A logical study of program equivalence

Guilhem Jaber

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

MINES Nantes
Institut Mines-Télécom

Thèse de Doctorat

Guilhem JABER

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École nationale supérieure des mines de Nantes
Label européen
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'informatique de Nantes-Atlantique (LINA)

Soutenue le 11 juillet 2014
Thèse n° : 2014EMNA-0124

A Logical Study of Program Equivalence

JURY

Rapporteurs :	**M. Lars BIRKEDAL**, Professor, Aarhus University
	M. Martin HOFMANN, Professor, LMU München
Examinateurs :	**M. Nick BENTON**, Senior Researcher, Microsoft Research Cambridge
	M. Pierre COINTE, Professeur, École des Mines de Nantes
	M. Pierre-Louis CURIEN, Directeur de Recherche CNRS, PPS, Université Paris 7
Directeur de thèse :	**M. Alexandre MIQUEL**, Professeur, Universidad de la República - Montevideo
Co-directeur de thèse :	**M. Nicolas TABAREAU**, Chargé de Recherche, Inria Rennes-Bretagne-Atlantique

Remerciements

Cette thèse doit énormément à l'encadrement remarquable de Nicolas et à sa disponibilité constante. Sa capacité à comprendre en quelques instants des idées sur lesquelles j'ai pu passer des journées, permettant très souvent de me débloquer, continue de m'impressionner. Je pense que je me souviendrai longtemps de nos heures passées derrière un tableau (parfois avec un peu de Jazz comme accompagnement :) à démêler l'écheveau des faisceaux ou le dépliage de la biorthogonalité. Qu'il en soit ici grandement remercié !

Les cours d'Alexandre m'ont conduit à aimer la logique, sa manière toujours claire de présenter des notions ô combien profondes, de nous en faire comprendre la *substantifique moelle*, est un modèle pour moi. C'est donc un honneur qu'il ait accepté de diriger cette thèse.

Je remercie considérablement Lars Birkedal et Martin Hofmann d'avoir accepté de rapporter cette thèse dans un délai aussi court, Merci à eux pour leurs précieux commentaires sur ce manuscrit. Merci également à Lars pour son accueil chaleureux lors de mes séjours à Copenhague.

Merci beaucoup à Nick Benton pour le séjour très enrichissant que j'ai pu effectuer à Cambridge sous sa direction, et pour sa présence dans ce jury.

Je suis également très honoré de pouvoir avoir Pierre Cointe et Pierre-Louis Curien comme membres de ce jury.

J'ai eu le plaisir (bien que trop rare !) d'avoir eu un certain nombre de discussions scientifiques liées à cette thèse, Merci pour cela à Matthieu, Paul-André, Alois, Valentin, Pierre, Kévin et Rémi.

Durant ces années à Nantes, j'ai rencontré des personnes souvent un peu folles, toujours très attachantes, et généralement portées sur la bouteille, avec qui j'ai partagé de très beaux moments! Ainsi, l'amitié sans faille de Cauê, Francesca, Lilly et Maen m'est très précieuse. Merci également aux zicos de Who's Next, aux colocs rue de Verdun et de la Chauvinière, Giulio, Jerem, Sarah, Brendaõ, Estelle, Solenne, Sophie, Vincent, Julie et

Antoine, mais aussi tous les autres, Amélie, Diego, Hugo et Diana.

Il ne faut pas non plus oublier d'où l'on vient (le sud !), Merci donc à Biquette, Dadou, Lolotte et Maxou pour tout les bon moments, sans oublier Anne-Lise, Emily et Tiphanie qui savent calmer leurs ardeurs :)

Et puis, Merci à Marie-Laure qui m'a soutenu pendant ces difficiles derniers mois de thèse, et pour tout ce qui est à venir!

Enfin, Merci à toute ma famille, ma sœur (et le petit Liam :) et mes parents, pour leur soutien constant, et sans qui rien de tout ça n'aurait pu voir le jour.

