Ψ en appliquant la règle **Réduction** ou **Dépliage**. La procédure de dépliage prend donc en entrée une spécification modale sans quantificateurs Sp et une formule modale sans quantificateurs φ et applique successivement les règles **Réduction** et **Dépliage** pour générer la suite de dérivations suivante :

$$(\Psi_0,\varphi) \vdash_D (\Psi_1,\varphi) \vdash_D (\Psi_2,\varphi) \dots$$

EXEMPLE VIII.2 — *Listes.*

Supposons qu'on veuille tester la formule $[pair][queue]\ tête = a \Rightarrow [queue][queue][pair]\ tête = b$. Le premier ensemble de couples Ψ_0 est le suivant :

$$\Psi_0 = \{(\{[pair][queue]\ tête = a \mid\sim [queue][queue][pair]\ tête = b\}, \mathrm{Id})\}$$

Appliquer la procédure de dépliage sur l'unique contrainte de l'ensemble de contraintes de ce couple donne la famille de couples suivante :

$$\begin{aligned}
\Psi_1 = \{&(\{[pair][queue][impair]\ tête = n_0 \mid\sim [queue][queue][pair]\ tête = m_0\}, \sigma_1), && (1)\\
&(\{\langle pair\rangle[queue][impair]\ tête = n_0 \mid\sim [queue][queue][pair]\ tête = m_0\}, \sigma_1), && (1)\\
&(\{[pair]\langle queue\rangle[impair]\ tête = n_0 \mid\sim [queue][queue][pair]\ tête = m_0\}, \sigma_1), && (1)\\
&(\{\langle pair\rangle\langle queue\rangle[impair]\ tête = n_0 \mid\sim [queue][queue][pair]\ tête = m_0\}, \sigma_1), && (1)\\
&(\{[queue][impair][queue]\ tête = n_0 \mid\sim [queue][queue][pair]\ tête = m_0\}, \sigma_1), && (5)\\
&(\{[pair][queue]\ tête = n_0 \mid\sim [queue][queue][pair][impair]\ tête = m_0\}, \sigma_2), && (2)\\
&(\{[pair][queue]\ tête = n_0 \mid\sim [queue][queue][queue][impair]\ tête = m_0\}, \sigma_2)\} && (6)
\end{aligned}$$

où $\sigma_1 : a \mapsto n_0, b \mapsto m_0, n \mapsto n_0$ et $\sigma_2 : a \mapsto n_0, b \mapsto m_0, n \mapsto m_0$. Chaque couple de Ψ_1 est étiqueté par le numéro de l'axiome avec lequel l'objectif de test a été unifié.

Le premier couple de Ψ_1 provient de l'unification de l'objectif de test avec l'axiome (1). Puisque $M_1\sigma_1(\psi_1) = \sigma_1(\gamma_1)$, où $M_1 = [pair][queue]$, ψ_1 est la formule $tête = n$ et γ_1 est la formule $tête = a$, les contraintes obtenues sont les séquents $N_1'\sigma_1(\zeta_1) \mid\sim \sigma_1(\delta_1)$ où ζ_1 est la formule $[impair]\ tête = n$, δ_1 est la formule $[queue][queue][pair]\ tête = b$, et N_1' doit être tel que $M_1\ \mathcal{N}\ N_1$. Selon la définition de la relation $\mathcal{N}$, plusieurs N_1 correspondent, à savoir $[pair][queue]$, $\langle pair\rangle[queue]$, $[pair]\langle queue\rangle$ et $\langle pair\rangle\langle queue\rangle$, d'où les quatre contraintes générées par l'unification avec l'axiome (1).

On remarque que la formule sous test n'est une conséquence de la spécification que si $a = b$. La procédure de dépliage engendre alors deux types de couples formés d'un ensembles de contraintes et d'une substitution : ceux dans lesquels la substitution σ est telle que $\sigma(a) = \sigma(b)$, qui mènent à des cas de test puisque ce sont des conséquences de la spécification, et ceux dans lesquels $\sigma(a) \neq \sigma(b)$, qui ne mènent pas à des cas de test. Ici, lorsqu'une contrainte est unifiée avec les deux membres de l'axiome (1) ou (2), la substitution fait coïncider a et b et le couple contraintes substitution obtenu mènera à des cas de test.

Comme on l'a déjà dit, la procédure de dépliage ne peut pas faire de distinction entre ces deux types de contraintes, mais celle-ci sera faite avant de soumettre les objectifs de test contraints correspondants, en appliquant une substitution close ρ à toutes les formules présentes dans le jeu de test contraint. Comme, par définition, $\rho(\psi)$ doit être une conséquence de la spécification, les objectifs de tests contraints dans lesquels $\sigma(a) \neq \sigma(b)$ ne seront pas soumis au système. ◇

De la même manière que dans le cadre de la sélection à partir de spécifications du premier ordre sans quantificateurs, la procédure de dépliage a été définie de façon à couvrir les comportements d'un objectif de test donné, représenté par la formule φ. Pour couvrir plus largement le jeu de tests exhaustif $Sp^\bullet \cap Obs$, une stratégie consiste à ordonner les formules modales en fonction de leur taille, de la façon suivante :

$$\Phi_0 = \{ \mid\sim r(x_1, \ldots, x_n) \mid r : s_1 \times \ldots \times s_n \in R, \forall i, 1 \leq i \leq n, x_i \in V_{s_i}\}$$

$$\Phi_{n+1} = \{ \mid\sim \neg\psi,\ \mid\sim [m]\psi,\ \mid\sim \psi_1 @ \psi_2 \mid m \in M_\Sigma(V), @ \in \{\wedge, \vee, \Rightarrow\}, \psi, \psi_1, \psi_2 \in \Phi_n\}$$

Ensuite, de manière à gérer la taille du jeu de tests $Sp^\bullet \cap Obs$, on commence par choisir un entier $k \in \mathbb{N}$, et on applique pour tout i, $1 \leq i \leq k$, la procédure de dépliage présentée ci-dessus à chaque formule appartenant à Φ_i.

On prouve maintenant les propriétés de correction et de complétude du critère de sélection défini par la procédure de dépliage à partir de spécifications modales sans quantificateurs.

THÉORÈME VIII.2. *Si* $\langle \Psi, \varphi \rangle \vdash_U \langle \Psi', \varphi \rangle$, *alors* $T_{\Psi,\varphi} = T_{\Psi',\varphi}$.

Preuve.
(Correction) Montrons que si $\langle \Psi, \varphi \rangle \vdash_U \langle \Psi', \varphi \rangle$, alors $T_{\Psi',\varphi} \subseteq T_{\Psi,\varphi}$.

Si la règle appliquée est **Réduction**, c'est trivial. Si la règle appliquée est la règle de **Dépliage**, par définition, on dit montrer que pour tout $(\mathcal{C}, \sigma') \in \Psi$, pour tout $\psi \in \mathcal{C}$, pour tout $(c, \sigma) \in Dep(\psi)$, $T_{(c,\sigma\circ\sigma'),\varphi} \subseteq T_{(\{\psi\},\sigma'),\varphi}$. On doit alors montrer que pour toute substitution close $\rho : V \to T_\Sigma$ telle que $Sp \models \rho(\chi)$, pour tout $\chi \in c$, il existe $\rho' : V \to T_\Sigma$ telle que $Sp \models \rho'(\psi)$.

On suppose que la formule ψ est de la forme $\gamma_1, \ldots, \gamma_m \mid\sim \delta_1, \ldots, \delta_n$, et que l'ensemble c tel que $(c, \sigma) \in Dep(\psi)$ est de la forme

$$\{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m), N_i'\sigma(\zeta_i) \mid\sim \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1\leq i\leq k}$$
$$\cup \{(\sigma(\gamma_{p+1}), \ldots, \sigma(\gamma_m) \mid\sim M_i'\sigma(\xi_i), \sigma(\delta_{q+1}), \ldots, \sigma(\delta_n)\}_{1\leq i\leq l}$$

où $1 \leq p \leq m$ et $1 \leq q \leq n$ sont tels que $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mid\sim \zeta_1, \ldots, \zeta_k, \varphi_1, \ldots, \varphi_q \in Ax$, pour tout $1 \leq i \leq p$ il existe $M_i \in M_\Sigma(V)^*$ tel que $M_i\sigma(\psi_i) = \sigma(\gamma_i)$, pour tout $1 \leq i \leq q$ il existe $N_i \in M_\Sigma(V)^*$ tel que $N_i\sigma(\varphi_i) = \sigma(\delta_i)$, et $M_1', \ldots, M_l', N_1', \ldots, N_k' \in M_\Sigma(V)^*$ tel que $(M_1, \ldots, M_p, M_1', \ldots, M_l')\mathcal{N}(N_1, \ldots, N_q, N_1', \ldots, N_k')$.

Puisque $(M_1, \ldots, M_p, M_1', \ldots, M_l')\mathcal{N}(N_1, \ldots, N_q, N_1', \ldots, N_k')$, on a l'arbre de preuve suivant, noté

Ax par la suite :

$$\dfrac{\dfrac{\dfrac{}{\psi_1,\ldots,\psi_p,\xi_1,\ldots,\xi_l \vdash \zeta_1,\ldots,\zeta_k,\varphi_1,\ldots,\varphi_q}\,\mathsf{Ax}}{\rho\sigma(\psi_1),\ldots,\rho\sigma(\psi_p),\rho\sigma(\xi_1),\ldots,\rho\sigma(\xi_l) \vdash \rho\sigma(\zeta_1),\ldots,\rho\sigma(\zeta_k),\rho\sigma(\varphi_1),\ldots,\rho\sigma(\varphi_q)}\,\mathsf{Subs}}{\vdots}$$

$$M_1\rho\sigma(\psi_1),\ldots,M_p\rho\sigma(\psi_p),M'_1\rho\sigma(\xi_1),\ldots,M'_l\rho\sigma(\xi_l) \vdash N'_1\rho\sigma(\zeta_1),\ldots,N'_k\rho\sigma(\zeta_k),N_1\rho\sigma(\varphi_1),\ldots,N_q\rho\sigma(\varphi_q)$$

où les points de suspension remplacent les multiples applications de la règle de nécessitation.

On a alors l'arbre de preuve suivant, où $\Gamma = \{M_1\rho\sigma(\psi_1),\ldots,M_p\rho\sigma(\psi_p)\}$, $\Delta = \{N_1\rho\sigma(\varphi_1),\ldots,N_q\rho\sigma(\varphi_q)\}$, $\Gamma' = \{\rho\sigma(\gamma_{p+1}),\ldots,\rho\sigma(\gamma_m)\}$, $\Delta' = \{\rho\sigma(\delta_{q+1}),\ldots,\rho\sigma(\delta_n)\}$, pour tout i, $1 \leq i \leq l$, $\Omega_i = \{M'_i\rho\sigma(\xi_i),\ldots,M'_l\rho\sigma(\xi_l)\}$ et pour tout i, $1 \leq i \leq k$, $\Lambda_i = \{N'_i\rho\sigma(\zeta_i),\ldots,N'_k\rho\sigma(\zeta_k)\}$. La composition $\sigma' \circ \sigma$ de deux substitutions $\sigma : V \to T_\Sigma(V)$ et $\sigma' : T_\Sigma(V) \to T_\Sigma(V)$, appliquée à une formule φ, est notée $\sigma'\sigma(\varphi)$.

$$\dfrac{\dfrac{\vdots}{\Gamma' \vdash M'_l\rho\sigma(\xi_l),\Delta'} \qquad \dfrac{\dfrac{\vdots}{\Gamma' \vdash M'_2\rho\sigma(\xi_2),\Delta'} \qquad \dfrac{\dfrac{\vdots}{\Gamma' \vdash M'_1\rho\sigma(\xi_1),\Delta'} \qquad SA}{\Gamma,\Gamma',\Omega_2 \vdash \Delta,\Delta'}}{\Gamma,\Gamma',\Omega_l \vdash \Delta,\Delta'}}{\Gamma,\Gamma' \vdash \Delta,\Delta'}$$

où SA est le sous-arbre suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{Ax}{\Gamma,\Omega_1 \vdash \Lambda_1,\Delta} \qquad \dfrac{\vdots}{\Gamma',N'_1\rho\sigma(\zeta_1) \vdash \Delta'}}{\Gamma,\Gamma',\Omega_1 \vdash \Lambda_2,\Delta,\Delta'} \qquad \dfrac{\vdots}{\Gamma',N'_2\rho\sigma(\zeta_2) \vdash \Delta'}}{\vdots} \qquad \dfrac{\vdots}{\Gamma',N'_k\rho\sigma(\zeta_k) \vdash \Delta'}}{\Gamma,\Gamma',\Omega_1 \vdash \Lambda_k,\Delta,\Delta'}}{\Gamma,\Gamma',\Omega_1 \vdash \Delta,\Delta'}$$

(Complétude) Montrons que si $\langle\Psi,\varphi\rangle \vdash_U \langle\Psi',\varphi\rangle$, alors $T_{\Psi,\varphi} \subseteq T_{\Psi',\varphi}$.

Par définition de la règle de **Dépliage**, on doit prouver que $T_{(\{\psi\},\sigma'),\varphi} \subseteq \bigcup_{(c,\sigma)\in Dep(\psi)} T_{(c,\sigma\circ\sigma'),\varphi}$. On doit alors prouver que pour toute substitution close $\rho : V \to T_\Sigma$ telle que $Sp \models \rho(\psi)$, il existe $(c,\sigma) \in Dep(\psi)$ tel qu'il existe $\rho' : V \to T_\Sigma$ telle que $Sp \models \rho'(\chi)$ pour tout $\chi \in c$. En d'autres termes, on doit prouver que $\rho(\psi)$ peut être déduit de la spécification Sp s'il existe $(c,\sigma) \in Dep(\psi)$, et $\rho' : V \to T_\Sigma$ tels que $Sp \models \rho'(\chi)$ pour tout $\chi \in c$.

Tout d'abord, remarquons que la procédure de dépliage définit une stratégie qui limite l'espace de recherche d'arbres de preuve aux arbres ayant une structure particulière. La procédure définit une stratégie de recherche de preuve qui sélectionne les arbres de preuve où :

- aucune instance de la règle de coupure n'apparaît au-dessus d'instances des règles de substitution et de nécessitation ;
- aucune instance de la règle de substitution n'apparaît au-dessus d'instances de la règle de nécessitation ;
- il n'y a pas d'instance de la règle de coupure dont les prémisses sont toutes les deux des conclusions d'instances de la règle de coupure.

On doit alors prouver qu'il existe un arbre de preuve de conclusion $\rho(\psi)$ possédant cette structure. On va en fait prouver une résultat plus fort : on va définir des transformations élémentaires d'arbres de preuve, qui permettent de réécrire des combinaisons élémentaires de règles d'inférence, et nous allons enduite prouver que la transformation globale résultante est faiblement normalisante, et que les formes normales sont des arbres de preuve ayant la structure voulue.

Le cas de la règle de coupure au-dessus de la règle de substitution :

$$\dfrac{\dfrac{\Gamma \vdash \Delta, \varphi \quad \Gamma', \varphi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\mathsf{Cut}}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta')}\mathsf{Subs} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash \Delta, \varphi}{\sigma(\Gamma) \vdash \sigma(\Delta), \sigma(\varphi)}\mathsf{Subs} \quad \dfrac{\Gamma', \varphi \vdash \Delta'}{\sigma(\Gamma'), \sigma(\varphi) \vdash \sigma(\Delta')}\mathsf{Subs}}{\sigma(\Gamma), \sigma(\Gamma') \vdash \sigma(\Delta), \sigma(\Delta')}\mathsf{Cut}$$

Le cas de la règle de coupure au-dessus de la règle de nécessitation :

$$\dfrac{\dfrac{\Gamma \vdash \Delta, \varphi \quad \Gamma', \varphi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\mathsf{Cut}}{[t]\Gamma, [t]\Gamma' \vdash \langle t\rangle\Delta, \langle t\rangle\Delta'}\mathsf{Nec} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash \Delta, \varphi}{[t]\Gamma \vdash \langle t\rangle\Delta, \langle t\rangle\varphi}\mathsf{Nec} \quad \dfrac{\Gamma', \varphi \vdash \Delta'}{[t]\Gamma', \langle t\rangle\varphi \vdash \langle t\rangle\Delta'}\mathsf{Nec}}{[t]\Gamma, [t]\Gamma' \vdash \langle t\rangle\Delta, \langle t\rangle\Delta'}\mathsf{Cut}$$

Le cas où deux instances de la règle de coupure se trouve au-dessus d'une troisième doit être divisé en deux cas, selon la position de la dernière formule coupée dans les prémisses de l'instance droite de la règle de coupure.

Le cas où la formule φ est dans la prémisse gauche. Par exemple :

$$\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1, \varphi \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1'}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\mathsf{Cut} \quad \dfrac{\Gamma_2, \varphi \mathrel{|\!\sim} \Delta_2, \varphi_2 \quad \Gamma_2', \varphi_2 \mathrel{|\!\sim} \Delta_2'}{\Gamma_2, \Gamma_2', \varphi \mathrel{|\!\sim} \Delta_2, \Delta_2'}\mathsf{Cut}}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\mathsf{Cut}$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1, \varphi \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1'}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\mathsf{Cut} \quad \Gamma_2, \varphi \mathrel{|\!\sim} \Delta_2, \varphi_2}{\Gamma_1, \Gamma_1', \Gamma_2 \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \varphi_2}\mathsf{Cut} \quad \Gamma_2', \varphi_2 \mathrel{|\!\sim} \Delta_2'}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\mathsf{Cut}$$

Le cas où la formule φ est dans la prémisse droite. Par exemple :

$$\dfrac{\dfrac{\Gamma_1 \mathrel{|\!\sim} \Delta_1, \varphi_1 \quad \Gamma_1', \varphi_1 \mathrel{|\!\sim} \Delta_1', \varphi}{\Gamma_1, \Gamma_1' \mathrel{|\!\sim} \Delta_1, \Delta_1', \varphi}\mathsf{Cut} \quad \dfrac{\Gamma_2 \mathrel{|\!\sim} \Delta_2, \varphi_2 \quad \Gamma_2', \varphi_2, \varphi \mathrel{|\!\sim} \Delta_2'}{\Gamma_2, \Gamma_2', \varphi \mathrel{|\!\sim} \Delta_2, \Delta_2'}\mathsf{Cut}}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mathrel{|\!\sim} \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\mathsf{Cut}$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \mid\sim \Delta_1, \varphi_1 \quad \Gamma_1', \varphi_1 \mid\sim \Delta_1', \varphi}{\Gamma_1, \Gamma_1' \mid\sim \Delta_1, \Delta_1', \varphi}\mathsf{Cut} \qquad \Gamma_2', \varphi_2', \varphi \mid\sim \Delta_2'}{\Gamma_1, \Gamma_1', \Gamma_2', \varphi_2 \mid\sim \Delta_1, \Delta_1', \Delta_2'}\mathsf{Cut} \qquad \Gamma_2 \mid\sim \Delta_2, \varphi_2}{\Gamma_1, \Gamma_1', \Gamma_2, \Gamma_2' \mid\sim \Delta_1, \Delta_1', \Delta_2, \Delta_2'}\mathsf{Cut}$$

Pour un arbre $\pi = \dfrac{\pi_1 \quad \pi_2}{\Gamma_1, \Gamma_2 \mid\sim \Delta_1, \Delta_2}\mathsf{Cut}$, notons $m(\pi)$ la mesure de π, définie comme étant le nombre d'instances de la règle de coupure dans π_2. Un arbre de preuve est dit maximal si et seulement s'il est de la forme

$$\dfrac{\dfrac{\dfrac{\pi_{1_1} \quad \pi_{1_2}}{\Gamma_1 \mid\sim \Delta_1, \varphi}\mathsf{Cut} \qquad \dfrac{\pi_{2_1} \quad \pi_{2_2}}{\Gamma_2, \varphi \mid\sim \Delta_2}\mathsf{Cut}}{\Gamma_1, \Gamma_2 \mid\sim \Delta_1, \Delta_2}\mathsf{Cut}$$

et $m(\pi) = 1$. En appliquant la stratégie qui consiste à réduire les arbres de preuve maximaux, il est facile de montrer que la mesure m décroît pour chaque transformation élémentaire donnée plus haut.

Puisque par hypothèse, $Sp \models \rho(\psi)$, et ψ n'est pas une tautologie, il existe nécessairement un axiome $\psi_1, \ldots, \psi_p, \xi_1, \ldots, \xi_l \mid\sim \zeta_1, \ldots, \zeta_r, \varphi_1, \ldots, \varphi_q$ et une substitution close ρ' telle que pour tout $1 \leq i \leq p$, il existe $M_i \in M_\Sigma(V)^*$ tel que $M_i\rho'(\psi_i) = \rho'(\gamma_i)$, pour tout $1 \leq i \leq q$ il existe $N_i \in M_\Sigma(V)^*$ tel que $N_i\rho'(\varphi_i) = \rho'(\delta_i)$, et $(M_1, \ldots, M_p, M_1', \ldots, M_l')\mathcal{N}(N_1, \ldots, N_q, N_1', \ldots, N_k')$. La substitution ρ' est donc un unificateur de chaque ψ_i et γ_i', et de chaque φ_i et δ_i'. Il existe donc un arbre de preuve résultant de la transformation définie plus haut, de conclusion $\rho'(\psi)$, où $\rho' = \rho''$, et de la forme :

$$\dfrac{\dfrac{\vdots}{\Gamma' \vdash M_l'\rho'(\xi_l), \Delta'} \qquad \dfrac{\dfrac{\dfrac{\vdots}{\Gamma' \vdash M_2'\rho'(\xi_2), \Delta'} \qquad \dfrac{\dfrac{\vdots}{\Gamma' \vdash M_1'\rho'(\xi_1), \Delta'} \qquad SA}{\Gamma, \Gamma', \Omega_2 \vdash \Delta, \Delta'}}{\vdots}}{\Gamma, \Gamma', \Omega_l \vdash \Delta, \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

où SA est le sous-arbre suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{Ax}{\Gamma, \Omega_1 \vdash \Lambda_1, \Delta} \qquad \dfrac{\vdots}{\Gamma', N_1'\rho'(\zeta_1) \vdash \Delta'}}{\Gamma, \Gamma', \Omega_1 \vdash \Lambda_2, \Delta, \Delta'} \qquad \dfrac{\vdots}{\Gamma', N_2'\rho'(\zeta_2) \vdash \Delta'}}{\vdots}}{\Gamma, \Gamma', \Omega_1 \vdash \Lambda_k, \Delta, \Delta'} \qquad \dfrac{\vdots}{\Gamma', N_k'\rho'(\zeta_k) \vdash \Delta'}}{\Gamma, \Gamma', \Omega_1 \vdash \Delta, \Delta'}$$

avec Ax le sous-arbre :

$$\dfrac{\dfrac{\dfrac{}{\psi_1,\ldots,\psi_p,\xi_1,\ldots,\xi_l \vdash \zeta_1,\ldots,\zeta_k,\varphi_1,\ldots,\varphi_q}\,\mathsf{Ax}}{\rho'(\psi_1),\ldots,\rho'(\psi_p),\rho'(\xi_1),\ldots,\rho'(\xi_l) \vdash \rho'(\zeta_1),\ldots,\rho'(\zeta_k),\rho'(\varphi_1),\ldots,\rho'(\varphi_q)}\,\mathsf{Subs}}{\vdots}}{M_1\rho'(\psi_1),\ldots,M_p\rho'(\psi_p),M'_1\rho'(\xi_1),\ldots,M'_l\rho'(\xi_l) \vdash N'_1\rho'(\zeta_1),\ldots,N'_k\rho'(\zeta_k),N_1\rho'(\varphi_1),\ldots,N_q\rho'(\varphi_q)}$$

□

4 Comparaison avec le test de conformité

Nous abordons, dans cette section, la comparaison de notre approche du test avec celle habituellement suivie pour tester des systèmes dynamiques et réactifs, le test de conformité. Nous nous inspirons de la présentation du test de conformité donnée par Assia Touil dans sa thèse [Tou06]. Nous tentons de mener la comparaison entre les deux approches essentiellement au travers de la notion de correction d'un système par rapport à sa spécification. La méthode que nous suivons ici consiste à placer les deux approches dans le même cadre formel, celui des spécifications axiomatiques.

Nous présentons tout d'abord le formalisme sur lequel repose le test de conformité, celui des systèmes de transitions étiquetées. Nous posons les hypothèses de test de ce cadre et définissons la relation de conformité. Nous cherchons ensuite à associer à un IOSTS une spécification exprimée en logique modale, de façon à pouvoir étudier la transposition de la relation de conformité dans le cadre axiomatique. Nous abordons également les notions d'objectifs de test et de verdicts, que nous comparons à nos propres notions. Nous dressons enfin un bilan concis de nos observations.

4.1 Systèmes de transitions symboliques à entrées et sorties

Syntaxe. Les IOSTS sont des modèles permettant de spécifier des systèmes réactifs en décrivant à la fois leurs interactions avec leur environnement (sous la forme d'envoi et de réception de messages) et l'évolution de leur état interne (par des affectations de variables) [JJRZ05]. Le formalisme des IOSTS est une extension de celui des systèmes de transitions étiquetées à entrées et sorties nommés IOLTS [JJ04] (pour *Input Output Labelled Transition Systems*) ainsi que de celui des systèmes de transitions symboliques [FTW04] (STS pour *Symbolic Transition Systems*).

L'état du système est décrit par un ensemble de variables appelées variables d'état. Chaque interprétation de ces variables caractérise un état du système. La partie « données » du système est décrite à l'aide de la logique du premier ordre. Pour décrire les communications du système avec son environnement ou avec d'autres systèmes, on se donne également un ensemble de canaux de communication. On a donc la définition suivante d'une signature pour un IOSTS.

DÉFINITION VIII.5 — *IOSTS-signature.*

Une IOSTS-signature Ω est un triplet (Σ, V, C) où :

- Σ est une signature du premier ordre ;
- V est un ensemble de variables sur Σ appelées *variables d'état* ;

– C est un ensemble de noms de *canaux de communication* où chaque nom c est muni d'une arité dans $S \cup \{\varepsilon\}$. C est muni d'une partition $C^?$ et $C^!$ désignant respectivement le sous-ensemble des *canaux d'entrée* et celui des *canaux de sortie*. Chacun de ces sous-ensembles est une famille indexée par $S \cup \{\varepsilon\}$: $C^? = (C^?_s)_{s \in S} \cup C^?_\varepsilon$ et $C^! = (C^!_s)_{s \in S} \cup C^!_\varepsilon$.

Les communications entre IOSTS sont représentées par des messages. Ces messages sont de deux types :
– les émissions au travers d'un canal d'un signal ou d'une valeur représentée par un terme du premier ordre ;
– les réceptions au travers d'un canal d'un signal ou d'une valeur à stocker sur une variable.

On ajoute à ces communications avec l'extérieur une action appelée action interne. Celle-ci représente un changement d'état résultant du fonctionnement interne du système et non d'une interaction avec l'environnement. Elle peut également être issue de la synchronisation de deux communications complémentaires de deux IOSTS. En effet, lorsque les IOSTS sont composés entre eux pour constituer un unique système, il est classique de considérer que l'échange de données entre deux sous-systèmes par synchronisation d'un envoi et d'une réception est une action interne du système global. Nous n'aborderons pas ici cet aspect.

DÉFINITION VIII.6 — *Actions de communication.*

Soit $\Omega = (\Sigma, V, C)$ une IOSTS-signature. L'ensemble des *actions de communication* sur Ω, noté $Act(\Omega)$, est l'union $Input(\Omega) \cup Output(\Omega) \cup \{\tau\}$ où :
– $Input(\Omega)$ est l'ensemble $\{c?x \mid c \in C^?_s, x \in V_s\} \cup \{c? \mid c \in C^?_\varepsilon\}$ dont les éléments sont appelés *réceptions* ;
– $Output(\Omega)$ est l'ensemble $\{c!t \mid c \in C^!_s, t \in T_\Sigma(V)_s\} \cup \{c! \mid c \in C^!_\varepsilon\}$ dont les éléments sont appelés *émissions* ;
– τ représente de façon générique une action interne.

Un IOSTS est alors un système de transitions étiquetées par des actions de communications dans lesquelles sont distinguées les réceptions, les émissions et les actions internes au système. Ces actions de communication peuvent être conditionnées par une formule du premier ordre appelée garde. Elles peuvent également être suivies d'une modification interne de l'état du système par une substitution des variables d'état. Parmi les états est distingué un état dit initial, dans lequel les variables d'état peuvent être initialisées.

DÉFINITION VIII.7 — *IOSTS.*

Soit $\Omega = (\Sigma, V, C)$ une IOSTS-signature. Un *IOSTS* $\mathbb{G}$ sur Ω est un quadruplet $(\mathbb{Q}, q_0, \mathbb{T}, \iota)$ où :
– $\mathbb{Q}$ est un ensemble dont les éléments sont appelés *états* ;
– $q_0 \in \mathbb{Q}$ est l'*état initial* ;
– $\mathbb{T} \subseteq \mathbb{Q} \times For(\Sigma) \times Act(\Omega) \times T_\Sigma(V)^V \times \mathbb{Q}$ est la *relation de transition* ;
– $\iota : V \to T_\Sigma(V)$ est la *fonction d'initialisation*.

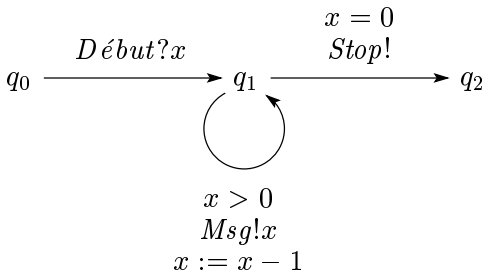
Une transition est donc un quintuplet $(q, \varphi, act, \varsigma, q')$ où q et q' sont respectivement les états source et cible de la transition, φ est la condition de franchissement de la transition, appelée garde, act est l'action de communication et ς est une substitution des variables qui permet de modifier l'état interne du système en modifiant la valeur d'un sous-ensemble des variables d'état. Une transition $(q, \varphi, act, \varsigma, q')$ est représentée

graphiquement de la manière suivante :

$$q \xrightarrow{\varphi \ act \ \varsigma} q'$$

EXEMPLE VIII.3 — *Compte à rebours.*

On donne ici l'exemple d'un IOSTS très simple. Le système qu'il représente attend de recevoir de l'environnement, au travers du canal $Début$, un entier qu'il stocke dans la variable x. Puis, tant que la valeur stockée dans x n'est pas nulle, il l'envoie vers l'environnement par le canal Msg et la décrémente. Lorsque la valeur est nulle, le message $Stop$ est envoyé et le système s'arrête.



La signature de cet IOSTS est le triplet (Σ, V, C) où :

$\Sigma = (S, F, R)$ avec :
$S = \{Nat\}$
$F = \{ \_ - \_ : Nat \times Nat \to Nat \}$
$R = \{ = : Nat \times Nat,$
$\quad > : Nat \times Nat \}$;

$V = V_{Nat} = \{x\}$

$C = C^{?}_{Nat} \amalg C^{!}_{Nat} \amalg C^{!}_{\varepsilon}$
$\quad = \{Début\} \cup \{Msg\} \cup \{Stop\}$

◇

Sémantique. Un IOSTS décrivant le comportement d'un système en termes de communications avec son environnement, il est caractérisé par l'ensemble des séquences finies d'émissions et de réceptions qu'il peut effectuer au cours d'une exécution. Les éléments de base constituant un IOSTS étant les transitions, nous avons tout d'abord besoin de donner une sémantique aux transitions.

DÉFINITION VIII.8 — *Sémantique d'une transition.*

Soient $\Omega = (\Sigma, V, C)$ une IOSTS-signature et $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \iota)$ un IOSTS sur Ω. Soit $\mathcal{M}$ un modèle de Σ. On note $Act_{\mathcal{M}}(\Omega)$ l'ensemble $(C \times \{?, !\} \times \mathcal{M}) \cup (C \times \{?, !\}) \cup \{\tau\}$. La *sémantique d'une transition* $tr = (q, \varphi, act, \varsigma, q') \in \mathbb{T}$ est la relation $Run(tr) \subseteq \mathcal{M}^V \times Act_{\mathcal{M}}(\Omega) \times \mathcal{M}^V$ telle que $(\nu^i, act_{\mathcal{M}}, \nu^f) \in Run(tr)$ si et seulement si :

- si act est de la forme $c!t$ (resp. $c!$), alors $\mathcal{M} \models_{\nu^i} \varphi$, $\nu^f = \varsigma \circ \nu^i$ et $act_{\mathcal{M}} = (c, !, \nu^i(t))$ (resp. $act_{\mathcal{M}} = (c, !)$) ;
- si act est de la forme $c?x$, alors $\mathcal{M} \models_{\nu^i} \varphi$, il existe ν^a telle que $\nu^a(y) = \nu^i(y)$ pour tout $y \neq x$, $\nu^f = \varsigma \circ \nu^a$ et $act_{\mathcal{M}} = (c, ?, \nu^a(x))$;
- si act est de la forme $c?$, alors $\mathcal{M} \models_{\nu^i} \varphi, \nu^f = \varsigma \circ \nu^i$ et $act_{\mathcal{M}} = (c, ?)$;
- si $act = \tau$, alors $\mathcal{M} \models_{\nu^i} \varphi, \nu^f = \varsigma \circ \nu^i$ et $act_{\mathcal{M}} = \tau$.

Les applications ν^i et ν^f désignent respectivement l'interprétation des variables avant et après le franchissement de la transition. L'expression $act_{\mathcal{M}}$ est la valeur du message envoyé ou reçu, muni du nom du canal et du symbole indiquant une émission ou une réception. On donne donc un sens à une transition d'une part au travers du changement d'état qu'elle induit et d'autre part au travers de la valeur qu'elle reçoit ou qu'elle produit.

Notation. Dans la suite, on notera les éléments de $Act_{\mathcal{M}}(\Omega)$ de la façon suivante :
- on notera $c?\nu(x)$ (respectivement $c?$) le triplet $(c, ?, \nu(x))$ (respectivement le couple $(c, ?)$) ;
- on notera $c!\nu(t)$ (respectivement $c!$) le triplet $(c, !, \nu(t))$ (respectivement le couple $(c, !)$).

Soit $tr \in \mathbb{T}$. On définit les applications $source : Run(tr) \to \mathcal{M}^V$, $act : Run(tr) \to Act_{\mathcal{M}}(\Omega)$ et $cible : Run(tr) \to \mathcal{M}^V$ de la façon suivante pour tout $r = (\nu^i, act_{\mathcal{M}}, \nu^f)$ de $Run(tr)$: $source(r) = \nu^i$, $act(r) = act_{\mathcal{M}}$ et $cible(r) = \nu^f$.

Les transitions peuvent être composées au sein d'un IOSTS afin d'obtenir ce qu'on appelle un chemin. Un chemin fini débute à l'état initial de l'IOSTS. Une exécution est alors une instance d'un chemin fini.

DÉFINITION VIII.9 — *Chemins finis.*

Soit $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \iota)$ un IOSTS sur une signature Ω. L'ensemble des chemins finis dans $\mathbb{G}$, noté $CF(\mathbb{G})$, est l'ensemble des suites finies $tr_1 \dots tr_n$ de transitions de $\mathbb{T}$ telles que $source(tr_1) = q_0$ et pour tout i, $1 \leq i < n$, $cible(tr_i) = source(tr_{i+1})$.

Un chemin fini d'un IOSTS étant une suite finie de transitions de cet IOSTS, donner une sémantique à un chemin revient alors à composer la sémantique des transitions le constituant.

DÉFINITION VIII.10 — *Exécution d'un chemin fini.*

Soient $\Omega = (\Sigma, V, C)$ une IOSTS-signature et $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \iota)$ un IOSTS sur Ω. Soit $\mathcal{M}$ un modèle de Σ. Soit un chemin fini $ch = tr_1 \dots tr_n \in CF(\mathbb{G})$. Les *exécutions de* ch sont toutes les suites $r_1 \dots r_n$ telles que :
- pour tout i, $1 \leq i \leq n$, $r_i \in Run(tr_i)$;
- il existe une interprétation $\nu_1 : V \to \mathcal{M}$ telle que $source(r_1) = \nu_1 \circ \iota$ et pour tout i, $1 \leq i < n$, $cible(r_i) = source(r_{i+1})$.

La suite de communications de l'IOSTS avec son environnement est appelée trace observable du chemin. La trace est la seule partie observable d'une exécution. Une trace observable d'un chemin est alors la restriction d'une exécution de ce chemin aux actions observables, c'est-à-dire aux réceptions et aux émissions de valeurs.

DÉFINITION VIII.11 — *Traces observables d'un chemin fini.*

Soit $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \iota)$ un IOSTS sur une signature Ω. Soit un chemin fini $ch = tr_1 \dots tr_n \in CF(\mathbb{G})$. L'ensemble des *traces observables de* ch, noté $Traces(ch)$, est l'ensemble des suites d'actions $act(r_1) \dots act(r_n)$ pour toute exécution $r_1 \dots r_n$ de ch.

La sémantique d'un IOSTS est alors l'ensemble des traces observables de tous les chemins finis de cet IOSTS.

DÉFINITION VIII.12 — *Sémantique d'un IOSTS.*

Soit $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \iota)$ un IOSTS sur une signature Ω. La *sémantique* de $\mathbb{G}$, notée $Traces(\mathbb{G})$, est définie de la façon suivante :

$$Traces(\mathbb{G}) = \bigcup_{ch \in CF(\mathbb{G})} Traces(ch)$$

4.2 Relation de conformité

Dans le cadre du test de conformité, il est supposé que l'environnement peut observer non seulement les sorties du système, mais également l'absence de sortie. Celle-ci peut être due à l'arrêt total du système (*deadlock*) ou bien à un arrêt provisoire dans un état où le système n'émet pas de valeur, ou encore à une suite d'actions internes bouclant indéfiniment. Cette absence de sortie est appelée quiescence. En pratique, la quiescence est observée en fixant un délai au-delà duquel on considère que le système ne réagira plus s'il n'a pas déjà réagi. Elle est modélisée par une action sur un canal noté δ et considérée comme une émission du système, puisqu'elle ne peut pas être contrôlée par l'environnement. On supposera dans cette section que les IOSTS ne contiennent pas de boucle d'actions internes.

On peut alors compléter un IOSTS en ajoutant des transitions étiquetées par $\delta!$ aux états qui peuvent être partiellement ou totalement bloquants. Un état peut être bloquant pour différentes raisons : le système est en attente d'une réception, il est donc bloqué jusqu'à la réception d'une valeur ; le système ne satisfait aucune des gardes qui lui permettrait d'émettre une valeur ou d'effectuer une action interne ; l'état ne possède aucune transition sortante. Dans chacun de ces cas, l'état sera muni d'une boucle étiquetée par $\delta!$. On appelle cette opération de complétion l'enrichissement d'un IOSTS par la quiescence.

DÉFINITION VIII.13 — *Enrichissement par la quiescence.*

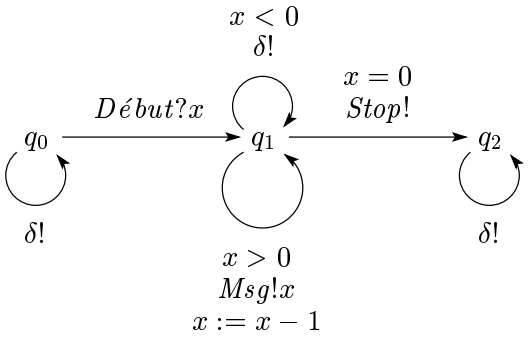
Soient $\Omega = (\Sigma, V, C)$ une IOSTS-signature et $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \mathrm{Id})$ un IOSTS sur Ω. L'*enrichissement de* $\mathbb{G}$ *par la quiescence* est l'IOSTS $\mathbb{G}_\delta = (\mathbb{Q}, q_0, \mathbb{T} \cup \mathbb{T}_\delta, \mathrm{Id})$ sur la signature $\Omega_\delta = (\Sigma, V, C \cup \{\delta\})$ tel que $(q, \varphi, act, \varsigma, q) \in \mathbb{T}_\delta$ si et seulement si :

- $act = \delta!, \varsigma = \mathrm{Id}$;
- si $tr_1, \ldots, tr_n$ sont toutes les transitions de la forme $tr_i = (q, \varphi_i, act_i, \varsigma_i, q_i)$ avec $act_i \in Output(\Omega)$ ou $act_i = \tau$:
  - si $n > 0$ alors $\varphi = \bigwedge_{i \leq n} \neg\varphi_i$,
  - sinon $\varphi = true$.

Tous les états sources de transitions étiquetées par une réception sont donc pourvus d'une boucle $\delta!$, ainsi que tous les deadlocks. On ajoute également une boucle étiquetée par $\delta!$ aux états dans lesquels le système ne peut pas effectuer les émissions ou les actions internes prévues s'il ne valide aucune des gardes de ses transitions sortantes, c'est-à-dire s'il valide $\bigwedge_{i \leq n} \neg\varphi_i$, où les φ_i, $1 \leq i \leq n$ sont les gardes.

EXEMPLE VIII.4 — *Compte à rebours.*

On enrichit l'IOSTS de l'exemple précédent par la quiescence. Dans l'état q_0, le système attend de recevoir une valeur, on lui ajoute donc une boucle étiquetée par $\delta!$. Dans l'état q_1, le système ne satisfait aucune des deux gardes si x est strictement négatif, on le munit donc d'une boucle $\delta!$ gardée par la formule $x < 0$. L'état q_2 est un deadlock, on lui ajoute donc également une boucle $\delta!$.