Contents

1

Résumé

Alors que les programmes informatiques jouent un rôle de plus en plus important au sein de systèmes critiques, dans les avions ou les équipements médicaux, le besoin de méthodes formelles pour s'assurer de leur correction devient inévitable. Parmi ces méthodes formelles, les preuves mathématiques de correction de programmes donnent le plus haut degrés de confiance que l'on peut espérer. Mais de telles preuves peuvent être excessivement complexe, s'assurer qu'elles sont correctes requière donc une vérification automatique. C'est le but des *assistants de preuves*, qui sont eux-mêmes des programmes informatiques conçus pour vérifier si une preuve d'un théorème est correcte. Par ailleurs, pour résonner sur des programmes, il est nécessaire de leur donner un *sens* précis, ce qui peut être fait en définissant une sémantique pour le langage de programmation dans lequel ils sont écrit. Dans cette thèse, à la fois le langage de l'assistant de preuve et le langage de programmation proviennent de la même notion centrale: le λ-calcul.

Fonctions et Types

Le λ-calcul a été introduit par A. Church en tant que modèle de calcul dans lequel "tout est fonction", où par fonction on entend un objet syntaxique pour lequel la liaison de variable est gérée explicitement. L'application d'une fonction à un argument est ensuite mise en œuvre par la *substitution* de variables. De telles fonctions, appelées λ-abstractions, peuvent être annotées par des types pour s'abstraire de leur calcul. Plus précisément, on donne à une fonction un type $A \to B$ pour exprimer qu'elle prends en argument un élément de type A et retourne un élément de type B. C'est ainsi qu'est défini le *λ-calcul simplement typé*. Ce langage joue un rôle crucial à la fois dans le domaine de la théorie

de la démonstration et des langage des programmation.

En effet, dans la théorie de la démonstration, ce langage est utilisé pour représenter preuves et formules via la fameuse *correspondance de Curry-Howard*. L'idée de base est qu'un terme de type $A \to B$ représente une preuve de l'implication $A \Rightarrow B$. Ainsi, vérifier la correction d'une preuve revient à vérifier le type du terme correspondant. Suivant cette idée, les travaux de P. Martin-Löf sont d'une importance capitale, en étendant cette correspondance au calcul des prédicats. Ainsi, on représente une formule quantifiée universellement $\forall x \in A.P(x)$ par un *type dépendant* $\Pi x : A.P$. Pour définir de tels types dépendants, on doit autoriser les termes à apparaître dans les types. Par exemple, on peut définir le type $\mathcal{M}_{m,n}$ des matrices rectangulaires de taille $m * n$, de telle manière que la multiplication de matrices sera de type $\mathcal{M}_{m,l} \to \mathcal{M}_{l,n} \to \mathcal{M}_{m,n}$. Ici, on utiles les entiers l, m, n (qui sont des termes) pour affiner le type des matrices, leur donnant un un pouvoir de spécification plus précis.

Ce système, appelé la *théorie des types de Martin-Löf*, est à la base d'assistants de preuve comme Coq ou Agda. Ces systèmes satisfont le fameux *critère de De Bruijn*, c'est à dire qu'elles génère un témoin de preuve, à savoir un λ-terme, qui peut être ensuite vérifier indépendamment, sans avoir à se fier aveuglément à l'assistant de preuves.

Le λ-calcul est également à l'origine des *langages fonctionnels*. Ils reposent sur le concept originel du λ-calcul, à savoir un modèle de calcul, pour obtenir un langage de programmation. Ce fut tout d'abord fait dans une version non typé par J. McCarthy avec LISP, puis ensuite des *langages fonctionnels typés* furent conçus, comme ML ou Haskell. Les types sont particulièrement utiles pour spécifier les programmes, suivant le fameux slogan de R. Milner:

"*Les programmes bien typés ne peuvent pas se tromper.*"

Il s'agit la de la fameuse propriété de *sûreté de typage*. Elle peut être prouvée de deux manières différentes, soit de manière purement syntaxique, via la méthode de "progrès et préservation", ou alors en associant à chaque type τ un ensemble de termes $[\![\tau]\!]$ qui sont sûrs, ce qui donne lieu à la méthode des *types sémantiques*, aussi appelée méthode de *réalisabilité*.

Le λ-calcul sera ainsi utilisé pour construire l'objet d'étude de cette thèse—les programmes fonctionnels— mais également pour représenter les preuves, sur ces objets, via la correspondance de Curry-Howard, en utilisant une théorie des types dépendants. Cependant, alors que les programmes que l'on considère sont *impurs*—c'est à dire qu'ils peuvent effectuer des effets de bord tels la divergence ou l'utilisation d'une mémoire persistante— les preuves sont elles représentées par des λ-termes *purs*, qui vérifient entre autres la propriété de terminaison [1].

Comme expliqué précédemment, on peut spécifier un programme fonctionnel via son type. Mais le système de types doit être suffisamment expressif pour permettre d'énoncer

1. Par terminaison, on veut dire qu'il n'existe pas de termes qui peuvent être réduits indéfiniment.

une spécification précise. Cela peut par exemple être fait en utilisant des “refinement types” [FP91] ou même des types dépendants [NMB08].

Dans cette thèse, on défendra plutôt l'idée que dans beaucoup de cas, le meilleur langage pour écrire la spécification d'un programme est le langage de programmation en lui-même. En effet, lorsque l'on écrit un programme, on a souvent tendance à d'abord écrire une version naïve en laquelle on a toute confiance, que l'on va ensuite optimiser. Dans ce cas, prouver que la version naïve est équivalente à la version optimisée nous donnerait un résultat de correction pour la version optimisée.

Bien sûr, pour des programmes plus complexes, cette alternative n'est pas toujours possible. Les compilateurs sont une classe d'exemples particulièrement intéressante. Leur correction est cruciale, puisqu'ils produisent le code de bas-niveau qui sera exécuté en pratique sur les ordinateurs. Ainsi, un bug dans un compilateur pourra anéantir tous les efforts effectués pour prouver qu'un programme satisfait sa spécification, en générant un code bas-niveau incorrect. Il ne s'agit pas la d'une vue de l'esprit, comme le montre les travaux dans [YCER11], où des dizaines de bugs furent trouvés dans GCC ou LLVM. Ainsi, de nombreux travaux sur la formalisation de la correction de compilateurs ont été effectués. Parmi ceux-ci, les travaux de X. Leroy et al. avec CompCert [Ler09] sont sûrement les plus impressionnants. Dans cette formalisation, la spécification du compilateur correspond à une simulation entre les sémantique des langages haut- et bas-niveau. Cependant, on peut imaginer des spécifications de compilateurs plus “modulaires”, qui permettraient l'interaction entre du code bas-niveau provenant de deux compilateurs différents, ou même écrit à la main. Suivant les travaux de N. Benton et al. [BT09, BH09], il est possible de définir une telle spécification modulaire en utilisation la conservation de l'équivalence de programmes vers le code bas-niveau. Cela est particulièrement intéressant pour prouver la correction d'optimisations effectuées par les compilateurs. Tout en travaillant sur cette idée [JT10, JT11], la nécessité d'un bon cadre logique pour formaliser les preuves d'équivalences de programmes nous est paru évidente. Ce fut le point de départ de cette thèse.

Équivalence Contextuelle

Nous avons choisi de nous concentrer sur la notion d'*équivalence contextuelle*, aussi appelée *équivalence observationnelle*, pour les langages fonctionnels impurs. Dans ce cadre, deux programmes sont dit équivalents si aucun contexte ne peut les distinguer. Au delà de son intérêt pratique, qui a été présenté précédemment, cette notion est fondamentale en sémantique dénotationnelle qui a pour but de construire des modèles *pleinement abstraits*. Dans de tels modèles, les dénotations de deux termes sont égales si et seulement si les deux termes sont contextuellement équivalents.

Comme on peut l'imaginer, le pouvoir expressif des contextes, vu comme des programmes “à trous”, joue un rôle majeur dans l'étude de l'équivalence contextuelle. Dans le cas dégénéré où les contextes peuvent inspecter le code source des programmes, comme

c'est le cas pour les langages de programmation bas-niveau ou du code assembleur, cette notion d'équivalence contextuelle s'effondre complètement vers l'égalité syntaxique: un programme ne peut être équivalent qu'à lui-même.

Dans la direction opposée, lorsque les contextes sont supposés être "purs", l'étude de l'équivalence contextuelle devient rapidement indécidable (même sans aucune forme de récursion ou de types de donnés infinis), comme l'a prouvé Loader dans [Loa01]. D'une certaine manière, cela expliqué pourquoi la quête d'une sémantique dénotationnelle pleinement abstraite pour PCF [2] fut beaucoup plus dur que pour PCF avec effets de bord.