◇

Dans le cadre du test de conformité, les hypothèses de test sont les suivantes :

1. Le système sous test peut être modélisé par un IOSTS qu'on note $IMPL$. Les données manipulées par cet IOSTS sont des valeurs de l'ensemble M sous-jacent au modèle $\mathcal{M}$ de Σ. Ces valeurs peuvent toutes être dénotées par un terme clos sur Σ.
2. $IMPL$ est un IOSTS contrôlable, c'est-à-dire qu'il possède les deux propriétés suivantes : la propriété de suites bornées d'actions internes, qui assure que le système ne peut pas boucler indéfiniment par des actions non observables, et celle d'absence de deadlock interne, qui assure qu'on peut toujours observer, grâce à une émission, la fin d'une exécution interne. Il est facile d'observer que si un IOSTS possède la première de ces deux propriétés, son enrichissement par la quiescence possède également la seconde. On considère donc que le système sous test peut être modélisé par un IOSTS possédant la propriété de suites bornées d'actions internes et enrichi par la quiescence.
3. Le système sous test est complet en entrée : en tout état, il accepte de recevoir n'importe quelle valeur sur n'importe quel canal.

Ces hypothèses sont formalisées de la façon suivante.

DÉFINITION VIII.14 — *Modèle du système sous test.*

Un *modèle du système sous test* est un IOSTS noté $IMPL$ sur une signature Ω_δ qui est l'enrichissement par la quiescence d'un IOSTS $\mathbb{G}$ sur Ω tel que $\mathbb{G}$ a les propriétés suivantes :

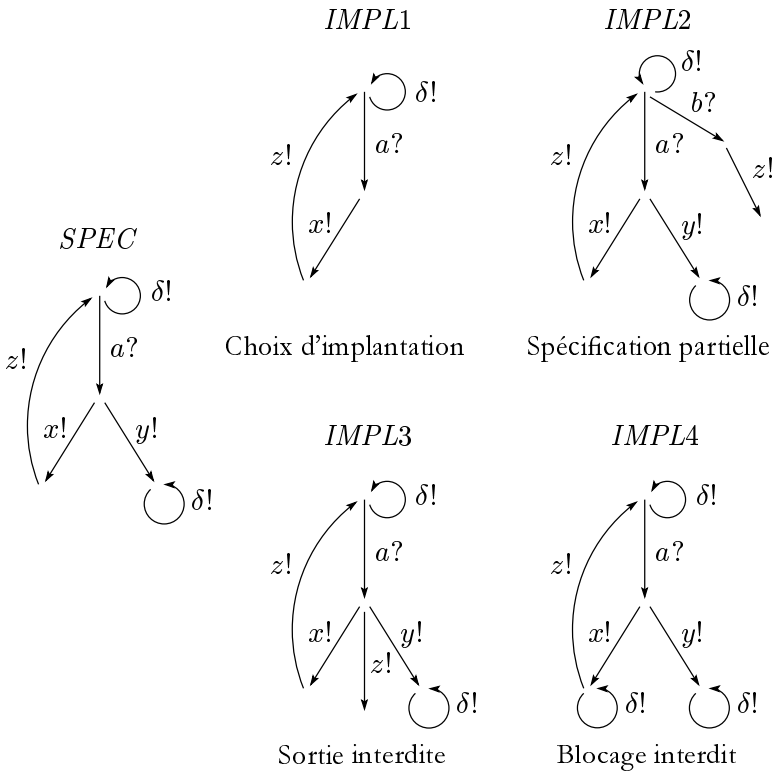
- suites bornées d'actions internes : il existe un entier naturel N tel que pour tout $ch \in CF(\mathbb{G})$, pour toute exécution $r_1 \dots r_m$ de ch, pour tout i, j, $1 \leq i \leq j \leq m$, tels que pour tout k, $i \leq k \leq j$, $act(r_k) = \tau$, on a $j - i \leq N$;
- complétude en entrée : pour toute trace $tr \in Traces(\mathbb{G})$, pour tout $act \in Act_{\mathcal{M}}(\Omega)$ de la forme $c?$ ou $c?a$, $tr \,.\, act \in Traces(\mathbb{G})$.

Sous ces hypothèses de test, il est alors possible de définir la notion de correction d'un système représenté par un IOSTS par rapport à sa spécification elle-même sous la forme d'un IOSTS. Cette relation de correction est appelée conformité. Intuitivement, un système est conforme à sa spécification si l'exécution par le système d'une trace de la spécification (enrichie par la quiescence) déclenche une émission autorisée par la spécification. La définition suivante est une adaptation au cadre des IOSTS de la relation *ioco* définie par Jan Tretmans [Tre92].

DÉFINITION VIII.15 — *Conformité.*

Soit $\Omega = (\Sigma, V, C)$ une IOSTS-signature. Soient $IMPL$ un modèle du système sous test sur Ω_δ et $\mathbb{S}$ un IOSTS sur Ω. $IMPL$ est *conforme* à $\mathbb{S}$ si et seulement si pour toute trace $tr \in Traces(\mathbb{S}_\delta) \cap Traces(IMPL)$, s'il existe $act_\mathcal{M} \in Act_\mathcal{M}(\Omega) \cup \{\delta!\}$ de la forme $c!t$ ou $c!$ telle que $tr \,.\, act_\mathcal{M} \in Traces(IMPL)$, alors $tr \,.\, act_\mathcal{M} \in Traces(\mathbb{S}_\delta)$.

Selon cette définition, une spécification peut être partielle sur les entrées : tous les comportements sont autorisés à l'implantation après une réception qui n'aurait pas été spécifiée. Cela correspond à la notion habituelle de correction selon laquelle un système fait au moins ce qu'il est censé faire mais peut également en faire plus. La figure suivante, due à Thierry Jéron [Jér01], illustre les différents aspects de la relation de conformité. Les implantations $IMPL\,1$ et $IMPL\,2$ sont conformes à la spécification : les sorties de $IMPL\,1$ après la réception sur le canal a sont incluses dans celles de $SPEC$ et les entrées de la spécification en l'état initial sont incluses dans celles de $IMPL\,2$. Au contraire, les implantations $IMPL\,3$ et $IMPL\,4$ ne sont pas conformes à la spécification : la sortie sur le canal z après la réception sur le canal a n'est pas autorisée par la spécification ainsi que le blocage après l'émission sur le canal x.



4.3 Modélisation de la relation de conformité dans notre cadre

De façon à pouvoir comparer les notions de correction et de conformité, il nous faut nous placer dans un formalisme unique, c'est-à-dire associer des spécifications logiques aux IOSTS ou associer des modèles sous la forme d'IOSTS aux spécifications modales. Nous choisissons ici de mener la comparaison en décrivant le modèle des IOSTS à l'aide d'une spécification exprimée en logique modale. Il s'avère en effet très difficile

de poser des restrictions sur les spécifications modales de façon à ce que les modèles qui leur sont associés soient des IOSTS. Il est même impossible de caractériser une classe de modèles qui seraient uniquement des IOSTS, une spécification axiomatique admettant nécessairement parmi ses modèles des modèles infinis.

Une façon simple de comparer les deux approches est alors de montrer que deux IOSTS vérifiant la relation de conformité sont des modèles d'une même spécification modale : si on note $SPEC$ l'IOSTS de la spécification, celui de l'implantation étant noté $IMPL$, si on note Sp la spécification modale associée à $SPEC$, alors on veut montrer que si $IMPL$ est conforme à $SPEC$, alors $IMPL$ et $SPEC$ sont tous deux des modèles de Sp.

Il nous faut pour cela capturer la relation de conformité dans la spécification modale que nous allons associer à $SPEC$. Plusieurs aspects de cette relation sont alors à prendre en compte : l'observabilité d'une part, les branchements et le non déterminisme en sortie d'autre part. Dans le cadre des IOSTS, la notion de déterminisme est la suivante [Rus06].

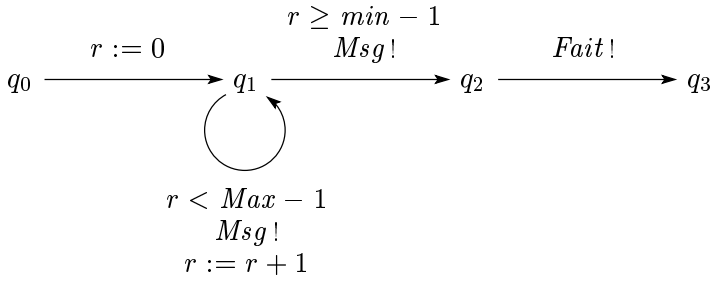
DÉFINITION VIII.16 — *Déterminisme.*

Soient $\Omega = (\Sigma, V, C)$ une IOSTS-signature et $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \mathrm{Id})$ un IOSTS sur Ω. On dit que $\mathbb{G}$ est *déterministe* si et seulement si :

- il ne contient pas d'actions internes τ ;
- il admet au plus un état initial : la fonction d'initialisation est totale et les termes qu'elle associe aux variables d'état sont clos.
- pour tout $q \in \mathbb{Q}$, pour toute paire de transitions distinctes $(q, \varphi_1, act_1, \varsigma_1, q_1)$ et $(q, \varphi_2, act_2, \varsigma_2, q_2)$ étiquetées par une action sur le même canal c, la conjonction des gardes $\varphi_1 \wedge \varphi_2$ n'est pas satisfiable.

EXEMPLE VIII.5 — *Non déterminisme.*

Le système représenté ici permet d'envoyer un message au moins min fois et au plus Max fois, où min et Max sont des constantes telles que $0 < min \leq Max$. Il est non déterministe car les deux gardes $r < Max$ et $r \geq min$ peuvent être satisfaites pour une même valeur de r.



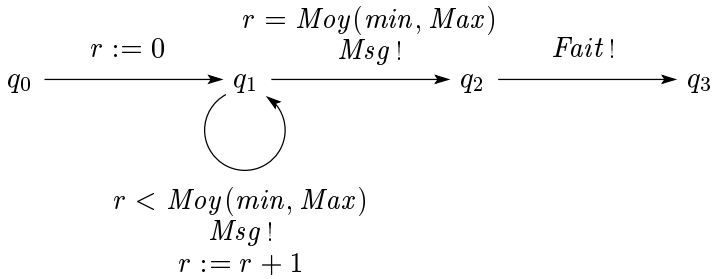
◇

La relation de conformité est fondée sur la comparaison des sorties possibles du système sous test, après une trace issue de la spécification, avec les sorties prévues par cette spécification. La comparaison ne portant que sur les actions observables, émissions et réceptions, la spécification modale ne doit également porter que sur les communications avec l'extérieur. Il est alors inutile de spécifier les changements internes de l'état tels que les substitutions de variables d'état. D'autre part, les sorties observées après une trace de la spécification doivent être incluses dans les sorties attendues, tandis que les entrées peuvent être plus

nombreuses. Il est donc nécessaire de prendre en compte le fait qu'une implantation conforme à $SPEC$ peut valider les mêmes axiomes que $SPEC$ tout en ayant moins de sorties et plus d'entrées. De plus, si $SPEC$ est non déterministe, il nous faut prendre en compte le fait qu'une implantation qui lui est conforme peut résoudre ce non déterminisme tout en n'ayant pas simplement sélectionné une branche mais en en ayant combiné plusieurs. Nous allons voir par la suite de quelle façon nous allons procéder pour construire une spécification prenant en compte ces différents aspects.

EXEMPLE VIII.6 — *Résolution du non déterminisme.*

Dans l'IOSTS suivant, *Moy* est une opération de $Nat \times Nat$ dans Nat qui donne la moyenne de deux entiers, arrondie à l'entier supérieur. Cet IOSTS est bien conforme à celui de l'exemple VIII.5, puisqu'il envoie le message exactement $Moy(min, Max)$ fois, si $0 < min \leq Max$. Cependant, il ne contient aucune des deux transitions de la spécification.



◇

Il nous faut tout d'abord fixer le langage que nous allons utiliser pour décrire un IOSTS donné. La signature modale qui va être associée à un IOSTS doit permettre d'écrire des propriétés sur cet IOSTS, aussi bien en termes de données que de dynamique. La partie données de l'IOSTS est déjà spécifiée en logique du premier ordre dans l'IOSTS-même. Spécifier la partie dynamique requiert la définition d'une sorte non observable représentant l'état ainsi que des observateurs et des prédicats sur cette sorte. Nous posons donc une sorte non observable nommée $\acute{E}tat$ sur laquelle nous allons définir comme observateurs les émissions et les réceptions ainsi que les variables d'état. Les communications avec l'environnement sont définies comme étant des observateurs de sorte non observable puisqu'elles permettent de faire évoluer le système d'un état vers un autre. On aura alors, dans le cas de l'IOSTS de l'exemple VIII.3, deux observateurs de sorte non observable : $D\acute{e}but? : \acute{E}tat \times Nat \to \acute{E}tat$ correspondant à la réception d'une valeur de sorte Nat sur le canal $D\acute{e}but$, $Msg! : \acute{E}tat \times Nat \to \acute{E}tat$ correspondant à l'émission d'une valeur de sorte Nat sur le canal Msg et $Stop! : \acute{E}tat \to \acute{E}tat$ correspondant à l'émission d'un signal sur le canal $Stop$. Les variables d'état, quant à elles, sont des observateurs de sorte observable, qui permettent d'observer une caractéristique de l'état. Pour le même IOSTS, la variable d'état x devient un observateur de sorte observable Nat, $x : \acute{E}tat \to Nat$. On se donne de plus un observateur de sorte observable associant à chaque état son nom dans l'ensemble d'états $\mathbb{Q}$, $\acute{e}tat : \acute{E}tat \to \mathbb{Q}$. Cet observateur permet de parler d'un état avec son historique sans avoir besoin de préciser celui-ci. On décrit ainsi de façon précise les comportements possibles de chaque état de l'IOSTS.

Le système sous test n'étant observable qu'au travers de ses émissions et de ses réceptions, il suffit de spécifier les communications. Pour observer une communication, on a besoin d'un prédicat sur l'état cible

de la transition indiquant que la communication a effectivement eu lieu. On se donne alors des prédicats sur la sorte $\acute{E}tat$ permettant de vérifier que la réception ou l'émission sur un certain canal d'une valeur donnée a eu lieu. Pour spécifier l'IOSTS de l'exemple VIII.3, on aura alors trois prédicats, un pour chaque action de communication : $Recu\_D\acute{e}but : \acute{E}tat \times Nat$ permet d'assurer qu'une valeur de sorte Nat a bien été reçue sur le canal $D\acute{e}but$, $\acute{E}mis\_Msg : \acute{E}tat \times Nat$ assure qu'une valeur de sorte Nat a été envoyée sur le canal Msg et $\acute{E}mis\_Stop : \acute{E}tat$ assure qu'un signal a été émis sur le canal $Stop$. Chaque action de communication sera alors représentée par une formule modale spécifiant le fait qu'après une cette action, le résultat de cette action est observé. Par exemple, on écrira $\langle\, D\acute{e}but?x \rangle\, Recu\_D\acute{e}but(x)$ de façon à exprimer le fait qu'après l'action $D\acute{e}but?x$, une réception a eu lieu sur le canal $D\acute{e}but$ et la valeur reçue est celle de x. De même, on écrira $\langle\, Stop! \rangle\, \acute{E}mis\_Stop$ pour assurer qu'après l'action $Stop!$, un signal a bien été émis sur le canal $Stop$.

Spécifier un IOSTS en logique modale revient alors à associer, à chaque état de l'IOSTS, l'ensemble des émissions et des réceptions qui sont possibles depuis cet état. Pour que $IMPL$ soit également un modèle de la spécification modale, plusieurs aspects de la relation de conformité sont à prendre en compte : en un état accessible par une trace de $SPEC$, $IMPL$ peut admettre moins de sorties que $SPEC$, mais doit en avoir au moins une, et ne peut pas en admettre plus que celles qui sont spécifiées ; en ce même état, il peut admettre plus d'entrées puisque la spécification peut être partielle, mais il doit admettre au moins les entrées spécifiées. Cet état doit alors valider la disjonction des sorties d'une part et la conjonction des entrées d'autre part. On va associer à chaque état q un axiome de la forme :

$$\acute{e}tat = q \Rightarrow \left(\bigvee_{e_i \text{ émission}} \varphi_i \Rightarrow (e_i \wedge \acute{e}tat = q_i) \right) \wedge \left(\bigwedge_{r_i \text{ réception}} \psi_i \Rightarrow (r_i \wedge \acute{e}tat = q_i) \right)$$

où les $(q, \varphi_i, e_i, \varsigma_i, q_i)$ sont transitions étiquetées par des émissions et les $(q, \psi_i, r_i, \varsigma_i, q_i)$ sont les transitions étiquetées par des réceptions. La spécification de l'IOSTS de l'exemple VIII.3 aura alors deux axiomes, un associé à q_0 et l'autre à q_1. Celui associé à q_0 dit qu'en cet état, il est possible de recevoir une valeur sur le canal $D\acute{e}but$, sans condition :

$$\acute{e}tat = q_0 \Rightarrow \langle\, D\acute{e}but?x \rangle (Recu\_D\acute{e}but(x) \wedge \acute{e}tat = q_1)$$

L'axiome associé à q_1 spécifie que deux émissions sont possibles en cet état : l'émission de la valeur de x sur le canal Msg si x est strictement positif, l'émission d'un signal sur le canal $Stop$ si x est nul.

$$\begin{aligned} \acute{e}tat = q_1 \Rightarrow \big(x > 0 \Rightarrow \langle\, Msg!x \rangle (\acute{E}mis\_Msg(x) \wedge \acute{e}tat = q_1)\big) \\ \vee \big(x = 0 \Rightarrow \langle\, Stop! \rangle (\acute{E}mis\_Stop \wedge \acute{e}tat = q_2)\big) \end{aligned}$$

De plus, on doit regrouper les émissions qui ont lieu sur un même canal de façon, en particulier, à prendre en compte le non déterminisme. Revenons à l'exemple VIII.5 et à son implantation possible de l'exemple VIII.6. Pour assurer que ces deux IOSTS valident le même axiome en l'état q_1, il ne faut pas distinguer les transitions portant une émission sur le même canal Msg. On écrira alors :

$$\begin{aligned} \acute{e}tat = q_1 \Rightarrow \big(r < Max \vee r \geq min \Rightarrow \langle\, Msg! \rangle (\acute{E}mis\_Msg \wedge \acute{e}tat = q_1) \\ \vee \langle\, Msg! \rangle (\acute{E}mis\_Msg \wedge \acute{e}tat = q_2)\big) \end{aligned}$$

De manière générale, on associe une spécification modale à un IOSTS de la façon suivante.

DÉFINITION VIII.17 — *Spécification modale d'un IOSTS.*

Soient $\Omega = (\Sigma, V, C)$ une IOSTS-signature avec $\Sigma = (S, F, R)$ et $\mathbb{G} = (\mathbb{Q}, q_0, \mathbb{T}, \iota)$ un IOSTS sur Ω. On note $\Sigma_{\mathbb{G}} = (S_{\mathbb{G}}, F_{\mathbb{G}}, R_{\mathbb{G}})$ la signature modale où :

- $S_{\mathbb{G}} = S_{obs} \amalg T$ avec $S_{obs} = S$ et T contient l'unique sorte non observable $\acute{E}tat$;
- $F_{\mathbb{G}} = F_{obs} \amalg F_{\acute{E}tat}$ avec $F_{obs} = F$ et $F_{\acute{E}tat}$ l'ensemble des observateurs de la sorte $\acute{E}tat$ défini par :

$$\begin{aligned} F_{\acute{E}tat} = \ & \{c? : \acute{E}tat \to \acute{E}tat \mid c \in C^?_\varepsilon\} \\ & \cup \{c! : \acute{E}tat \to \acute{E}tat \mid c \in C^!_\varepsilon\} \\ & \cup \{c? : \acute{E}tat \times s \to \acute{E}tat \mid c \in C^?_s\} \\ & \cup \{c! : \acute{E}tat \times s \to \acute{E}tat \mid c \in C^!_s\} \\ & \cup \{x : \acute{E}tat \to s \mid x \in V_s\} \\ & \cup \{\acute{e}tat : \acute{E}tat \to \mathbb{Q}\} \end{aligned}$$

- $R_{\mathbb{G}} = R_{obs} \amalg R_{\acute{E}tat}$ avec $R_{obs} = R$ et $R_{\acute{E}tat}$ l'ensemble des prédicats sur la sorte $\acute{E}tat$ défini par :

$$\begin{aligned} R_{\acute{E}tat} = \ & \{Recu\_c : \acute{E}tat \mid c \in C^?_\varepsilon\} \\ & \cup \{\acute{E}mis\_c : \acute{E}tat \mid c \in C^!_\varepsilon\} \\ & \cup \{Recu\_c : \acute{E}tat \times s \mid c \in C^?_s\} \\ & \cup \{\acute{E}mis\_c : \acute{E}tat \times s \mid c \in C^!_s\} \end{aligned}$$

La *spécification modale de* $\mathbb{G}$, notée $Sp_{\mathbb{G}}$, est le couple $(\Sigma_{\mathbb{G}}, Ax_{\mathbb{G}})$ où $Ax_{\mathbb{G}}$ est l'ensemble de formules construit comme suit.