Ici, nous nous intéressons à des contextes et des programme qui disposent d'une notion de mémoire mutable. Plus précisément, nous considérons la notion de *références*, qui est la manière standard d'implémenter les cellules mémoires dans des langages dérivés de ML, comme OCaml. Une référence est définie via le constructeur ref v, pour stocker une valeur v dans le tas (alors que les arguments fournis aux fonctions sont, de manière standard, stockés dans la pile). Ces références sont ensuite réduites vers des locations, qui peuvent être vues comme des adresses mémoires. Cependant, comparé aux langages bas-niveau comme C, on ne peut effectuer aucune forme d'arithmétique de pointeurs sur les locations: elles doivent être vues comme des noms abstraits pour les cellules mémoires. Les valeurs stockées dans les références peuvent être d'ordre supérieur, c'est à dire des clôtures. Cela permet d'encoder la récursion.

En utilisant les références pour disposer d'une mémoire globale (donc persistante), une fonction peut alors compter combien de fois elle est appelée. Ainsi, lorsqu'elle est appelée deux fois avec le même argument, elle peut potentiellement retourner deux résultats différents. Mais de la même manière les contextes peuvent utiliser les références, ainsi deux termes sont équivalents s'ils effectuent des appels de fonctions, fournies par les contextes, d'une manière équivalente, et avec des arguments équivalents. C'est l'idée de la *synchronisation des callbacks*, qui est fausse lorsque les contextes sont purs.

Relations logiques à la Kripke

En général, il est difficile de raisonner sur l'équivalence contextuelle à cause de la quantification sur tous les contextes présente dans sa définition. Certaines techniques ont été introduites pour éviter cette quantification, et ainsi prouver l'équivalence contextuelle d'une manière plus simple. Les relations logiques sont un des cadres de travail les plus utilisés pour cela, qui sont également d'une importance fondamendale pour définir la notion de *paramétricité*, suivant en cela les travaux fondateurs de J. Reynolds [Rey83]. Les relations logiques sont une généralisation des types sémantiques, utilisés pour prouver la sûreté de typage, à un cadre relationnel (plutôt que prédicatif). Ce sont des relations binaires $\mathcal{V}[\![\tau]\!]$ et $\mathcal{E}[\![\tau]\!]$ définies respectivement sur les valeurs et les termes de type τ. La relation $\mathcal{V}[\![\tau]\!]$ est définie par induction sur τ.

2. un λ-calcul simplement typé avec entiers naturels et récursion

Sur les types de base, la relation impose directement l'égalité des valeurs. Sur les types fonctionnels $\tau \to \sigma$, un couple de valeurs $\lambda x_1.M_1, \lambda x_2.M_2$ est dans $\mathcal{V}[\![\tau \to \sigma]\!]$ lorsque :

$$\forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!], ((\lambda x_1.M_1)v_1, (\lambda x_2.M_2)v_2) \in \mathcal{E}[\![\sigma]\!]$$

Par ailleurs, deux termes sont dans $\mathcal{E}[\![\tau]\!]$ lorsqu'ils se réduisent vers deux valeurs qui sont dans $\mathcal{V}[\![\tau]\!]$. En utilisant ces définitions, on peut montrer que deux termes de type τ sont contextuellement équivalents lorsqu'ils sont dans $\mathcal{E}[\![\tau]\!]$.

Cependant, on ne peut étendre directement cette définition à un langage avec références. En effet, dans ce cas, la réduction d'un terme M dépend des valeurs stockées par les références, c'est à dire dans le tas. Ainsi, A. Pitts et I. Stark [PS98] ont proposé de paramétrer la définition de $\mathcal{E}[\![\tau]\!]$ et de $\mathcal{V}[\![\tau]\!]$ par des invariants sur les tas. De tels invariants sont appelés *mondes*, suivant en cela l'intuition des modèles de Kripke. La notion de mondes a ensuite été développé par A. Ahmed [ADR09], puis par D. Dreyer et al. [DNB12] en utilisant des systèmes de transitions d'états, où chaque état représente les invariants sur le tas, permettant ainsi de prouver des équivalences qui étaient hors de portées des travaux de Pitts et Stark.