Pour tout état $q \in \mathbb{Q}$, on note [1]

$$tr_1, \ldots, tr_M, \ldots, tr_N, \ldots, tr_P, \ldots, tr_Q$$

les transitions sortantes de q, avec

$$tr_1, \ldots, tr_{k_1}, \ldots, tr_{k_2}, \ldots, tr_{k_m} = tr_M$$

et

$$tr_{k_m+1}, \ldots, tr_{k_{m+1}}, \ldots, tr_{k_{m+2}}, \ldots, tr_{k_n} = tr_N$$

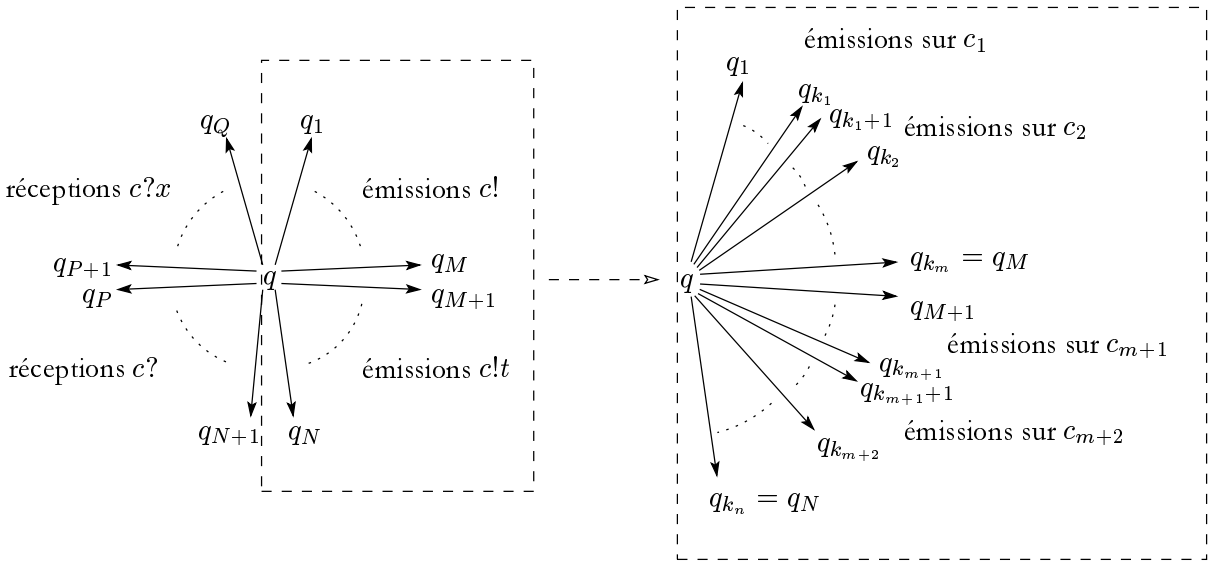
Pour tout i, $1 \leq i \leq Q$, la transition tr_i est le quintuplet $(q, \varphi_i, act_i, \varsigma_i, q_i)$, avec :

- pour tout i, $1 \leq i \leq m$, pour tout j, $k_{i-1} < j \leq k_i$, $act_j = c_i!$ $(k_0 = 0)$;
- pour tout i, $m < i \leq n$, pour tout j, $k_{i-1} < j \leq k_i$, $act_j = c_i!t_j$;
- pour tout i, $N < i \leq P$, act_i est de la forme $c_i?$;
- pour tout i, $P < i \leq Q$, act_i est de la forme $c_i?x_i$,

[1] Voir le schéma page suivante.

On associe alors à l'état q la formule suivante :

$$
\begin{aligned}
\mathit{état} = q \Rightarrow \Bigg(& \bigvee_{1\le i\le m} \Big(\bigvee_{k_{i-1}<j\le k_i} \varphi_j \Rightarrow \langle\, c_i! \rangle (\mathit{Émis}\_c_i \wedge \mathit{état} = q_j) \Big) \\
& \vee \bigvee_{m<i\le n} \Big(\bigvee_{k_{i-1}<j\le k_i} \varphi_j \Rightarrow \bigvee_{k_{i-1}<j\le k_i} \langle\, c_i! t_j \rangle (\mathit{Émis}\_c_i(t_j) \wedge \mathit{état} = q_j) \Big) \Bigg) \\
& \wedge \bigwedge_{N<i\le P} \Big(\varphi_i \Rightarrow \langle\, c_i? \rangle (\mathit{Recu}\_c_i \wedge \mathit{état} = q_i) \Big) \\
& \wedge \bigwedge_{P<i\le Q} \Big(\varphi_i \Rightarrow \langle\, c_i? x_i \rangle (\mathit{Recu}\_c(x_i) \wedge \mathit{état} = q_i) \Big)
\end{aligned}
$$



Au vu de la construction de $Sp_{\mathbb{G}}$, il est simple de vérifier que $\mathbb{G}$ est un modèle de cette spécification. Nous donnons par la suite quelques exemples qui illustrent chacun des aspects de la construction.

EXEMPLE VIII.7 — *Compte à rebours.*

On récapitule ici l'exemple développé au cours du texte précédant la définition VIII.17.

spec REBOURS =

type $Nat ::= 0 \mid s(Nat)$

cotype $\mathit{État}$

op $\_ - \_ : Nat \times Nat \to Nat$

obs $D\acute{e}but? : \acute{E}tat \times Nat \to \acute{E}tat$
$Msg! : \acute{E}tat \times Nat \to \acute{E}tat$
$Stop! : \acute{E}tat \to \acute{E}tat$
$x : \acute{E}tat \to Nat$
$\acute{e}tat : \acute{E}tat \to \mathbb{Q}$
preds $= : Nat \times Nat$
$> : Nat \times Nat$
predobs $Recu\_D\acute{e}but : \acute{E}tat \times Nat$
$\acute{E}mis\_Msg : \acute{E}tat \times Nat$
$\acute{E}mis\_Stop : \acute{E}tat$
- $\acute{e}tat = q_0 \Rightarrow \langle D\acute{e}but?x \rangle (Recu\_D\acute{e}but(x) \wedge \acute{e}tat = q_1)$
- $\acute{e}tat = q_1 \Rightarrow \big(x > 0 \Rightarrow \langle Msg!x \rangle (\acute{E}mis\_Msg(x) \wedge \acute{e}tat = q_1)\big)$
$\vee \big(x = 0 \Rightarrow \langle Stop! \rangle (\acute{E}mis\_Stop \wedge \acute{e}tat = q_2)\big)$

end

◇

EXEMPLE VIII.8 — *Non déterminisme.*

La spécification modale associée à l'IOSTS de l'exemple VIII.5 est la suivante. Les constantes *min* et *Max* sont déclarées dans la partie fixe de la signature. Comme on l'a précisé plus haut, on regroupe les émissions qui ont lieu sur le même canal de façon à prendre en compte le non déterminisme.

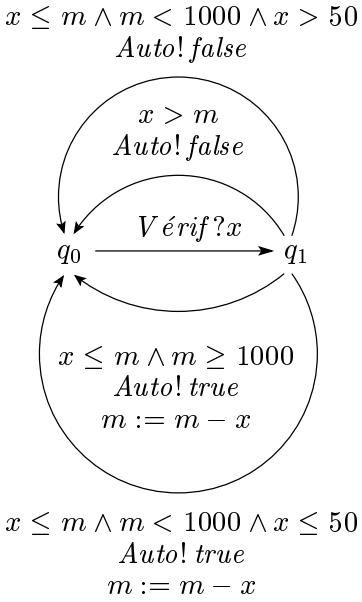
spec MESSAGESNONDET =
type $Nat ::= 0 \mid s(Nat)$
cotype $\acute{E}tat$
ops $\_ + \_ : Nat \times Nat \to Nat$
$min : \to Nat$
$Max : \to Nat$
obs $Msg! : \acute{E}tat \to \acute{E}tat$
$Fait! : \acute{E}tat \to \acute{E}tat$
$r : \acute{E}tat \to Nat$
$\acute{e}tat : \acute{E}tat \to \mathbb{Q}$
preds $< : Nat \times Nat$
$\geq : Nat \times Nat$
predobs $\acute{E}mis\_Msg : \acute{E}tat$
$\acute{E}mis\_Fait : \acute{E}tat$
- $\acute{e}tat = q_1 \Rightarrow \Big((r < Max \vee r \geq min)$
$\Rightarrow \langle Msg! \rangle (\acute{E}mis\_Msg \wedge \acute{e}tat = q_1) \vee \langle Msg! \rangle (\acute{E}mis\_Msg \wedge \acute{e}tat = q_2)\Big)$
- $\acute{e}tat = q_2 \Rightarrow \langle Fait! \rangle (\acute{E}mis\_Fait \wedge \acute{e}tat = q_3)$

end

◇

EXEMPLE VIII.9 — *Autorisation de retrait.*

On spécifie ici un système dont le but est de donner l'autorisation au client d'une banque de retirer une somme d'argent donnée. Pour donner cette autorisation, le système procède à différentes vérifications. Le retrait n'est pas accepté si le montant demandé x est supérieur au montant m disponible sur le compte du client. Lorsque le montant est disponible, l'autorisation de retrait est accordée directement si le client dispose de plus de 1000 euros sur son compte. S'il dispose de moins de 1000 euros, il n'a le droit de retirer que des sommes inférieures à 50 euros.



De la même manière que dans les exemples précédents, on se donne une sorte non observable $\acute{E}tat$ sur laquelle on déclare les observateurs $Vérif?$ et $Auto!$ de sorte non observable et les observateurs x et m de sorte observable. On déclare de plus deux prédicats sur la sorte $\acute{E}tat$ permettant d'assurer que la réception sur le canal $Vérif$ et l'émission sur le canal $Auto$ ont bien eu lieu. On associe un axiome à chacun des deux états de l'IOSTS. En q_0, on spécifie la réception d'une valeur sur le canal $Vérif$. En q_1, on regroupe les émissions ayant lieu sur le même canal, c'est-à-dire ici toutes les émissions sur le canal $Auto$. Cet axiome peut paraître trivial, mais il est impossible de faire mieux à un niveau syntaxique. En effet, reconnaître que *true* est différent de *false* ne peut se faire qu'à un niveau sémantique, lorsqu'on a décidé d'un modèle pour interpréter les données. Avec cette distinction, il serait possible de séparer les conditions qui mènent à l'émission de *true* de celles qui mènent à l'émission de *false*.

spec AUTORISATION =

types $Nat ::= 0 \mid s(Nat)$

$Bool ::= true \mid false$

cotype $\acute{E}tat$

op $\_ - \_ : Nat \times Nat \to Nat$

obs $Vérif? : État \times Nat \to État$
$Auto! : État \times Bool \to État$
$x : État \to Nat$
$m : État \to Nat$
$état : État \to \mathbb{Q}$
preds $< : Nat \times Nat$
$\leq : Nat \times Nat$
$> : Nat \times Nat$
$\geq : Nat \times Nat$
predobs $Recu\_Vérif : État \times Nat$
$Émis\_Auto : État \times Bool$

- $état = q_0 \Rightarrow \langle Vérif?x \rangle (Recu\_Vérif(x) \wedge état = q_1)$
- $état = q_1 \Rightarrow \Big(x > m \vee \big(x \leq m \wedge m < 1000 \wedge x > 50\big)$
$\vee \big(x \leq m \wedge m \geq 1000\big) \vee \big(x \leq m \wedge m < 1000 \wedge x \leq 50\big)\Big)$
$\Rightarrow \Big(\langle Auto!\, true \rangle (Émis\_Auto(true) \wedge état = q_0)$
$\vee \langle Auto!\, false \rangle (Émis\_Auto(false) \wedge état = q_0)\Big)$

end

◇

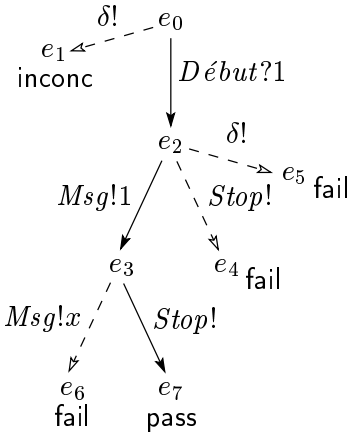
Étant donnée la méthode que nous avons suivie pour associer une spécification modale à un IOSTS, il est maintenant très simple de vérifier qu'un IOSTS conforme à celui-ci est également un modèle la spécification obtenue. Pour s'en donner l'intuition, considérons de nouveau l'IOSTS non déterministe de l'exemple VIII.5, que nous noterons $SPEC$, et son implantation possible donnée à l'exemple VIII.6, que nous noterons $IMPL$. Dans ce deuxième système, un message est émis tant que la valeur de r est inférieure ou égale à la moyenne des constantes min et Max. Cette condition implique la condition $r < Max \vee r \geq min$ du premier axiome de Sp_{SPEC}. De plus, l'émission d'un signal sur le canal Msg est bien possible dans cet état et les états cibles des deux transitions possibles sont les états q_1 et q_2. Le système modélisé par $IMPL$ valide donc le premier axiome de Sp_{SPEC}. Le deuxième axiome est trivialement vérifié.

4.4 Objectifs de test

Dans l'approche du test de conformité, l'ensemble des comportements à tester sur le système est l'ensemble des comportements représentés par la spécification de ce système enrichie par la quiescence. Cet ensemble étant la plupart du temps infini, il est nécessaire de sélectionner un nombre fini de ces comportements considéré comme représentatifs [JJRZ05]. Un objectif de test, dans ce cadre, sert à identifier un certain nombre de chemins de la spécification enrichie par la quiescence. Chaque chemin est considéré comme une classe de traces.

Il existe plusieurs méthodes pour écrire des objectifs de test. Ils peuvent par exemple être sélectionnées manuellement [JJRZ05] ou à l'aide de techniques d'exécution symboliques [Tou06, GLRT06]. Dans ce dernier

cas, un objectif de test est un arbre enraciné en q_0, dont les transitions sont étiquetées uniquement par des actions de communication et dont les feuilles sont étiquetées par les différents verdicts possibles : pass si le chemin menant à cette feuille est le comportement à tester ; fail si le chemin menant à cette feuille ne doit pas se trouver dans l'implantation ; inconc si le chemin menant à cette feuille est accepté par la spécification mais ne correspond pas au comportement que l'on voulait tester. On veut par exemple tester un système à partir de la spécification donnée à l'exemple VIII.3. Un système implantant cette spécification est censé compter à rebours à partir de la valeur qui lui aura été donnée en entrée. Un exemple simple de comportement qu'on peut vouloir tester est celui par exemple qui consiste à recevoir la valeur 1 sur le canal $Début$, puis à l'émettre sur le canal Msg et enfin à émettre un signal sur le canal $Stop$. Ce comportement est représenté par l'objectif de test suivant.



Si on ne donne aucune valeur en entrée au système, il est autorisé par la spécification à être quiescent, mais ce n'est pas le comportement que l'on voulait tester, d'où le verdict inconclusif. S'il est quiescent après avoir reçu la valeur 1, il échoue au test puisqu'il ne peut être quiescent en cet état que si la valeur reçue est strictement négative. De même, il ne peut pas émettre sur le canal $Stop$ en cet état puisque la valeur reçue n'est pas nulle. Après avoir émis la valeur 1, il doit émettre un signal sur le canal $Stop$. Aucun autre comportement n'est autorisé.

Le verdict inconclusif permet en particulier de gérer le non déterminisme de la spécification en prévoyant tous les branchements autorisés même si ce ne sont pas ceux visés par l'objectif de test. Si on voulait écrire ce même objectif de test dans notre cadre modal, on obtiendrait la formule suivante :

$$\langle Début?1 \rangle \langle Msg!1 \rangle \langle Stop! \rangle Émis\_Stop$$

On remarque que le verdict inconclusif n'existe pas dans notre cadre, un objectif de test sous la forme d'une formule représentant exactement ce qu'on veut tester. Si le système sous test « dévie » de cet objectif, parce qu'il est non déterministe par exemple, il est impossible de dire à partir de cette formule unique si le comportement observé correspond ou non à celui prévu par la spécification. Pour représenter un objectif de test tel que celui décrit par l'arbre ci-dessus, il faudrait un ensemble de formules qui représenteraient chacune un chemin de l'objectif, ce qui revient à n'avoir que des objectifs de test sous la forme de chemins. On perd de fait la structure d'arbre, c'est-à-dire la possibilité d'évaluer une exécution qui dévierait de l'objectif. D'autre part, perdre la structure d'arbre empêche de gérer de façon efficace le non déterminisme. En effet,

en cas de non déterminisme, il nous faut soumettre le même test plusieurs fois pour arriver jusqu'au bout du chemin visé et pouvoir conclure.

4.5 Bilan

Nous avons, au cours de cette section, tenté de rapprocher les cadres de test du test de conformité et du test à partir de spécifications axiomatiques. Nous avons pris le parti d'associer à un IOSTS une spécification exprimée en logique modale afin qu'un IOSTS qui soit conforme à celui-ci soit également un modèle de cette spécification modale. Par cette traduction, nous constatons que notre approche est plus abstraite, et donc également moins fine, que celle du test de conformité. En effet, des IOSTS qui sont différenciés par la relation de conformité comme étant la spécification et l'implantation sont, dans notre cadre, deux implantations d'une même spécification. La relation de conformité permet donc de différencier des comportements que notre notion de correction confond. Ceci est en partie dû à la présence du « ou » logique dans les axiomes de la spécification modale. En effet, pour valider une disjonction de formules, il suffit de valider une de ces formules. Lorsque la disjonction est validée, il est impossible de dire *a priori* quelle(s) sous-formule(s) a (ont) été validée(s). Dans le cas des axiomes spécifiant un IOSTS, les sous-formules de la disjonction représentent des transitions, il est donc impossible de dire quelle(s) transition(s) a (ont) été prise(s) lorsque la disjonction est validée.

La façon dont nous avons abordé la comparaison des deux approches, par l'association d'une spécification modale à un IOSTS, n'est qu'une des pistes possibles pour cette étude. Elle pourrait également être menée à un autre niveau, pas nécessairement de façon formelle. Il serait par exemple certainement très fructueux de mener une étude de cas selon ces deux approches du test. Il serait possible, de cette manière, de constater dans la pratique la différence d'expressivité des deux langages ainsi que la facilité d'écriture des objectifs de test et de la décision des verdicts. On peut penser que les objectifs de test exprimés sous la forme de formules logiques seraient plus proches de l'idée qu'on aurait des propriétés à tester, tandis que les tests issus des IOSTS seraient plus simples à concrétiser, étant eux-mêmes exécutables.

De façon plus générale, l'abstraction des spécifications axiomatiques constitue leur principal avantage. Un système peut être testé à partir d'une même spécification axiomatique initiale à toutes les étapes de la conception. En effet, les propriétés décrites par une telle spécification sont suffisamment abstraites pour que toutes les implantations de cette spécification, quel que soit leur niveau de détail, les vérifient. D'autre part, l'avantage majeur de l'approche à base de systèmes de transitions est d'être plus proche de l'implantation, ces modèles étant eux-mêmes exécutables. Les tests sous forme de chemins étant plus proches des exécutions réelles du système sous test, la concrétisation des tests en est simplifiée, ainsi que le problème de l'oracle.

Cette étude comparative n'est bien entendu qu'une ébauche préliminaire à un travail plus approfondi de mise en relation point à point des notions sur lesquelles reposent la théorie du test.

Conclusion

Nous avons proposé, sur la base de la théorie du test définie pour des systèmes décrits à l'aide de spécifications axiomatiques, des extensions des résultats d'exhaustivité et de la méthode de sélection par dépliage des axiomes. Nous avons ainsi proposé une approche nouvelle au test de systèmes dynamiques.

La sélection de tests a été initialement étudiée de le cadre des spécifications équationnelles conditionnelles positives. Bien que la propriété d'exhaustivité ait été reconnue comme primordiale pour débuter la phase de sélection, elle n'avait pas été étudiée de façon approfondie dans les premiers travaux sur la sélection. Nous avons alors, dans chacun des formalismes logiques auxquels nous nous sommes intéressés, effectué deux tâches. Nous avons d'une part étudié les conditions nécessaires à imposer à la spécification et au système sous test pour obtenir l'exhaustivité du jeu de tests composé des conséquences sémantiques observables de la spécification. Nous avons d'autre part adapté et étendu la procédure de sélection par dépliage des axiomes à ces formalismes et montré sa correction et sa complétude. Dans les deux cadres généraux des spécifications du premier ordre sans quantificateurs et des spécifications modales du premier ordre sans quantificateurs, nous avons montré que les conditions nécessaires à l'exhausitivité du jeu de test visé étaient mineures car faciles à assurer dans la pratique, ce qui assure une généralisation satisfaisante de la sélection dans ce cadre.

L'approche habituellement suivie pour tester des systèmes dynamiques est celle du test de conformité, où le système sous test est spécifié à l'aide d'un automate ou d'un système de transitions. Ces formalismes de spécification permettent une description synthétique de la dynamique d'un système. Cependant, étant elle-même exécutable, cette description est assez proche de l'implantation. L'approche que nous proposons, fondée sur les formalismes axiomatiques, permet alors de gagner en abstraction dans la spécification des systèmes. L'étude du rapprochement de notre cadre de test avec celui du test de conformité que nous avons menée au chapitre VII montre que le processus global de test profite également de cette abstraction. Nous avons également constaté que des objectifs de test plus abstraits sont aussi moins précis, en particulier en ce qui concerne les différents verdicts qu'on peut vouloir leur associer. Contrairement au test de conformité, nous ne sommes pas capable d'accepter partiellement un comportement déviant de l'objectif de test. D'autre part, nous avons évoqué la difficulté, dans notre cadre, de générer des cas de test à partir de nos objectifs de test lorsque la spécification n'est pas exécutable. Un compromis entre l'abstraction des propriétés à tester et la facilité de leur soumission serait alors à étudier.

Ce travail ouvre également la voie vers d'autres perspectives.

Dans le processus de test d'un système, l'étape suivant la définition de critères de sélection est la génération d'un jeu de tests effectif à soumettre au système. Dans notre cadre, la génération consiste à

appliquer l'hypothèse d'uniformité aux jeux de test contraints obtenus par dépliage d'un objectif de test initial. Il s'agit en fait, pour chacun des objectifs de test contraints, de résoudre les contraintes qui lui sont associées afin de calculer un ou plusieurs cas de test correspondant à cet objectif. La résolution de ces contraintes peut s'avérer délicate. En effet, même dans le cas simple des spécifications conditionnelles positives, elle est impossible si l'égalité n'est pas décidable, c'est-à-dire si on ne dispose pas d'un moyen effectif pour résoudre les équations. Pour assurer la possibilité de résoudre ces contraintes, une solution consiste à imposer que les spécifications considérées soient exécutables, c'est-à-dire qu'elles correspondent directement à un programme. C'est la condition imposée dans l'outil LOFT [Mar95] pour lequel la méthode de sélection par dépliage a été implantée : les axiomes conditionnels positifs doivent pouvoir être transformés en règles de réécriture telles que le système de réécriture obtenu soit confluent et réducteur [AABLM05].

Dans le cas des spécifications du premier ordre sans quantificateurs, une condition similaire de décidabilité des spécifications est nécessaire pour résoudre les contraintes associées aux jeux de tests. Cependant, assurer cette condition est bien plus complexe que dans le cas conditionnel positif. Il en est de même dans le cadre des logiques modales. Il serait alors intéressant d'étudier les conditions à imposer aux spécifications pour assurer qu'elles permettent de résoudre les contraintes générées par le dépliage.

Le travail présenté ici est une première étape vers la définition d'un cadre formel de sélection de tests pour les langages de spécification CASL et COCASL. En effet, la logique sous-jacente au langage de spécification algébrique CASL est la logique du premier ordre et celle sous-jacente à son extension coalgébrique COCASL est la logique modale du premier ordre que nous avons présentée. Le prolongement de ce travail en vue de prendre en compte les caractéristiques particulières de chacun de ces langages pourrait s'établir comme suit :

- il s'agirait dans un premier temps de traiter le prédicat d'égalité de façon spécifique. En effet, dans les logiques que nous avons présentées, l'égalité est définie comme un prédicat quelconque ce qui fait perdre l'avantage de ses propriétés en tant que relation de congruence. Il faudrait alors spécialiser les calculs des séquents pour chacune de ces logiques en ajoutant des règles d'inférence dédiée au traitement de l'égalité. La procédure de sélection en serait alors plus efficace ;
- dans un deuxième temps, il faudrait prendre en compte l'existence de sous-sortes et de fonctions partielles dans CASL et COCASL [GM92, CMR99]. Cela nécessiterait de traiter la relation de sous-sortes ainsi que le prédicat de définissabilité et l'égalité existentielle. L'égalité existentielle est vérifiée par deux termes seulement s'ils sont définis et qu'ils sont interprétés par la même valeur, contrairement à l'égalité forte (le cas habituel) qui est vérifiée soit parce que les termes ne sont pas définis, soit parce qu'ils sont définis et interprétés par la même valeur ;
- à un niveau plus général, les langages CASL et COCASL incluent des primitives de structuration de spécifications. Il est possible de renommer, de réduire ou d'enrichir une spécification, par exemple, ou de faire l'union de plusieurs spécifications. Le test d'intégration, c'est-à-dire le test de la composition de plusieurs modules ou unités de programme, relève d'une problématique différente de celle du test unitaire où on ne teste qu'un module. Il s'agit, à partir de modules déjà testés, de dégager les propriétés qui sont nées de la composition de ces modules mais qui n'apparaissaient pas dans les modules individuels. Il faudrait alors étudier le test d'intégration au travers des primitives de structuration de CASL et COCASL, afin de proposer des solutions au test de la composition de modules spécifiés dans ces langages. Pour ce faire, nous pourrions nous inspirer des travaux de Patrícia Machado [Mac00, MS02].

Annexes

ANNEXE A

Notions de théorie des catégories

1 Catégories

DÉFINITION A.1 — *Catégorie.*

Une catégorie $\mathcal{C}$ consiste en une classe $\mathcal{C}_o$, notée également $|\mathcal{C}|$, d'*objets* $A, B, C\ldots$ et une classe $\mathcal{C}_m$ de *morphismes* ou *flèches* $f, g, h\ldots$ entre ces objets, munie des opérations suivantes :

- $dom : \mathcal{C}_m \to \mathcal{C}_o$;
- $codom : \mathcal{C}_m \to \mathcal{C}_o$;
- $\mathrm{Id} : \mathcal{C}_o \to \mathcal{C}_m$,

associant à chaque flèche sa source (*domaine*), respectivement sa cible (*codomaine*), et à tout objet A sa *flèche identité* Id_A. De plus, il existe une opération partielle $\circ$ de *composition* de flèches. La composition de f et g est définie dès que $codom(f) = dom(g)$. Le résultat est un morphisme $g \circ f$ avec $dom(g \circ f) = dom(f)$ et $codom(g \circ f) = codom(g)$. Les conditions suivantes doivent être vérifiées lorsque la composition est définie :

- $(h \circ g) \circ f = h \circ (g \circ f)$;
- $\mathrm{Id}_A \circ f = f$ et $g = g \circ \mathrm{Id}_A$.

Pour tout couple d'objets (A, B), on note $Hom_{\mathcal{C}}(A, B)$ l'ensemble des flèches entre A et B dans $\mathcal{C}$.

On définit maintenant la notion de foncteur. Intuitivement, un foncteur est un morphisme de catégories.

DÉFINITION A.2 — *Foncteur.*

Soient $\mathcal{C}$ et $\mathcal{D}$ deux catégories. Un *foncteur* $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ est la donnée d'un couple $(\mathcal{F}_o, \mathcal{F}_m)$ d'applications tel que :

- $\mathcal{F}_o : \mathcal{C}_o \to \mathcal{D}_o$, appelée *fonction objet*, associe à tout objet $C \in \mathcal{C}_o$ un objet $\mathcal{F}(C) \in \mathcal{D}_o$;
- $\mathcal{F}_m : \mathcal{C}_m \to \mathcal{D}_m$, appelée *fonction flèche*, associe à tout morphisme $f : C \to C' \in \mathcal{C}_m$ une flèche $\mathcal{F}(f) : \mathcal{F}(C) \to \mathcal{F}(C') \in \mathcal{D}_m$,

vérifiant les deux propriétés suivantes :

1. $\mathcal{F}_m(\mathrm{Id}_C) = \mathrm{Id}_{\mathcal{F}_o(C)}$;

2. $\mathcal{F}_m(g \circ f) = \mathcal{F}_m(g) \circ \mathcal{F}_m(f)$ (si $g \circ f$ est défini dans $\mathcal{C}$).

Dans la suite, on ne précisera les indices $_o$ et $_m$ que lorsque ce sera nécessaire.

2 Constructions sur les catégories

DÉFINITION A.3 — *Sous-catégorie.*

Soit $\mathcal{C}$ une catégorie. Une sous-catégorie $\mathcal{D}$ de $\mathcal{C}$ est une catégorie telle que :
- tous les objets et tous les morphismes de $\mathcal{D}$ sont des objets et des flèches de $\mathcal{C}$: $\mathcal{D}_o \subseteq \mathcal{C}_o$ et $\mathcal{D}_m \subseteq \mathcal{C}_m$;
- les domaines et les codomaines d'une flèche de $\mathcal{D}$ sont les mêmes que dans $\mathcal{C}$: pour tout $f \in \mathcal{D}_m$, $dom_{\mathcal{D}}(f) = dom_{\mathcal{C}}(f)$ et $codom_{\mathcal{D}}(f) = codom_{\mathcal{C}}(f)$;
- pour tout objet $A \in \mathcal{D}_o$, $\mathrm{Id}_A \in \mathcal{D}_m$ où Id_A est l'identité de A dans $\mathcal{C}$;
- pour tout $f : A \to B, g : B \to C \in \mathcal{D}_m$, la composition (dans $\mathcal{C}$) $g \circ f$ appartient à $\mathcal{D}_m$ et est la composition dans $\mathcal{D}$.

Lorsqu'une sous-catégorie préserve toutes les flèches de la catégorie dont elle est issue, on dit qu'elle est pleine. Cette notion est à rapprocher de celle de sous-graphe.

DÉFINITION A.4 — *Sous-catégorie pleine.*

Soient $\mathcal{C}$ et $\mathcal{D}$ deux catégories telles que $\mathcal{D}$ est une sous-catégorie de $\mathcal{C}$. $\mathcal{D}$ est une sous-catégorie pleine de $\mathcal{C}$ si les flèches entre deux objets A et B de $\mathcal{D}$ sont exactement les flèches entre A et B : $\forall A, B \in \mathcal{D}_o, Hom_{\mathcal{D}}(A, B) = Hom_{\mathcal{C}}(A, B)$.

À partir d'une catégorie $\mathcal{C}$, il est possible de construire une catégorie notée $\mathcal{C}^{op}$ en inversant toutes les flèches. Cette catégorie est appelée catégorie duale de $\mathcal{C}$.

DÉFINITION A.5 — *Catégorie duale.*

Soit $\mathcal{C}$ une catégorie. La catégorie duale de $\mathcal{C}$ est la catégorie $\mathcal{C}^{op}$ définie par :
- $\mathcal{C}^{op}_o = \mathcal{C}_o$ et $\mathcal{C}^{op}_m = \mathcal{C}_m$;
- pour tout $f : A \to B \in \mathcal{C}_m$, $f : B \to A \in \mathcal{C}^{op}_m$;
- pour tout $h = g \circ f \in \mathcal{C}_m$, $h = f \circ g \in \mathcal{C}^{op}_m$.

3 Propriétés des morphismes

DÉFINITION A.6 — *Morphisme inverse.*

Soit $\mathcal{C}$ une catégorie. Soient $f : A \to B$ et $g : B \to A$ deux morphismes de $\mathcal{C}$ tels que $f \circ g = \mathrm{Id}_B$ et $g \circ f = \mathrm{Id}_A$. On dit alors que g est un *inverse* de f et, bien sûr, que f est un inverse de g.

DÉFINITION A.7 — *Isomorphisme.*

Soit $\mathcal{C}$ une catégorie, et soient A et B deux objets de $\mathcal{C}$. un morphisme $f : A \to B$ est un *isomorphisme* s'il admet un inverse. Dans ce cas, A et B sont dits *isomorphes*.

Une fonction $f : A \to B$ dans *Set* est injective si pour tout $x, y \in A$, si $x \neq y$ alors $f(x) \neq f(y)$. Un monomorphisme est un type particulier de morphisme dans une catégorie qui généralise la notion de fonction injective. En particulier, unmonomorphisme dans la catégorie des ensembles est exactement une fonction injective. Si f est une flèche d'une catégorie quelconque, la définition est la même, à un changement près, la notion d'élément n'ayant plus de sens dans ce contexte.

DÉFINITION A.8 — *Monomorphisme.*

Soit $\mathcal{C}$ une catégorie. Un morphisme $f : A \to B$ de $\mathcal{C}$ est un monomorphisme si pour tout objet Z dans $\mathcal{C}$ et pour tous morphismes $g, h : Z \to A$ dans $\mathcal{C}$, si $g \neq h$, alors $f \circ g \neq f \circ h$.

Un épimorphisme dans une catégorie est un monomorphisme dans la catégorie duale. On a donc la définition suivante :

DÉFINITION A.9 — *Épimorphisme.*

Soit $\mathcal{C}$ une catégorie. Un morphisme $f : A \to B$ de $\mathcal{C}$ est un épimorphisme si pour tout objet Z dans $\mathcal{C}$ et pour tous morphismes $g, h : B \to Z$ dans $\mathcal{C}$, si $g \circ f \neq h \circ f$, alors $g = h$.

Les épimorphismes dans la catégorie des ensembles sont les fonctions surjectives. On s'attend alors à ce qu'un isomorphisme soit à la fois un monomorphisme et un épimorphisme. Cependant, ce n'est pas directement vrai avec les définitions précédentes, une condition supplémentaire est nécessaire.

Un morphisme $f : A \to B$ dans une catégorie est un isomorphisme s'il a un inverse $g : B \to A$ qui satisfait les deux équations $f \circ g = \mathrm{Id}_B$ et $g \circ f = \mathrm{Id}_A$. Lorsque seule la première équation est satisfaite, f est dit être un inverse à gauche de g et réciproquement, g est dit être un inverse à droite de f.

DÉFINITION A.10 — *Morphisme scindant.*

Soit $\mathcal{C}$ une catégorie. Soient $f : A \to B$ et $g : B \to A$ deux morphismes de $\mathcal{C}$ tels que g soit un inverse à droite de f. Alors f est appelé *épimorphisme scindant* et g est appelé *monomorphisme scindant.*

On a alors la proposition suivante :

PROPOSITION A.1. *Soit $\mathcal{C}$ une catégorie. Un morphisme $f : A \to B$ dans $\mathcal{C}$ est un isomorphisme si et seulement si il est à la fois un épimorphisme scindant et un monomorphisme, ou bien, il est à la fois un épimorphisme et un monomorphisme scindant.*

4 Théorie des ensembles et catégories

4.1 Produit cartésien

Soient X et Y deux ensembles. Le *produit cartésien* de X et Y est défini par :

$$X \times Y = \{(x, y) \mid x \in X \text{ et } y \in Y\}$$

On a les fonctions de projection suivantes :

$$\begin{array}{rcl} \pi : X \times Y & \to & X \\ (x,y) & \mapsto & x \end{array} \qquad \begin{array}{rcl} \pi' : X \times Y & \to & Y \\ (x,y) & \mapsto & y \end{array}$$

Soient Z un ensemble et $f : Z \to X$ et $g : Z \to Y$ deux fonctions. Il existe une unique fonction « paire » $\langle f, g\rangle : Z \to X \times Y$ telle que $\pi \circ \langle f, g\rangle = f$ et $\pi' \circ \langle f, g\rangle = g$, c'est-à-dire $\langle f, g\rangle(z) = (f(z), g(z))$ pour tout $z \in Z$. On remarque que $\langle \pi, \pi'\rangle = id : X \times Y \to X \times Y$ et que $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle : W \to X \times Y$ où $h : W \to Z$.

L'opération produit cartésien ne s'applique pas qu'aux ensembles mais aussi aux fonctions. Soient deux fonctions $f : X \to X'$ et $g : Y \to Y'$. Le produit cartésien de f et g est défini par[1] :

$$\begin{array}{rcl} f \times g : X \times Y & \to & X' \times Y' \\ (x,y) & \mapsto & (f(x), g(y)) \end{array} \tag{A.1}$$

On remarque que $f \times g$ peut être décrit en termes de projections et de fonction paire comme suit : $f \times g = \langle f \circ \pi, g \circ \pi'\rangle$. On peut vérifier que :

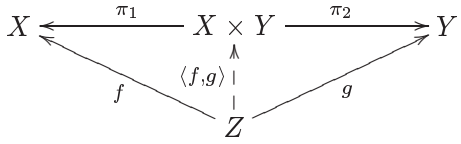
$$id_X \times id_Y = id_{X \times Y} \text{ et } (f \circ h) \times (g \circ k) = (f \times g) \circ (h \times k)$$

Le produit cartésien $\times$ est donc *fonctoriel* : il ne s'applique pas que sur les ensembles mais aussi sur les fonctions, et il préserve l'identité et la composition. L'opération produit $\times$ est un foncteur de $\mathcal{S}et \times \mathcal{S}et \to \mathcal{S}et$ (où $\mathcal{S}et \times \mathcal{S}et$ désigne la catégorie produit).

Le produit cartésien d'ensembles est en fait une instance de la notion générale de produit dans une catégorie.

DÉFINITION A.11 — *Produit.*

Soit $\mathcal{C}$ une catégorie. Le produit de deux objets $X, Y \in \mathcal{C}$ est un nouvel objet $X \times Y \in \mathcal{C}$ avec deux morphismes de projection $\pi_1 : X \times Y \to X$ et $\pi_2 : X \times Y \to Y$ qui sont universels : pour toute paire d'applications $f : Z \to X$ et $g : Z \to Y$ dans $\mathcal{C}$, il existe un unique morphisme $\langle f, g\rangle : Z \to X \times Y$ dans $\mathcal{C}$ tel que le diagramme suivant commute.



Le produit de deux objets peut ne pas exister dans une catégorie, mais s'il existe, il est unique à isomorphisme près, c'est-à-dire que s'il existe un autre objet $X \otimes Y$ avec des projections $X \xleftarrow{p_1} X \otimes Y \xrightarrow{p_2} Y$ vérifiant la propriété d'universalité, alors il existe un unique isomorphisme $X \times Y \xrightarrow{\cong} X \otimes Y$ qui commute avec les projections.

On a défini le produit de deux objet X et Y. On note X^n le produit d'un objets X avec lui-même n fois. On peut généraliser la notion de produit à une collection d'objets $(X_i)_{i \in I}$ indexés par un ensemble I quelconque, on le note $\prod_{i \in I} X_i$. Le produit X^n est alors le produit $\prod_{1 \leq i \leq n} X_i$ de n copies de X.

On s'intéresse en particulier au *produit vide*, indexé par l'ensemble vide $\emptyset$: l'objet final.

[1] Par abus de notation, on utilise le symbole $\times$ pour désigner l'opération sur les ensembles ainsi que sur les fonctions.

DÉFINITION A.12 — *Objet terminal.*

Un objet *final* dans une catégorie $\mathcal{C}$ est un objet, usuellement noté $1 \in \mathcal{C}$, tel que pour tout objet $X \in \mathcal{C}$, il existe un unique morphisme $!_X : X \to 1 \in \mathcal{C}$.

Quand une catégorie possède le produit binaire $\times$ et un objet final 1, on dit que la catégorie possède les produits finis : pour toute liste d'objets $X_1, \ldots, X_n$, on peut faire le produit $X_1 \times \ldots \times X_n$.

Une catégorie peut ne pas avoir d'objet final, mais la catégorie $\mathcal{S}et$ des ensembles en a un. N'importe quel ensemble à un élément est un objet final de $\mathcal{S}et$.

4.2 Somme

Soient X et Y deux ensembles. La *somme* (ou *coproduit*, ou encore *union disjointe*) de X et Y est définie par :

$$X + Y = \{\langle 0, x\rangle \mid x \in X\} \cup \{\langle 1, y\rangle \mid y \in Y\}$$

On a les coprojections suivantes :

$$\begin{array}{rcl@{\qquad}rcl} \kappa : X & \to & X + Y & \kappa' : Y & \to & X + Y \\ x & \mapsto & \langle 0, x\rangle & y & \mapsto & \langle 1, y\rangle \end{array}$$

On a également une opération de « cotupling » : si $f : X \to Z$ et $g : Y \to Z$ sont deux fonctions, il existe une unique fonction $[f, g] : X + Y \to Z$ telle que $[f, g] \circ \kappa = f$ et $[f, g] \circ \kappa' = g$, c'est-à-dire $[f, g](u) = f(u)$ si $u = \langle 0, x\rangle$ et $[f, g](u) = g(u)$ si $u = \langle 1, y\rangle$. On remarque que $[\kappa, \kappa'] = id$ et que $h \circ [f, g] = [h \circ f, h \circ g]$.

On étend la définition de la somme aux fonctions de la manière suivante. Si $f : X \to X'$ et $g : Y \to Y'$ sont deux fonctions, on définit la somme de f et g par[2] :

$$(f + g)(u) = \begin{cases} \langle 0, f(x)\rangle & \text{si } u = \langle 0, x\rangle \\ \langle 1, g(y)\rangle & \text{si } u = \langle 1, y\rangle \end{cases} \tag{A.2}$$

On peut aussi définir $f + g$ en termes de coprojections et cotupling : $f + g = [\kappa \circ f, \kappa' \circ g]$. On peut vérifier que :

$$id_X + id_Y = id_{X+Y} \text{ et } (f \circ h) + (g \circ k) = (f + g) \circ (h + k)$$

La somme a donc également la propriété de *fonctorialité*. L'opération $+$ est donc un foncteur de $\mathcal{S}et \times \mathcal{S}et \to \mathcal{S}et$, tout comme le produit.

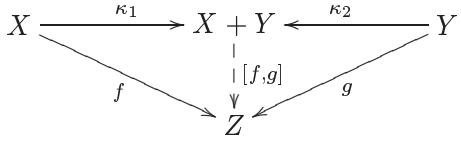
La somme sur les ensembles est une instance de la notion générale de somme dans une catégorie.

DÉFINITION A.13 — *Somme.*

La somme de deux objets $X, Y \in \mathcal{C}$ est un nouvel objet $X + Y \in \mathcal{C}$ avec deux morphismes de coprojections $\kappa_1 : X \to X + Y$ et $\kappa_2 : Y \to X + Y$ qui sont universels : pour toute paire d'applications

[2] Par un abus de notation similaire au précédent, on note $+$ la somme d'ensembles et de fonctions.