La Technique du Step-Indexing

Les relations logiques à la Kripke définies par Pitts et Stark le furent sur un langage avec "références de base", c'est à dire des références qui peuvent stocker uniquement des valeurs de base mais pas des fonctions. Dans sa thèse [Ahm04], A. Ahmed a étendu ces définitions aux types récursifs et aux références d'ordre supérieur. Pour cela, elle a du gérer un problème de circularité dans la définition des mondes et des relations logiques. Appel et McAllester [AM01] avaient déjà rencontré le même genre de problème de circularité dans leurs travaux sur la sûreté de typage pour du code bas-niveau avec types récursifs. Ainsi, la définition des types sémantiques TypeSem, qui sont des ensemble de valeurs "bien élevées", dépend de la définition des mondes World qui sont des invariants sémantiques sur ce que les locations peuvent stocker. Écrites sous forme d'équations récursives, cela donne

$$\text{World} \stackrel{def}{=} \text{Loc} \to \text{TypeSem} \text{ et } \text{TypeSem} \stackrel{def}{=} \text{World} \to \text{Value}$$

Mais alors, en dépliant la définition de TypeSem, on obtient que TypeSem est égal à (Loc $\to$ TypeSem) $\to$ Value. On souhaite donc définir TypeSem sous la forme d'un point fixe $\mu X.(\text{Loc} \to X) \to \text{Value}$. Cependant, X se trouve en position négative dans cette définition, donc en général un tel point fixe n'existe pas. En effet, on ne peut pas appliquer le théorème de *Knaster-Tarski*.

Ce genre de définition récursive peut être résolu en passant d'un cadre ensembliste à un cadre catégorique. C'est ce qui est usuellement fait en sémantique dénotationnelle [SS71]. Plus précisément, en *théorie des domaines* [Sco82], on travaille dans la catégorie des

ordres partiels complets, où de tels équations peuvent être résolues. Ainsi, on peut traiter l'équation $D = D \to D$ correspondant au λ-calcul pur.

Cependant, dans cette thèse on souhaite raisonner de manière *operationelle* sur les programmes, alors que la sémantique dénotationnelle interprète les programmes sous la forme d'objets mathématiques abstraits. Pour rester dans un monde opérationnel, Appel et McAllester[AM01] ont introduit la notion de "step-index" pour résoudre ces problèmes de circularité. L'idée de base est de stratifier la définition des types sémantiques avec un entier naturel représentant, en première approche, le nombre d'étapes de réduction pour lesquelles le programme en question se comporte correctement. Ainsi, la définition de TypeSem et de World est maintenant indexée par un entier n, comme le montre la définition suivante:

$$\text{World}_{n+1} \stackrel{def}{=} \text{Loc} \to \text{TypeSem}_n \text{ et } \text{TypeSem}_n \stackrel{def}{=} \text{World}_n \to \text{Value}_n$$

Cette idée a été utilisée par Ahmed pour définir les relations logiques, tout d'abord pour les langages avec types récursifs [Ahm06] puis pour les langages avec références d'ordre superieur, dans un travail fait en collaboration avec Dreyer et Rossberg [ADR09]. Dans tous ces travaux, la technique du step-indexing utilisée dans les définitions des relations logiques représente le nombre d'étapes de réduction pour lesquelles les deux programmes en question sont équivalents. De cette manière, il devient possible de prouver l'équivalence de nombreux programmes qui utilise des traits récursifs, en effectuant une simple induction sur les step-indexes.

Mais la gestion des step-indexes dans une preuve est en pratique —pour citer N. Benton—"laide" ("ugly"). Dans leurs travaux sur un "modèle très modal" (a "very modal model") [AMRV07], Appel, Melliès, Richard et Vouillon ont proposé un cadre pour raisonner de manière abstraite sur les step-indexes en utilisant la logique de Gödel-Löb. Pour cela, ils ont introduit une modalité "later", notée $\triangleright$, dont le sens, (défini à l'aide d'une *sémantique de Kripke*) est "vrai dans le futur". Ainsi donc, la *règle de Löb* $(\triangleright P \to P) \to P$, qui peut être vue comme un principe d'induction, est valide dans cette logique. On peut alors l'utiliser pour raisonner de manière aisée sur des programmes récursifs, comme on l'aurait fait avec des step-indexes explicites.

Cette modalité fut en fait utilisée antérieurement par Nakano [Nak00, Nak01] pour définir un λ-calcul avec des types récursifs généraux tout en étant fortement normalisant. Pour garantir cette propriété de normalisation, la modalité $\triangleright$ (qui est notée $\bullet$ dans ces travaux) est utilisée pour "garder" les types récursifs.