$f : X \to Z$ et $g : Y \to Z$, il existe un unique morphisme $[f, g] : X + Y \to Z$ tel que le diagramme suivant commute.



Il y a également une notion de somme vide.

DÉFINITION A.14 — *Objet initial.*

Un objet *initial* 0 dans une catégorie $\mathcal{C}$ est un objet tel que pour tout objet $X \in \mathcal{C}$, il existe un unique morphisme $!_X : 0 \to X \in \mathcal{C}$.

Comme pour les produits, on dit qu'une catégorie possède les sommes finis si elle possède les sommes $+$ et un objet initial. Dans ce cas, on peut faire les sommes $X_1 + \ldots + X_n$ de toute liste finie d'objets X_i.

Dans $\mathcal{S}et$, l'objet initial est l'ensemble vide $\emptyset$.

4.3 Espace de fonctions

Étant donnés deux ensembles X et Y, on considère l'ensemble

$$Y^X = \{f \mid f \text{ est une fonction totale } X \to Y\}$$

Cet ensemble est appelé espace de fonctions ou bien exposant de X et Y. Comme le produit et le somme, il est muni de certaines opérations de base. Il y a une fonction d'évaluation $ev : Y^X \times X \to Y$ qui à une paire (f, x) associe le résultat $f(x)$ de f appliquée à x. On a également une fonction d'abstraction : pour toute fonction $f : Z \times X \to Y$, il y a une fonction $\Lambda(f) : Z \to Y^X$ qui à tout $z \in Z$ associe la fonction $x \mapsto f(z, x)$ qui elle-même associe à tout $x \in X$ $f(z, x) \in Y$.

On va montrer que l'opération exposant est un foncteur de $\mathcal{S}et^{\mathrm{op}} \times \mathcal{S}et \to \mathcal{S}et$.

Pour deux fonctions $k : X \to U \in \mathcal{S}et^{\mathrm{op}}$ et $h : Y \to V \in \mathcal{S}et$, on veut définir une fonction $h^k : Y^X \to V^U$. Comme $k : X \to U$ est un morphisme de $\mathcal{S}et^{\mathrm{op}}$, cela signifie qu'en réalité, c'est la fonction $k : U \to X$. On peut alors définir h^k sur une fonction $f \in Y^X$ comme

$$h^k(f) = h \circ f \circ k \tag{A.3}$$

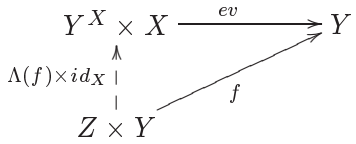
qui est bien une fonction de V^U. On doit maintenant montrer que l'identité et la composition sont conservées. On a $(id_Y^{id_X})(f) = id_Y \circ f \circ id_X = f$. Pour la préservation de la composition, on doit se souvenir que la composition dans une catégorie « duale » est inversée.

$$\begin{aligned}(h_2^{k_2} \circ h_1^{k_1})(f) &= h_2^{k_2}(h_1 \circ f \circ k_1) \\ &= h_2 \circ h_1 \circ f \circ k_1 \circ h_1 \\ &= ((h_2 \circ h_1)^{(k_2 \circ k_1)})(f)\end{aligned}$$

On donne maintenant la définition catégorielle de l'exposant.

DÉFINITION A.15 — *Espace de fonctions.*

Soit $\mathcal{C}$ une catégorie possédant le produit $\times$. L'exposant de deux objets $X, Y \in \mathcal{C}$ est un nouvel objet $Y^X \in \mathcal{C}$ avec un morphisme d'évaluation $ev : Y^X \times X \to Y$ tel que pour toute application $f : Z \times X \to Y$ dans $\mathcal{C}$, il existe un unique morphisme d'abstraction $\Lambda(f) : Z \to Y^X$ tel que le diagramme suivant commute.



4.4 Ensemble des parties

Étant donné un ensemble X, on définit par

$$\mathcal{P}(X) = \{U \mid U \subseteq X\}$$

l'ensemble des parties de X. Cette opération est aussi fonctorielle. Pour une fonction $f : X \to Y$, la fonction $\mathcal{P}(f) : \mathcal{P}(X) \to \mathcal{P}(Y)$ est donnée par ce qu'on appelle l'*image directe* : pour tout $U \subseteq X$,

$$\begin{array}{rcl} \mathcal{P}(f)(U) & = & \{f(x) \mid x \in U\} \\ & = & \{y \in Y \mid \exists x \in X, f(x) = y \wedge x \in U\} \end{array} \tag{A.4}$$

On note l'ensemble des parties *finies* de X par $\mathcal{P}_f(X) = \{U \mid U \subseteq X \wedge U \text{ fini}\}$.

4.5 Monoïde

Étant donné un ensemble X, on définit par

$$X^* = \{x_1 \dots x_n \mid \forall 1 \leq i \leq n, x_i \subseteq X\}$$

l'ensemble des mots sur X. Cette opération est aussi fonctorielle. Pour une fonction $f : X \to Y$, la fonction f^* est définie par

$$\begin{array}{rcl} f^* : X^* & \to & Y^* \\ x_1 \dots x_n & \mapsto & f(x_1) \dots f(x_n) \end{array} \tag{A.5}$$

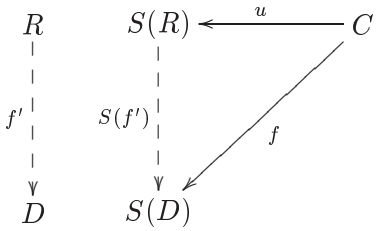
5 Limites et colimites

Les notions de produit, somme, objet initial et terminal qui viennent d'être présentées sont en fait des cas particuliers des notions de limites et colimites.

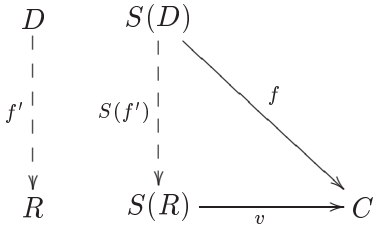
DÉFINITION A.16 — *Flèche universelle.*

Si $S : \mathcal{D} \to \mathcal{C}$ est un foncteur et $C \in |\mathcal{C}|$ est un objet de $\mathcal{C}$, une *flèche universelle de C vers S* est la donnée d'une paire (R, u), où $R \in |\mathcal{D}|$ est un objet de $\mathcal{D}$ et $u \in \mathcal{C}(C, S(R))$ est une flèche de $\mathcal{C}$, telle que pour toute autre paire (D, f) où $D \in |\mathcal{D}|$ et $f \in \mathcal{C}(C, S(D)))$, il existe une unique flèche $f' \in \mathcal{D}(R, D)$ de $\mathcal{D}$ telle que $S(f') \circ u = f$.

La définition précédente signifie en fait que chaque flèche f vers S se factorise de façon unique au travers de la flèche universelle u comme indiqué sur le diagramme ci-dessous.



On a défini la notion de flèche universelle d'un objet C vers un foncteur $S : \mathcal{D} \to \mathcal{C}$. Le concept dual est également utile. Une flèche universelle de S vers C est une paire (R, v) où R est un objet $R \in \mathcal{D}$ et v est une flèche $v : S(R) \to C$ de codomaine C telle que pour toute autre paire (D, f), où $D \in \mathcal{D}$ et $f : S(D) \to C$, il existe une unique flèche $f' : D \to R$ telle que $f = v \circ S(f')$, comme dans le diagramme commutatif suivant :



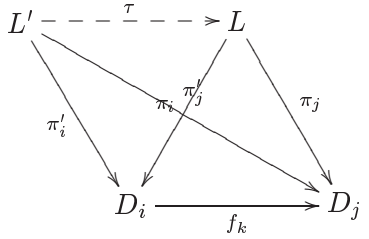
Les projections $\pi_1 : A \times B \to A$ et $\pi_2 : A \times B \to B$ d'un produit dans $\mathcal{C}$ sont des exemples de flèches universelles. En effet, étant donnés un objet $C \in \mathcal{C}$ et deux flèches $f : C \to A$ et $g : C \to B$, il existe un unique $h : C \to A \times B$ tel que $\pi_1 \circ h = f$ et $\pi_2 \circ h = g$. $\langle \pi_1, \pi_2 \rangle$ est donc une « paire universelle ». Pour en faire une flèche universelle, on introduit le foncteur diagonal $\Delta : \mathcal{C} \to \mathcal{C} \times \mathcal{C}$ défini par $\Delta(C) = \langle C, C \rangle$. Alors la paire f, g précédente devient une flèche $\langle f, g \rangle : \Delta(C) \to \langle A, B \rangle$ dans $\mathcal{C} \times \mathcal{C}$, et $\langle \pi_1, \pi_2 \rangle$ est une flèche universelle de Δ vers l'objet $\langle A, B \rangle$.

DÉFINITION A.17 — *Cône, limite.*

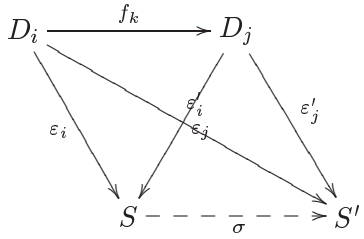
Soit un diagramme $\mathcal{D}$ dont les objets sont une famille $(D_i)_{i \in I}$ et les flèches une famille $(f_k)_{k \in K}$. Un *cône* sur $\mathcal{D}$ est un objet L muni de morphismes $\pi_i : L \to D_i$ pour chaque $i \in I$, tel que pour chaque flèche $f_k : D_i \to D_j$, $f_k \circ \pi_i = \pi_j$.

Un cône $(L, (\pi_i)_{i \in I})$ est appelée une *limite faible* de $\mathcal{D}$ si pour tout autre cône $(L', (\pi'_i)_{i \in I})$ sur $\mathcal{D}$ il existe un morphisme $\tau : L' \to L$ tel que $\pi'_i = \pi_i \circ \tau$ pour tout $i \in I$.τ est appelé *morphisme médiateur*.

$L = (L, (\pi_i)_{i \in I})$ est appelé la *limite* de $\mathcal{D}$ si L est une limite faible et si le morphisme médiateur $\tau : L' \to L$ est toujours unique. Les morphismes π_i sont appelés *morphismes canoniques*.



Les *colimites* sont définies comme étant les duales des limites, c'est-à-dire que le concept est le même avec toutes les flèches inversées. Plus précisément, un *cocône* $S = (S, (\varepsilon_i)_{i \in I})$ a des flèches $\varepsilon_i : D_i \to S$. S est une *colimite faible* si pour tout autre cocône S' il existe un morphisme médiateur $\sigma : S \to S'$ tel que $\sigma \circ \varepsilon_i = \varepsilon'_i$. S est la *colimite* de $\mathcal{D}$ si pour tout autre cocône S' le morphisme médiateur σ est unique.

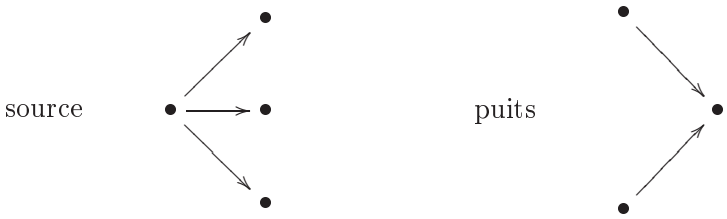


Les limites et colimites, si elles existent, sont uniques à isomorphisme près. Dans la catégorie *Set*, toutes les limites et colimites possibles existent. *Set* est dite *complète* et *cocomplète* pour cette raison.

Les limites sur un type particulier de diagramme jouent souvent un rôle important, on leur donne alors des noms spécifiques, indiqués dans la tableau suivant.

type du diagramme	limite	colimite
vide	objet terminal	objet initial
sans flèche	produit	somme
source		pushout
puits	pullback	
flèches parallèles	égaliseur	coégaliseur

On considère des types de diagrammes particuliers. Un diagramme consistant en une collection de flèches, toutes de même domaine, est appelé une *source*. Son dual, un *puits*, est une collection de flèches de codomaine commun. Une source peut aussi être considérée comme un cône sur un diagramme sans flèche, et un puits comme un cocône.



Annexe B

Normalisation d'arbres de preuve

Nous présentons ici brièvement des résultats généraux que nous avons établis sur la normalisation d'arbres de preuve. Cette présentation est un résumé d'un rapport de recherche plus complet [ABL07], dans lequel une série d'exemples bien connus sont donnés. Nous ne donnerons pas ici d'exemples, la thèse en comportant déjà quelques uns [1].

Nous présentons ici un cadre général dans lequel des résultats de normalisation d'arbres de preuve, comme le résultat de logicalité en raisonnement équationnel ou l'élimination des coupures en calcul des séquents ou en déduction naturelle par exemple, peuvent être unifiés et généralisés. Ce cadre permet de donner des conditions simples à vérifier et suffisantes pour assurer ces résultats de normalisation. De plus, ces conditions étant purement syntaxiques, elles peuvent être implantées. Elles correspondent aux propriétés que doivent vérifier certaines combinaisons élémentaires de règles d'inférence pour assurer que le processus de transformation d'arbres de preuve global termine.

Dans de nombreuses situations en logique, pour faciliter l'usage d'un système logique, pour obtenir des résultats de consistance des systèmes de preuve, ou encore pour prouver automatiquement des théorèmes, des stratégies de recherche de preuves sont utilisées. Ces stratégies permettent de limiter l'espace de recherche des preuves à une certaine classe d'arbres ayant une structure spécifique, dans le but de réduire la complexité de la recherche ou même simplement de la rendre réalisable. Différentes stratégies ont été définies dans des contextes variés. Par exemple,

- le résultat de logicalité, qui établit la correspondance entre la dérivabilité et la convertibilité en réécriture pour de nombreuses logiques équationnelles (sous-équationnelle [vO04], sans sortes [Bir35], multi-sorte, conditionnelle [Kap84], partielle [ABD02], *etc.*) ;
- le résultat d'élimination des coupures qui montre que la règle de coupure est redondante pour le calcul des séquents et le calcul de la déduction naturelle de plusieurs logiques (logique du premier ordre classique [Gen35], logique du premier ordre intuitionniste [Kle52], quelques logiques modales [Wal90], logique linéaire [Gir87], déduction modulo [DW99], *etc.*) ;

[1] Ces résultats sont en effet utilisés dans les preuves de complétude de la méthode de sélection par dépliage des axiomes que nous présentons dans différents formalismes au cours de cette thèse.

- la propriété de confluence des systèmes de réécriture qui établit que la dérivabilité peut être prouvée par des « vallées » pour plusieurs logiques dans lesquelles les relations sont transitives (logique équationnelle [DJ90, BN98], logique des préordres [LA96, Str96], logiques des relations spéciales [Sch98]) [2] ;
- en utilisant l'isomorphisme de Curry-Howard, les résultats de normalisation en λ-calcul typé débutés par D. Prawitz [Pra65].

Dans tous ces cas, la principale difficulté est de montrer que la dérivabilité pleine (c'est-à-dire sans stratégie de preuve particulière) coïncide avec la dérivabilité restreinte à une classe donnée de preuves (c'est-à-dire munie d'une stratégie spécifique). La correction d'une stratégie particulière de recherche de preuves, c'est-à-dire le fait que la dérivabilité restreinte est incluse dans la dérivabilité pleine, est évidente, car les preuves résultant de l'application de cette stratégie sont des instances particulières des preuves menées sans stratégie. La complétude, qui exprime l'inclusion inverse, est bien plus difficile à assurer. En effet, il faut montrer que pour tout théorème dont l'arbre de preuve appartient à la classe générale, il existe un arbre de preuve de ce théorème dans la classe restreinte, c'est-à-dire construit selon la stratégie considérée. Dans la plupart des systèmes logiques, la complétude est une conséquence d'un résultat plus fort qui consiste à définir des transformations simples pour réécrire des combinaisons élémentaires de règles d'inférence et à montrer que la transformation d'arbres de preuve globale induite par ces transformations est normalisante. Par exemple, en ce qui concerne le résultat de logicalité en logique équationnelle classique, les transformations consistent à distribuer la règle de substitution et de contexte sur la règle de transitivité, et à éliminer toutes les règles sous la réflexivité. Pour l'élimination des coupures en calcul des séquents, les transformations consistent à éliminer la règle de coupure sous les axiomes et à la faire remonter sur les autres règles, pour pouvoir l'éliminer une fois aux feuilles.

La définition de ces transformations élémentaires est généralement fastidieuse mais facile. La difficulté est de montrer que la transformation d'arbres de preuve globale induite est normalisante. Par exemple, prouver le résultat d'élimination des coupures demande d'utiliser des inductions emboîtées, car il n'est pas évident que le processus qui consiste à remonter la règle de coupure sur les autres règles termine [Gal86]. À chaque fois, ces résultats de normalisation d'arbres de preuve sont établis pour chaque système logique sous-jacent de façon *ad-hoc*.

Les résultats que nous présentons ici permettent d'unifier et de généraliser la méthode de normalisation d'arbres de preuve utilisée en calcul des séquents, en déduction naturelle, dans le raisonnement équationnel ou en réécriture de termes, dans un cadre abstrait de systèmes logiques quelconques. L'abstraction est obtenue en étudiant la normalisation d'arbres de preuve dans le cadre abstrait des systèmes formels, ce qui permet d'établir des résultats indépendamment du système logique.

1 Préliminaires

On considère ici qu'un système formel est réduit à un couple (F, R), l'alphabet A de la définition I.18 du chapitre I n'étant d'aucune utilité.

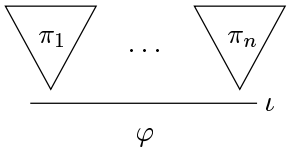
Notations

[2] D'autre part, dans le cadre équationnel, G. Dowek a montré que la confluence d'un système de réécriture peut être définie comme la propriété d'élimination des coupures d'un système de preuve associé à ce système de réécriture : la déduction modulo asymétrique [Dow03].

1. Étant donné un arbre de preuve π, on note $\mathcal{LM}(\pi)$ (respectivement $\mathcal{LS}(\pi)$) le multi-ensemble[3] (respectivement l'ensemble[4]) des feuilles de π. Nous utiliserons la notation $\{\!\{ a_1, a_2, \ldots, a_n \}\!\}$ pour désigner un multi-ensemble.

2. Un arbre de preuve π de conclusion φ est noté $\pi : \varphi$.

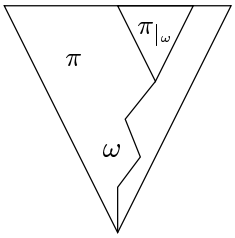
3. On écrit $\pi = (\pi_1, \ldots, \pi_n, \varphi)_\iota$, avec $n \in \mathbb{N}$, l'arbre de preuve dont la dernière instance de règle est $\dfrac{\varphi_1, \ldots, \varphi_n}{\varphi}\iota$ et tel que pour tout i, $1 \leq i \leq n$, π_i est le sous-arbre de π de conclusion φ_i.



En utilisant la numérotation standard des nœuds d'un arbre par des mots sur l'ensemble des entiers naturels, on peut faire référence à la position d'une formule dans un arbre.

DÉFINITION B.1 — *Position dans un arbre.*

Soit π un arbre de preuve. Une *position* dans π est un mot ω sur $\mathbb{N}$ qui représente le chemin de la racine de π au sous-arbre dont la conclusion apparaît à cette position. Ce sous-arbre est noté $\pi_{|\omega}$.



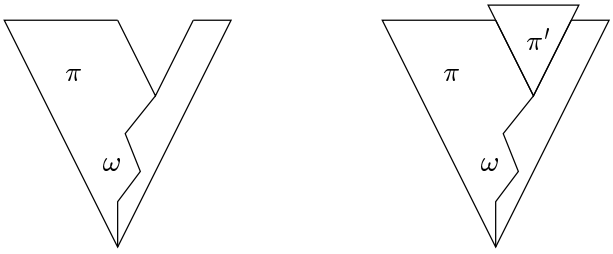
DÉFINITION B.2 — *Contexte.*

Soit π un arbre de preuve. Soit $\omega \in \mathbb{N}^*$ une position dans π. L'arbre π dans lequel la position ω est laissée vacante, noté $\pi[.]_\omega$, est appelée un *contexte*.

Si π' est un arbre de preuve de même conclusion que $\pi_{|\omega}$, $\pi[\pi']_\omega$ est l'arbre de preuve obtenu à partir de π en remplaçant le sous-arbre $\pi_{|\omega}$ par π'.

[3] Dans $\mathcal{LM}(\pi)$, toutes les feuilles de π sont considérées. Nous avons alors un multi-ensemble car plusieurs occurrences de la même formule peuvent apparaître dans π.

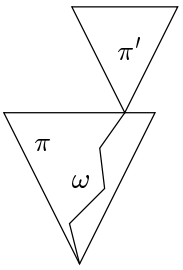
[4] Ici, lorsqu'une formule φ de π possède plusieurs occurrences, une seulement est considérée dans $\mathcal{LS}(\pi)$.



$\pi[.]_\omega$ $\pi[\pi']_\omega$

DÉFINITION B.3 — *Composition.*

Soient π et $\pi' : \varphi$ deux arbres de preuve et ω la position d'une feuille dans π telle que $\pi_{|\omega} = \varphi$. On note alors $\pi \cdot_\omega \pi'$, au lieu de $\pi[\pi']_\omega$, la *composition* de π et π' à la position ω.