Traduction de Forcing

Travaillant sur le step-indexing et sa reformulation via la logique de Gödel-Löb, il nous a alors semblé clair que l'on pouvait voir ces techniques comme des instances de *forcing*. Par forcing, on entend une méthode générale, introduite à l'origine par P. Cohen

[CD66], pour prouver l'indépendance de l'hypothèse du continu. Cette méthode a ensuite été reformulée sous de nombreuses autres formes. La présentation qui s'intègre le mieux dans notre cadre de travail est celle utilisant les *topos de faisceaux* ou de *prefaisceaux*, issue des travaux effectués par Lawvere et Tierney [Tie72].

Cette intuition d'un lien entre step-indexing et forcing fut ensuite confirmé par les travaux de Birkedal et al. sur le *topos des arbres* [BMSS11], où ils construisent un modèle d'une théorie des types avec *types récursifs gardés* en utilisant des prefaisceaux sur les entiers naturels. Par ailleurs, la manière de définir les relations logiques à la Kripke comme étant des relations logiques indexées par des mondes est aussi, comme on peut si attendre, un exemple de forcing, en utilisant la sémantique de Kripke pour les logiques modales. Cette idée a été explorée par Dreyer et al. in [DNRB10], où ils définissent une logique modale pour raisonner sur les relations logiques à la Kripke.

Notre but fut donc d'être capable d'étendre une théorie des types de manière modulaire— plus précisément la théorie des types sous-jacente à l'assistant de preuves Coq— en utilisant cette notion de construction de forcing. Cela nous a conduit à construire une méthode générale pour définir une traduction de forcing sur l'ensemble de la théorie des types considérée, en utilisant une internalisation de la construction de préfaisceaux, paramétrée par un ensemble de *conditions de forcing*. Une telle construction nous permet de définir des *couches de forcing*, c'est à dire des extensions de la théorie de base avec de nouveaux principes logiques, tout en conservant les propriétés cruciales suivantes:

— la consistance de la théorie,
— la decidabilité du type-checking,
— la propriété de canonicité, c'est à dire le fait que les termes clos se réduisent sur des valeurs.

Une conservation des deux derniers résultats ne peut en général être obtenu via des méthodes sémantiques.

En instanciant l'ensemble des conditions de forcing par les entiers naturels, nous pouvons ainsi définir ce que nous appelons la *couche de step-indexing*, où la construction des types récursifs gardés issus de [BMSS11] devient possible. En travaillant avec une théorie des types munies d'univers, nous avons en outre pu définir un opérateur de point fixe sur les types, qui permet de définir de manière interne de tels types récursifs gardés.

Définition directe et par biorthogonalité

Une autre technique essentielle dans la définition des relations logiques est la *biorthogonalité*, aussi appelé $\top\top$-clôture. La biorthogonalité a une longue histoire en théorie de la démonstration. Ainsi, elle a été utilisée par J.-Y. Girard pour prouver l'élimination des coupures de la logique linéaire, et par J.-L. Krivine dans ses travaux sur la réalisabilité classique, pour étendre la correspondance de Curry-Howard à la logique classique avec l'axiome du choix.

Dans le cadre des relations logiques, elle est utilisée pour définir $\mathcal{E}[\![\tau]\!]$ non pas directement par rapport à $\mathcal{V}[\![\tau]\!]$, mais plutôt comme clôture de $\mathcal{V}[\![\tau]\!]$ via une relation auxiliaire $\mathcal{K}[\![\tau]\!]$ sur les contextes d'évaluations.

Cette définition sous forme de clôture est utile pour trois différentes raisons:

— pour raisonner sur des langages disposant de primitives dont la réduction est dépendante du contexte, comme `call/cc` ou l'instruction `jump` en assembleur,

— pour construire des modèles opérationnels de langages combinant polymorphisme et récursion [Pit00], de manière à garantir l'*admissibilité* des relations logiques,

— pour obtenir la complétude des relations logiques, c'est à dire que deux termes de type τ qui sont contextuellement équivalents sont dans $\mathcal{E}[\![\tau]\!]$.