2 Procédure de transformation d'arbres de preuve

Dans tous les calculs logiques pour lesquels des stratégies de recherche de preuves ont été appliquées, la complétude de la dérivabilité restreinte par rapport à la dérivabilité pleine est obtenue en définissant des procédures de transformation d'arbres de preuve normalisantes fondées sur des transformations d'arbres de preuve élémentaires. Lorsqu'on étudie la plupart de ces procédures, on peut observer qu'elles consistent à remplacer, dans les preuves, des schémas élémentaires, que nous appellerons arbres de preuve élémentaires, de la forme $(\iota_1, \ldots, \iota_n, \varphi)_\iota$ où chaque ι_i $(1 \leq i \leq n)$ est soit une instance de règle dans R, soit une formule de F, par des arbres de preuve en forme normale (c'est-à-dire des arbres auquel on ne peut plus appliquer de transformation). Ces schémas élémentaires décrivent des situations critiques qui ne respectent pas la stratégie car certaines des instances de règle ι_i $(1 \leq i \leq n)$ ne devraient pas se trouver au-dessus de ι. Par exemple, parmi les transformations sur lesquelles repose le résultat de logicalité en logique équationnelle[5], on trouve la règle de transformation suivante :

$$\dfrac{\dfrac{t = t'' \qquad t'' = t'}{t = t'}\,\mathit{Trans.}}{\sigma(t) = \sigma(t')}\,\mathit{Sub.} \quad \rightsquigarrow \quad \dfrac{\dfrac{t = t''}{\sigma(t) = \sigma(t'')}\,\mathit{Sub.} \qquad \dfrac{t'' = t'}{\sigma(t'') = \sigma(t')}\,\mathit{Sub.}}{\sigma(t) = \sigma(t')}\,\mathit{Trans.}$$

[5]Une présentation complète de ce résultat se trouve dans le rapport de recherche [ABL07].

Un autre exemple, tiré du résultat d'élimination des coupures en calcul des séquents, est le suivante [6] :

$$\dfrac{\dfrac{\Gamma,\varphi \vdash \Delta \qquad \Gamma,\varphi' \vdash \Delta}{\Gamma,\varphi\vee\varphi' \vdash \Delta}\vee\text{-gauche} \qquad \dfrac{\Gamma' \vdash \Delta',\varphi,\varphi'}{\Gamma' \vdash \Delta',\varphi\vee\varphi'}\vee\text{-droit}}{\Gamma,\Gamma' \vdash \Delta,\Delta'}\mathsf{Cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma,\varphi \vdash \Delta \qquad \Gamma' \vdash \Delta',\varphi,\varphi'}{\Gamma,\Gamma' \vdash \Delta,\Delta',\varphi'}\mathsf{Cut} \qquad \Gamma,\varphi' \vdash \Delta}{\Gamma,\Gamma' \vdash \Delta,\Delta'}\mathsf{Cut}$$

Dans le résultat d'élimination des coupures pour la déduction naturelle, se trouve la règle de transformation suivante :

$$\dfrac{\dfrac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\wedge\textbf{-intro}}{\Gamma \vdash \varphi}\wedge\textbf{-elim} \qquad \rightsquigarrow \qquad \Gamma \vdash \varphi$$

DÉFINITION B.4 — *Arbre de preuve élémentaire.*

Soit $\mathcal{S} = (F, R)$ un système formel. Un *arbre de preuve élémentaire* est un arbre de preuve de la forme $(\iota_1, \ldots \iota_n, \varphi)_\iota$ tel que pour tout i, $1 \leq i \leq n$, soit $\iota_i \in F$ soit il existe $r \in R$ avec $\iota_i \in r$.

DÉFINITION B.5 — *Procédure de transformation.*

Soit $\mathcal{S} = (F, R)$ un système formel. Une *procédure de transformation (d'arbres de preuve) pour* $\mathcal{S}$ est une relation binaire sur les arbres de preuve $\rightsquigarrow$ telle que pour tout $\pi \rightsquigarrow \pi'$:

- π est un arbre de preuve élémentaire ;
- π' est un arbre de preuve en forme normale ;
- $\mathcal{LS}(\pi') \subseteq \mathcal{LS}(\pi)$;
- π et π' sont de même conclusion.

On note $\rightsquigarrow_{\mathcal{S}}$, la clôture de $\rightsquigarrow$ par contexte d'arbre de preuve et composition aux feuilles.

En utilisant la terminologie habituelle en réécriture, la relation $\rightsquigarrow$ est *fortement normalisante* (ou termine) si toute suite de réductions est finie, et *faiblement normalisante* si toute preuve admet une forme normale.

La normalisation de la procédure de transformation globale obtenue par l'application répétée des règles de transformation ne peut pas être assurée sans conditions supplémentaires. De plus, des exemples connus de procédures de transformation rentrent dans le cadre de la définition B.5 mais sont faiblement normalisantes, tandis que d'autres sont fortement normalisantes (voir la section suivante). De façon à prendre en compte la plus grande famille de procédures de transformation, dans la section suivante, nous étudierons les conditions à poser sur les règles de transformations élémentaires, qui soient faciles à vérifier, et cependant suffisamment puissantes pour assurer la normalisation.

3 Théorème de normalisation forte

Une idée simple mais puissante sur laquelle repose la plupart des méthodes permettant de prouver la terminaison, tel que l'ordre récursif sur les chemins (RPO), consiste à comparer les termes des règles de réécriture en comparant tout d'abord le symbole de leur racine, puis récursivement, en comparant les

[6] Voir le rapport de recherche [ABL07] et [GLT89] pour une présentation complète de ce résultat.

collections de leurs sous-termes immédiats. Par conséquent, nous commençons par supposer un ordre bien fondé sur les éléments atomiques des arbres de preuve, c'est-à-dire les instances de règles. Ainsi, étant donnée une procédure de transformation $\rightsquigarrow$, on dispose d'un ordre bien fondé $\preceq$ sur $\bigcup_{r\in R} r$. Cet ordre est souvent simple à définir pour la plupart des instances de règles. En effet, il est évident d'imposer que chaque instance ι d'une règle qui ne doit pas se trouver sous des instances ι' d'autres règles dans les arbres de preuve ait un poids plus faible pour cet ordre que ι'. Parfois, cet ordre est défini de façon plus *ad-hoc* pour des instances d'une même règle d'inférence. Par exemple, dans la preuve de terminaison du résultat d'élimination des coupures, cet ordre est défini de la façon suivante [7] :

$$\forall @ \in \{\vee, \neg, \exists\},\ \mathsf{Cut} \succ @\text{-droit}, @\text{-gauche}, \mathsf{Taut}$$

et

$$\frac{\Gamma_1 \mid\sim \Delta_1, \varphi_1 \qquad \Gamma_1', \varphi_1 \mid\sim \Delta_1'}{\Gamma_1, \Gamma_1' \mid\sim \Delta_1, \Delta_1'}\mathsf{Cut} \quad \succ \quad \frac{\Gamma_2 \mid\sim \Delta_2, \varphi_2 \qquad \Gamma_2', \varphi_2 \mid\sim \Delta_2'}{\Gamma_2, \Gamma_2' \mid\sim \Delta_2, \Delta_2'}\mathsf{Cut} \quad \Leftrightarrow \quad |\varphi_2| < |\varphi_1|$$

où $|\varphi|$ est la profondeur de la formule définie par $sup_k(1 + |\varphi_k|)$ si les φ_i sont les sous-formules directes de φ.

Dans la plupart des exemples de résultats de normalisation forte, les procédures de normalisation $\rightsquigarrow$ consistent plus ou moins à distribuer certaines règles au-dessus d'autres de façon à respecter la stratégie de recherche de preuves (voir par exemple le cas présenté plus haut de la règle *Sub.* sur la règle *Trans.*). La condition 1 généralise cette notion de distributivité.

CONDITION 1. *Pour tout* $(\iota_1, \ldots, \iota_n, \varphi)_\iota \rightsquigarrow \pi$ *et pour tout sous-arbre* $(\pi_1', \ldots, \pi_m', \varphi')_{\iota'}$ *de* π *:*

1. $\iota \succeq \iota'$ *si chaque* $\pi_i' \in F \cup \bigcup_{r\in R} r$

   $\iota \succ \iota'$ *sinon*

2. *Si* $\iota \sim \iota'$, *alors* $\{\!\{ \iota_1, \ldots, \iota_n \}\!\} \gg \{\!\{ \iota_1', \ldots, \iota_m' \}\!\}$ *où* $\gg$ *étend* $\succ$ *aux multi-ensembles sur* $F \cup \bigcup_{r\in R} r$.

On remarque que la condition 1 est un cas particulier de l'ordre récursif sur les chemins qui peut être facilement implanté et donc automatiquement vérifié, étant donné un ordre bien fondé sur les instances de règles. On a alors le résultat suivant :

THÉORÈME B.1. *Toute procédure de transformation* $\rightsquigarrow$ *qui vérifie la condition 1 est fortement normalisante.*

Preuve. En utilisant des termes de preuve pour représenter les preuves, avec un ordre récursif sur les chemins $>^{rpo}$ pour ordonner les preuves induit par l'ordre bien-fondé $\succeq$ sur les instances de règles, on montre que $\rightsquigarrow \subseteq >^{rpo}$. Soit $(\iota_1, \ldots, \iota_n, \varphi)_\iota \rightsquigarrow \pi$ une règle de transformation. Par induction sur π, montrons que $(\iota_1, \ldots, \iota_n, \varphi)_\iota >^{rpo} \pi$.

[7] Une présentation complète du calcul de Gentzen pour la logique du premier ordre classique se trouve au chapitre II.

Cas de base. π est la formule φ. D'après la définition B.5, $\mathcal{LS}(\pi) \subseteq \mathcal{LS}((\iota_1, \ldots, \iota_n, \varphi)_\iota)$. Par conséquent, comme les ordres de réduction ont la propriété de sous-terme, on conclut que $(\iota_1, \ldots, \iota_n, \varphi)_\iota >^{rpo} \pi$.

Cas général. π est de la forme $(\pi_1, \ldots, \pi_n, \varphi)_{\iota'}$. Deux cas doivent alors être considérés :

1. $\iota \succ \iota'$. Par hypothèse d'induction, pour tout i, $1 \leq i \leq n$, $(\iota_1, \ldots, \iota_n, \varphi)_\iota >^{rpo} \pi_i$, et donc d'après la définition de RPO, on a $(\iota_1, \ldots, \iota_n, \varphi)_\iota >^{rpo} \pi$.
2. $\iota \sim \iota'$. Par la condition 1.1, chaque π_i est dans $F \cup \bigcup_{r \in R} r$ $(1 \leq i \leq n)$. Par la condition 1.2, on a $\{\!\{ \iota_1, \ldots, \iota_n \}\!\} \succcurlyeq \{\!\{ \iota'_1, \ldots, \iota'_m \}\!\}$, et donc $\{\!\{ \iota_1, \ldots, \iota_n \}\!\} \gg^{rpo} \{\!\{ \iota'_1, \ldots, \iota'_m \}\!\}$. Par la définition de RPO, on conclut alors que $(\iota_1, \ldots, \iota_n, \varphi)_\iota >^{rpo} \pi$.

□

4 Théorème de normalisation faible

Cependant, des problèmes peuvent advenir avec des règles de transformation de la forme

$$(\iota_1, \ldots, \iota_n, \varphi)_\iota \rightsquigarrow \pi$$

telles qu'il existe un sous-arbre $\pi' = (\iota'_1, \ldots, \iota'_m, \varphi')_{\iota'}$ de π avec $\iota'_j \in F \cup \bigcup_{r \in R} r$ pour tout j, $1 \leq j \leq m$, satisfaisant :

- $\iota \sim \iota'$;
- $\mathcal{LM}(\iota'_j) = \mathcal{LM}((\iota_1, \ldots, \iota_n, \varphi)_\iota)$.

En effet, de telles règles de transformation peuvent empêcher d'utiliser les techniques standard comme RPO pour prouver la terminaison, donc d'obtenir un résultat de normalisation forte.

Un exemple d'une telle règle de transformation est la règle **R1** définie dans la procédure d'élimination des coupures dans le calcul des séquents **LK** qui considère la règle de coupure suivante :

$$\frac{\Gamma \Rightarrow \Delta, \varphi \qquad \Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\,\text{Cut}$$

En effet, avec une telle règle de coupure, la règle d'affaiblissement doit être exprimée de façon explicite de façon à permettre la transformation suivante :

R1

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \Rightarrow \Delta_1, \varphi}{\Gamma'_1 \Rightarrow \Delta'_1, \varphi}\,@\text{Right} \qquad \Gamma'_1, \varphi \Rightarrow \Delta'_1}{\Gamma'_1 \Rightarrow \Delta'_1}\,\text{Cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1 \Rightarrow \Delta_1, \varphi}{\Gamma_1, \Gamma'_1 \Rightarrow \Delta_1, \Delta'_1, \varphi}\,\text{Weak.} \qquad \dfrac{\Gamma'_1, \varphi \Rightarrow \Delta'_1}{\Gamma_1, \Gamma'_1, \varphi \Rightarrow \Delta_1, \Delta'_1}\,\text{Weak.}}{\Gamma_1, \Gamma'_1 \Rightarrow \Delta_1, \Delta'_1}\,\text{Cut}}{\Gamma'_1 \Rightarrow \Delta'_1}\,@\text{Right}$$

La seule mesure qui décroisse avec une telle règle est le nombre d'occurrences des règles @Left et @Right (c'est-à-dire des règles qui n'appartiennent pas à l'ensemble *Élim* de la définition B.6 ci-dessous) qui apparaissent au-dessus de la règle de coupure dans la partie droite de la règle. Par conséquent, en appliquant une stratégie qui élimine, dans les arbres de preuve, les arbres de preuve maximaux (voir la définition de maximalité ci-dessous) qui sont les plus proches des feuilles, ce type de mesure sera suffisant pour obtenir un résultat de normalisation faible.

DÉFINITION B.6 — *Ensemble* $\acute{E}lim$.

Soit $\rightsquigarrow$ une procédure de transformation. On définit l'ensemble $\acute{E}lim$ de la façon suivante :

$$\acute{E}lim = \{\iota \mid \exists (\iota_1, \ldots, \iota_n, \varphi)_\iota \rightsquigarrow \pi\}$$

L'ensemble $\acute{E}lim$ contient alors les instances de règles qu'il faut faire remonter ou éliminer pour respecter la stratégie choisie. Pour la procédure d'élimination des coupures, $\acute{E}lim$ contient alors toutes les instances des règles Cut, Weak et Sub.

Nous résumons maintenant cette discussion par la condition suivante exprimée dans notre cadre abstrait :

CONDITION 2. *Pour toute règle de transformation* $(\iota_1, \ldots, \iota_n, \varphi)_\iota \rightsquigarrow \pi$ *:*

1. *Il existe* i, $1 \leq i \leq n$, *tel que* $\iota_i \notin \acute{E}lim$.
2. $\mathcal{LM}(\pi) \subseteq \mathcal{LM}((\iota_1, \ldots, \iota_n, \varphi)_\iota)$.
3. *Pour tout instance de règle* ι' *apparaissant dans* π, $\iota \succeq \iota'$.
4. *Si* π *est de la forme* $(\pi_1, \ldots, \pi_m, \varphi)_{\iota'}$ *alors toute instance de règle différente de* ι' *appartient à* $\acute{E}lim$.

De nouveau, la condition 2 étant purement syntaxique, elle peut être automatiquement vérifiable étant donnée une procédure de transformation et un ordre bien-fondé sur les instances de règles.

Nous en arrivons maintenant au théorème principal.

THÉORÈME B.2. *Toute procédure de transformation* $\rightsquigarrow$ *qui vérifie la condition 2 est faiblement normalisante.*

Pour les besoins de la preuve, nous introduisons les quelques définitions suivantes.

DÉFINITION B.7 — *Arbre de preuve maximal.*

Soit $\rightsquigarrow$ une procédure de transformation. Un arbre de preuve $\pi = (\pi_1, \ldots, \pi_n, \varphi)_\iota$ est *maximal* pour $\rightsquigarrow$ si et seulement si pour tout i, $1 \leq i \leq n$, π_i est en forme normale, mais π ne l'est pas.

DÉFINITION B.8 — *Longueur d'une preuve.*

Soit $\pi = (\pi_1, \ldots, \pi_n, \varphi)_\iota$ une preuve. La *longueur* de π, notée $|\pi|$, est inductivement définie comme suit :

- si φ est une feuille, alors $|\varphi| = 0$;
- si les π_i pour tout i, $1 \leq i \leq n$, sont les sous-arbres directs de π, alors $\sum_i |\pi_i|$ si $\iota \in \acute{E}lim$ et $\sum_i |\pi_i| + 1$ sinon.

Ainsi, la longueur d'une preuve π est le nombre d'instances de règles apparaissant dans π qui n'appartiennent pas à $\acute{E}lim$.

DÉFINITION B.9 — *Rang d'une instance de règle dans une preuve.*

Tout ensemble bien-fondé étant isomorphe à un unique ordinal, on note $d : \left(\bigcup_{r \in R} r, \preceq\right) \to \alpha$, où α est un ordinal, cet isomorphisme. Soit π une preuve et soit $\pi' = (\pi_1, \ldots, \pi_n, \varphi)_\iota$ un sous-arbre de preuve de π avec $\iota \in \acute{E}lim$. Alors, le *rang* de ι dans π, noté $rk_\pi(\iota)$ est défini comme étant $d(\iota) + |\pi'|$.

Le rang de ι est donc le nombre d'instances de règles qui n'appartiennent pas à $\acute{E}lim$ et apparaissent au-dessus de ι dans π'.

DÉFINITION B.10 — *Mesure d'une preuve.*

Soit π une preuve. La *mesure* de π, notée $mes(\pi)$, est définie comme étant $\sum_\iota rk_\pi(\iota)$ où ι couvre toutes les instances de règles d'inférence de π qui appartiennent à $Elim$.

On peut maintenant procéder à la preuve.

Preuve. La preuve de ce théorème est une généralisation de la preuve de Tait [Tai89].

Le théorème va être prouvé en montrant que l'application de toute règle de transformation n'augmente pas la mesure des preuves. Deux étapes vont être nécessaires. Nous allons tout d'abord montrer que pour tout arbre de preuve maximal π transformé en $\overline{\pi}$ par une règle de transformation satisfait $mes(\overline{\pi}) < mes(\pi)$. Ensuite, nous allons montrer que ce résultat peut être étendu à toute preuve en suivant la stratégie qui consiste à réduire les sous-arbres maximaux.

Soit $\pi = (\pi_1, \ldots, \pi_n, \varphi)_\iota$ un arbre de preuve maximal tel que chaque π_i est de la forme $(\pi_{i_1}, \ldots, \pi_{i_{n_i}}, \varphi_i)_{\iota_i}$ $(1 \leq i \leq n)$. Soit $(\iota_1, \ldots, \iota_n, \varphi)_\iota \rightsquigarrow \pi'$ une règle de transformation. Cette règle transforme π en $\overline{\pi} = (\overline{\pi}_1, \ldots, \overline{\pi}_o, \varphi)_{\iota'}$. Comme π est un arbre de preuve maximal, pour tout sous-arbre de preuve $\pi'' = (\pi''_1, \ldots, \pi''_p, \varphi')_{\iota''}$ de $\overline{\pi}$ tel que l'instance de règle $\iota'' \in \acute{E}lim$, ι'' apparaît dans π'. Par la condition 2.4, aucune instance de règle de $\bigcup_{r \in R} r \setminus \acute{E}lim$ n'a été introduite dans π'' au-dessus de ι''. De plus, par la condition 2.1, il existe au moins une occurrence d'une instance de règle de $\bigcup_{r \in R} r \setminus \acute{E}lim$ qui n'apparaît plus dans π'' et par la condition 2.2, les autres occurrences de telles instances de règles étaient déjà dans π. Par conséquent, on peut conclure que $|\pi''| \leq |\pi| - 1$. Enfin, par la condition 2.3, on a $\iota \succeq \iota''$. On a donc $d(\iota'') + |\pi''| \leq d(\iota) + |\pi| - 1$.

Maintenant, soit π une preuve qui n'est pas en forme normale et qui est maximale. Soit $\pi' = (\pi'_1, \ldots, \pi'_n, \varphi)_{\iota'}$ un sous-arbre de preuve de π qui n'est pas maximal et tel que $\iota' \in \acute{E}lim$. Toute règle de transformation qui peut s'appliquer à π est appliquée soit à un sous-arbre de preuve maximal π'' de π mais pas à π', soit à un sous-arbre de preuve de π'. Dans le premier cas, $rk_\pi(\iota') = rk_{\overline{\pi}}(\iota')$ où $\overline{\pi}$ est la preuve obtenue à partir de π en appliquant la règle de transformation en question. Dans le second cas, par la condition 2.1 et 2.4, la règle de transformation a au moins éliminé une occurrence d'une instance de règle de $\bigcup_{r \in R} r \setminus \acute{E}lim$ et en a ajouté au plus une. On a donc que $rk_\pi(\iota') \geq rk_{\overline{\pi}}(\iota')$.

On peut alors conclure que dans les deux cas ci-dessus (le cas dans lequel π est maximal et celui dans lequel π n'est pas en forme normale et n'est pas maximal) $mes(\pi) > mes(\overline{\pi})$. □

Dans les procédures d'élimination des coupures pour le calcul des séquents dont les règles structurelles sont explicites (c'est-à-dire qu'on dispose des règles de contraction et d'affaiblissement et que les séquents sont définis comme des listes plutôt que comme des ensembles de formules), il est connu que des suites infinies de réduction sont possibles. En effet, dans un arbre de preuve, lorsque des instances de la règle de contraction se trouvent au-dessus d'une instance de la règle de coupure, on a la transformation suivante :

R2
$$\dfrac{\dfrac{\dfrac{\Gamma, \varphi, \varphi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\text{Contr.} \qquad \Gamma \vdash \varphi, \Delta}{\Gamma \vdash \Delta}\text{Cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma, \varphi, \varphi \vdash \Delta \qquad \dfrac{\Gamma \vdash \varphi, \Delta}{\Gamma, \varphi \vdash \Delta, \varphi}\text{Weak.}}{\Gamma, \varphi \vdash \Delta}\text{Cut} \qquad \Gamma \vdash \Delta, \varphi}{\Gamma \vdash \Delta}\text{Cut}$$

On remarque qu'aucune des conditions 1 et 2 n'est vérifiée par **R2**. Ceci est dû au fait que le séquent $\Gamma \vdash \varphi, \Delta$ et la règle Cut sont dupliqués, et au fait que Contr. $\prec$ Cut.

Pour ce calcul des séquents, des procédures d'élimination des coupures normalisantes ont été développées. La première preuve de normalisation forte a été donnée par Dragalin en 1979 [Dra88]. Plus récemment, d'autres preuves de normalisation forte de procédures d'élimination des coupures ont été données dans [TB99, Her95, UB01]. Tous ces résultats forts sont obtenus soit en cachant les contractions dans les règles d'introduction ou dans la règle de coupure, soit en transformant la règle de coupure de façon à empêcher que des instances de la règle de contraction puissent apparaître au-dessus d'instances de la règle de coupure. Cette dernière méthode est celle qui a été suivie par E. Tahhan Bittar dans [TB99]. Dans ce travail, la règle de coupure, appelée Mix, est définie de la façon suivante :

$$\dfrac{\Gamma, \varphi^n \vdash \Delta \qquad \Gamma' \vdash \varphi^m, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\text{Mix}$$

où la notation φ^n signifie $\underbrace{\varphi, \ldots, \varphi}_{n \text{ fois}}$. Pour ce calcul, la règle **R2** définie plus haut devient la règle suivante :

$$\dfrac{\dfrac{\Gamma, \varphi, \varphi \vdash \Delta}{\Gamma, \varphi \vdash \Delta}\text{Contr.} \qquad \Gamma' \vdash \varphi, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\text{Mix} \quad \rightsquigarrow \quad \dfrac{\Gamma, \varphi, \varphi \vdash \Delta \qquad \Gamma' \vdash \varphi, \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\text{Mix}$$

En utilisant l'ordre de précédence

$$\forall @ \in \{\vee, \neg, \exists, \text{Contr.}, \text{Weak.}\}, \text{Mix} \succ @\text{-droit}, @\text{-gauche}, \text{Taut}$$

on montre facilement que la procédure d'élimination des coupures satisfait la condition 1. Bien entendu, en considérant une règle Mix de la forme

$$\dfrac{\Gamma, \varphi^n \vdash \Delta \qquad \Gamma \vdash \varphi^m, \Delta}{\Gamma \vdash \Delta}\text{Mix}$$

on n'obtient qu'un résultat de normalisation faible.

BIBLIOGRAPHIE

[AABLM05] Marc Aiguier, Agnès Arnould, Clément Boin, Pascale Le Gall et Bruno Marre. Testing from algebraic specifications : test data set selection by unfolding axioms. In *Formal Approaches to Testing of Software (FATES'05)*, volume 3997 de *Lecture Notes in Computer Science*, pages 203–217, 2005. Version longue disponible à l'adresse http ://ftp.lami.univ-evry.fr/pub/publications/reports/2005/lami_110.ps.

[AALL07] Marc Aiguier, Agnès Arnould, Pascale Le Gall et Delphine Longuet. Test selection criteria for quantifier-free first-order specifications. In *Fundamentals of Software Engineering (FSEN'07)*, volume 4767 de *Lecture Notes in Computer Science*, pages 144–159, 2007.

[ABD02] Marc Aiguier, Diane Bahrami et Catherine Dubois. On a generalised logicality theorem. In *AISC'2002*, volume 2385 de *Lecture Notes in Artificial Intelligence*, pages 51–64, 2002.

[ABK+02] Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella et Andrzej Tarlecki. CASL : The Common Algebraic Specification Language. *Theoretical Computer Science*, 2(286) :153–196, 2002.

[ABL07] Marc Aiguier, Clément Boin et Delphine Longuet. Abstract proof normalization. Technical report, IBISC, Université d'Évry-Val d'Essonne, 2007. Disponible à l'adresse http ://www.ibisc.fr/~dlonguet.

[AL07] Marc Aiguier et Delphine Longuet. Test case selection from modal specifications of reactive systems. In *Theoretical Aspects of Software Engineering (TASE'07)*, 2007. À paraître.

[AM80] Michael Arbib et Ernest Manes. Machines in a category. *Journal of Pure and Applied Algebra*, 19 :9–20, 1980.

[Bar77] Jon Barwise. *Handbook of mathematical logic*, chapitre « An introduction to first-order logic ». North Holland, Elsevier, 1977.

[Bar05] Fabrice Barbier. *Résultats de théorie abstraite des modèles dans les institutions : vers la combinaison de logiques*. Thèse de doctorat, LaMI, Université d'Évry-Val d'Essonne, 2005.

[BBL97] Gilles Bernot, Laurent Bouaziz et Pascale Le Gall. A theory of probabilistic functional testing. In *International Conference on Software Engineering*, pages 216–226, 1997.

[BCDM03] Fabrice Baray, Philippe Codognet, Daniel Diaz et Henri Michel. Code-based test generation for validation of functional processor descriptions. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 de *Lecture Notes in Computer Science*, pages 569–584, 2003.

[Ber91] Gilles Bernot. Testing against formal specifications : a theoretical view. In *Theory and Practice of Software Development (TAPSOFT'91)*, volume 494 de *Lecture Notes in Computer Science*, pages 99–119, 1991.

[BGM91] Gilles Bernot, Marie-Claude Gaudel et Bruno Marre. Software testing based on formal specifications : a theory and a tool. *Software Engineering Journal*, 6(6) :387–405, 1991.

[Bir35] Garrett Birkhoff. On the structure of abstract algebras. In *Proceedings of The Cambridge Philosophical Society*, volume 31, pages 433–454, 1935.

[Bir46] Garrett Birkhoff. Universal algebra. In *Proceedings of the first Canadian mathematical Congress*, pages 310–326, Toronto, 1946. University of Toronto Press.

[BN98] Franz Baader et Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[BW04] Achim D. Brucker et Burkhart Wolff. Symbolic test case generation for primitive recursive functions. In *Formal Approaches to Software Testing (FATES'04)*, volume 3395 de *Lecture Notes in Computer Science*, pages 16–32, 2004.

[BW05] Achim D. Brucker et Burkhart Wolff. Interactive testing using HOL-TestGen. In *Formal Approaches to Testing of Software (FATES'05)*, volume 3997 de *Lecture Notes in Computer Science*, 2005.

[CH00] Koen Claessen et John Hughes. QuickCheck : a lightweight tool for random testing of Haskell programs. *ACM SIGPLAN Notices*, 35(9) :268–279, 2000.

[CMR99] Maura Cerioli, Till Mossakowski et Horst Reichel. From total equational to partial first order logic. In E. Astesiano, H.-J. Kreowski et B. Krieg-Brückner, éditeurs, *Algebraic Foundations of Systems Specifications*, IFIP State-of-the-Art Reports, pages 31–104. Springer, 1999.

[CoFI04] CoFI (The Common Framework Initiative). *CASL Reference Manual*, volume 2960 de *Lecture Notes in Computer Science*. Springer, 2004.

[DF93] Jeremy Dick et Alain Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *Formal Methods Europe (FME'93)*, volume 670 de *Lecture Notes in Computer Science*, pages 268–284, 1993.

[DJ90] Nachum Dershowitz et Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, éditeur, *Handbook of TCS*, volume B : Formal Models and Semantics, pages 243–320. Elsevier, 1990.

[Dow03] Gilles Dowek. Confluence as a cut-elimination property. In R. Nieuwenhuis, éditeur, *Rewriting Techniques and Applications*, volume 2706 de *Lecture Notes in Computer Science*, pages 2–14, 2003.

[Dra88] Albert Grigorevitch Dragalin. Mathematical intuitionism, introduction to proof theory. *Translations of Mathematical Monographs*, 67 :185–199, 1988. Traduction de l'original russe de 1979.

[DW99] Gilles Dowek et Benjamin Werner. Proof normalization modulo. In *International Workshop on Types for Proofs and Programs*, pages 62–77, 1999.

[FTW04] Lars Frantzen, Jan Tretmans et Tim A. C. Willemse. Test generation based on symbolic specifications. In *Formal Approaches to Software Testing (FATES'04)*, volume 3395 de *Lecture Notes in Computer Science*, pages 1–15, 2004.

[Gal86] Jean Gallier. *Logic for computer science : Foundations of Automatic Theorem Proving.* Wiley, 1986.

[Gau95] Marie-Claude Gaudel. Testing can be formal, too. In *Theory and Practice of Software Development (TASPOFT'95)*, volume 915 de *Lecture Notes in Computer Science*, pages 82–96, 1995.

[GB92] Joseph Goguen et Rod Burstall. Institutions : Abstract model theory for specification and programming. *Journal of the ACM*, 39(1) :95–146, 1992.

[GBR00] Arnaud Gotlieb, Bernard Botella et Michel Rueher. A CLP framework for computing structural test data. In *Computational Logic*, volume 1861 de *Lecture Notes in Computer Science*, pages 399–413, 2000.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. *Mathematische Zeitschrift*, 39(176-210) :405–431, 1935. Traduction anglaise dans M.-E. Szabo, éditeur, The collected Papers of Gerhard Gentzen, pages 68–131, North-Holland, 1969.

[GH78] John Guttag et James Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10 :27–52, 1978.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[GLRT06] Christophe Gaston, Pascale Le Gall, Nicolas Rapin et Assia Touil. Symbolic execution techniques for test purpose definition. In *Testing of Communicating Systems (TestCom'06)*, volume 3964 de *Lecture Notes in Computer Science*, pages 1–18, 2006.

[GLT89] Jean-Yves Girard, Yves Lafont et Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[GM92] Joseph Goguen et José Meseguer. Order-sorted algebra I : Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2) :217–273, 1992.

[GTW78] Joseph Goguen, James Thatcher et Eric Wagner. *Current Trends in Programming Methodology*, volume 4, chapitre « An initial algebra approach to the specification, correctness and implementation of abstract data types ». Prentice Hall, 1978.

[GTWW75] Joseph Goguen, James Thatcher, Eric Wagner et Jesse Wright. Abstract data types as initial algebras and the correctness of data representations. In *Proceedings of the Conference on Computer Graphics, Pattern Recognition and Data Structures*, pages 89–93, 1975.

[Gum99] Peter Gumm. Elements of the general theory of coalgebras, 1999.

[Gut75] John Guttag. *The specification and application to programming of abstract data types*. Thèse de doctorat, Université de Toronto, 1975.

[Her68] Jacques Herbrand. *Écrits logiques*. Presses Universitaires de France, 1968.

[Her95] Hugo Herbelin. A λ-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholskiet J. Tiuryn, éditeurs, *Proceedings of the 8th International Workshop on Computer Science Logic*, volume 933 de *Lecture Notes in Computer Science*, pages 61–75, 1995.

[HR94] Ulrich Hensel et Horst Reichel. Defining equations in terminal coalgebras. In *Recent Trends in Data Type Specification*, pages 307–318, 1994.

[HWB97] Rolf Hennicker, Martin Wirsing et Michel Bidoit. Proof systems for structured specifications with observability operators. *Theoretical Computer Science*, 173(2) :393–443, 1997.

[Jac] Bart Jacobs. Introduction to coalgebras - Towards mathematics of states and observations. Livre en cours de rédaction.

[Jac95a] Bart Jacobs. Mongruences and cofree coalgebras. In *Algebraic Methodology and Software Technology*, volume 936 de *Lecture Notes in Computer Science*, pages 245–260, 1995.

[Jac95b] Bart Jacobs. Objects and classes, co-algebraically. In *Object Orientation with Parallelism and Persistence*, pages 83–103, 1995.

[JJ04] Claude Jard et Thierry Jéron. TGV : theory, principles and algorithms, a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Software Tools for Technology Transfer (STTT)*, 7(4) :297–315, 2004.

[JJRZ05] Bertrand Jeannet, Thierry Jéron, Vlad Rusu et Elena Zinovieva. Symbolic test selection based on approximate analysis. In *Tools and algorithms for the construction and analysis of systems (TACAS'05)*, volume 3440 de *Lecture Notes in Computer Science*, pages 349–364, 2005.

[JM97] Jean-Marc Jézéquel et Bertrand Meyer. Design by contract : The lessons of Ariane. *Computer (IEEE)*, 30(2) :129–130, 1997.

[JR97] Bart Jacobs et Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62 :222–259, 1997.

[Jér01] Thierry Jéron. Le test de conformité : état de l'art. Rapport pour l'AAE (Architecture Électronique Embarquée), 2001.

[Kap84] Stéphane Kaplan. Conditional rewrite rules. *Theoretical Computer Science*, 33 :175–193, 1984.

[KATP02] Pieter W. M. Koopman, Artem Alimarine, Jan Tretmans et Marinus J. Plasmeijer. GAST : Generic automated software testing. In *Implementation of Functional Languages*, volume 2670 de *Lecture Notes in Computer Science*, pages 84–100, 2002.

[Kle52] Stephen Cole Kleene. *Introduction to Meta-mathematics*. North-Holland, Elsevier, 1952.

[Kur00] Alexander Kurz. *Logics for Coalgebras and Applications to Computer Science*. Thèse de doctorat, Fakultät für Mathematik und Informatik, Université de Munich, 2000.

[Kur01] Alexander Kurz. Specifying coalgebras with modal logic. *Theoretical Computer Science*, 260 :119–138, 2001.

[LA96] Jordi Levy et Jaume Agustí. Bi-rewrite systems. *Journal of Symbolic Computation*, 22 :279–314, 1996.

[LA07] Delphine Longuet et Marc Aiguier. Specification-based testing for COCASL's modal specifications. In *Conference on Algebra and Coalgebra in Computer Science (CALCO'07)*, volume 4624 de *Lecture Notes in Computer Science*, pages 356–371, 2007.

[LeG93] Pascale Le Gall. *Les algèbres étiquetées : une sémantique pour les spécifications algébriques fondée sur une utilisation systématique des termes. Application au test de logiciel avec traitement d'exceptions.* Thèse de doctorat, LRI, Université de Paris-Sud, 1993.

[LA96] Pascale Le Gall et Agnès Arnould. Formal specification and test : correctness and oracle. In *11th Workshop on Algebraic Development Techniques (WADT'96)*, volume 1130 de *Lecture Notes in Computer Science*, pages 342–358, 1996.

[LY96] David Lee et Mihalis Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[LZ74] Barbara Liskov et Stephen N. Zilles. Programming with abstract data types. *Notices*, 9 :50–59, 1974.

[MA86] Ernest Maneset Michael Arbib. *Algebraic approaches to program semantics*. Texts and monographs in computer science. Springer-Verlag, 1986.

[Mac00] Patrícia Machado. Testing from structured algebraic specifications. In *Algebraic Methodology and Software Technology (AMAST'00)*, volume 1816 de *Lecture Notes in Computer Science*, pages 529–544, 2000.

[Mar91] Bruno Marre. Toward automatic test data set selection using algebraic specifications and logic programming. In *International Conference on Logic Programming (ICLP'91)*, pages 202–219, 1991.

[Mar95] Bruno Marre. LOFT : a tool for assisting selection of test data sets from algebraic specifications. In *Theory and Practice of Software Development (TAPSOFT'95)*, volume 915 de *Lecture Notes in Computer Science*, pages 799–800, 1995.

[Mes89] José Meseguer. General logics. In H.-D. Ebbinghaus et al., éditeurs, *Proceedings of the Logic Colloquium*, pages 275–329, North-Holland, 1989.

[Mos99] Lawrence S. Moss. Coalgebraic logic. *Annals of Pure and Applied Logic*, 96(1-3) :277–317, 1999.

[MS98] Raymond E. Miller et Junehwa Song. A characterization of the general protocol conformance test sequence generation problem for EFSM's. Technical Report CS-TR-3913, Department of Computer Science, University of Maryland, 1998.

[MS02] Patrícia Machado et Donald Sannella. Unit testing for CASL architectural specifications. In *Mathematical Foundations of Computer Science*, volume 2420 de *Lecture Notes in Computer Science*, pages 506–518, 2002.

[MSRR06] Till Mossakowski, Lutz Schröder, Markus Roggenbach et Horst Reichel. Algebraic-coalgebraic specification in COCASL. *Journal of Logic and Algebraic Programming*, 67 :146–197, 2006.

[ONS93] Fernando Orejas, Marisa Navarro et Ana Sánchez. Implementation and behavioural equivalence : a survey. In *Recent Trends in Data Type Specification*, volume 655 de *Lecture Notes in Computer Science*, pages 144–163, 1993.

[PBG99] Alexandre Petrenko, Sergiy Boroday et Roland Groz. Confirming configurations in EFSM. In *Formal Methods for Protocol Engineering and Distributed Systems (FORTE'99)*, volume 156 de *IFIP Conference Proceedings*, pages 5–24, 1999.

[Pra65] Dag Prawitz. Natural deduction : A proof-theoretical study. *Almqvist and Wiksell, Stockholm*, 1965.

[RdBJ00] Vlad Rusu, Lydie du Bousquet et Thierry Jéron. An approach to symbolic test generation. In *2nd International Workshop on Integrated Formal Method (IFM'00)*, volume 1945 de *Lecture Notes in Computer Science*, pages 338–357, 2000.

[Rei95] Horst Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structures in Computer Science*, 5(2) :129–152, 1995.

[Rus06] Vlad Rusu. *Formal verification and conformance testing for reactive systems*. Habilitation à diriger des recherches, IRISA, Université de Rennes 1, 2006.

[Rut00] Jan Rutten. Universal coalgebra : a theory of systems. *Theoretical Computer Science*, 249 :3–80, 2000.

[Röß01] Martin Rößiger. From modal logic to terminal coalgebras. *Theoretical Computer Science*, 260 :209–228, 2001.

[Sch98] W. Marco Schorlemmer. Term rewriting in a logic of special relations. In *Algebraic Methodology and Software Technology (AMAST'98)*, volume 1548 de *Lecture Notes in Computer Science*, pages 178–195, 1998.

[SP82] Michael Smyth et Gordon Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computation*, 11(4) :761–783, 1982.

[Str96] Georg Struth. Non-symmetric rewriting. Technical report, MPI für Informatik, 1996.

[Tai89] William W. Tait. Normal derivability in classical logic. In J. Barwise, éditeur, *The Syntax and Semantics of Infinitary Languages*, pages 204–236, 1989.

[TB99] Elias Tahhan-Bittar. Strong normalization proofs for cut-elimination in Gentzen's sequent calculi. In *Proceedings of the Symposium on Algebra and Computer Science, Helena Rasiowa in Memoriam*, volume 46, pages 179–225. Banach Center Publication, 1999.

[Tou06] Assia Touil. *Exécution symbolique pour le test de conformité et le test de raffinement*. Thèse de doctorat, IBISC, Université d'Évry-Val d'Essonne, 2006.

[Tre92] Jan Tretmans. *A formal approach to conformance testing*. Thèse de doctorat, Université de Twente, Enschede, Pays-Bas, 1992.

[Tre95] Jan Tretmans. Testing labelled transition systems with inputs and outputs. In *International Workshop on Protocols Test Systems (IWPTS'95)*, 1995.

[UB01] Christian Urban et Gavin M. Bierman. Strong normalisation of cut-elimination in classical logic. *Acta Informatica*, 45(1-2) :123–155, 2001.

[Ven06] Yde Venema. *Handbook of Modal Logic*, volume 3 de *Studies in logic and practical reasoning*, chapitre « Algebras and coalgebras », pages 331–426. Elsevier, 2006.

[vO04] Vincent van Oostrom. Sub-birkhoff. In *7th International Symposium on Functional and Logic Programming*, volume 2998 de *Lecture Notes in Computer Science*, pages 180–195, 2004.

[Wal90] Lincoln A. Wallen. *Automated Deduction for Non Classical Logics*. The MIT Press, 1990.

[Zil74] Stephen N. Zilles. Algebraic specification of data types. Computation structures group memo 119, Laboratory for Computer Science, MIT, 1974.

TABLE DES MATIÈRES