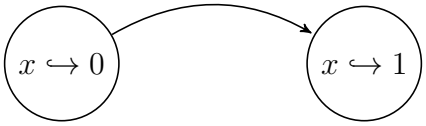
Dans cette thèse, nous ne considérerons pas d'opérateurs de contrôle tel que `call/cc`, ainsi le flot de contrôle sera supposé séquentiel. Pour gérer la présence de récursion dans le langage, nous avons vu que le step-indexing est en pratique suffisant, ainsi nous n'aurons pas besoin de la notion d'admissibilité. Ainsi, seul l'obtention de la complétude des relations logiques par rapport à l'équivalence contextuelle pourrait justifier l'utilisation de la biorthogonalité ici.

Cependant, lorsque l'on travaille avec des relations logiques définies par biorthogonalité, on s'appuie sur le *principe d'invariance locale* [PS98] qui permet de déplier la biorthogonalité vers une définition directe.

Mais la complétude obtenue par biorthogonalité est un peu trompeuse, puisqu'elle ne signifie pas que l'on peut en pratique prouver n'importe quelle équivalence correcte en utilisant ces relations logiques. C'est ce qu'illustre le fameux "awkward example" :

$$\begin{array}{lcl} M_1^{awk} & = & \texttt{let x = ref 0 in } \lambda \texttt{f.x} := \texttt{1}; \texttt{f}(); \, !x \\ M_2^{awk} & = & \lambda \texttt{f.f}(); \, 1 \end{array}$$

Pour prouver que ces deux termes sont équivalents, l'invariant qui contraint x à pointer soit vers 0 ou vers 1 n'est pas suffisant pour déduire que le premier programme retourne 1. Ainsi, le modèle de Pitts et Stark, qui utilise des invariants comme mondes, ne peut pas prouver cette équivalence, ce qui peut sembler étrange à première vue puisque ces relations logiques sont complètes. Cela vient du fait qu'utiliser la clôture par biorthogonalité pour prouver une équivalence est aussi dur que de raisonner directement sur l'équivalence contextuelle, lorsque l'on ne peut utiliser le principe d'invariance locale. En fait, l'impossibilité de prouver cette équivalence avec les définitions de Pitts et Stark a mené Ahmed à raffiner leur modèle, pour permettre aux invariants sur les tas d'évoluer durant l'exécution [ADR09]. En utilisant les travaux ultérieurs de Dreyer et al. [DNB12], le monde utilisé pour prouver cette équivalence est le système de transitions d'états suivant:



qui s'assure que lorsque x pointe vers 1, il ne pointera plus jamais vers 0. En utilisant ce principe plus précis, il devient ensuite possible de prouver que M_1 est équivalent à M_2. De tels systèmes de transitions peuvent en pratique être vue sous la forme d'une abstraction du flot de contrôle du programme, ne prenant en compte que la modification du tas.

Sémantique des jeux nominale opérationnelle

En absence de la biorthogonalité, prouver la complétude des relations logiques pour un langage avec références est un problème ouvert. En effet, on doit alors raisonner sur la forme des contextes, ce qui peut être très complexe. Pour traiter ce problème, nous avons choisi d'établir un lien entre les relations logiques et le modèle dénotationnel de notre langage. La sémantique des jeux est alors apparue comme étant le choix le plus évident pour cela, plus précisément le modèle développé par A. Murawski et N. Tzevelekos pour des "bonnes références générales" [MT11b] [3] qui est en effet pleinement abstrait pour RefML. Ce modèle utilise des *ensembles nominaux* [Pit13] pour abstraire le raisonnement sur les "noms" des locations. Nous utiliserons également cette idée tout au long de cette thèse.

Ce modèle ne s'intègre cependant pas facilement avec les relations logiques, comme on peut le voir sur l'exemple suivant, représenté par le terme M:

$$\lambda\, \texttt{f.let x} \;=\; \texttt{ref0 in f}(\lambda\_.\texttt{x} := \texttt{1})\ \texttt{in}\ M_1$$

Ce terme conserve la référence x privée, et fournit au contexte la possibilité de la modifier à 1 à l'aide d'une fonction "setter". Mais ensuite, il n'est plus possible de relier M_1 à M pour deux raisons :

— x est gardée privée par le terme, mais la dénotation de M_1 est défini comme si toutes ses locations étaient divulguées au contexte,

— le contexte a accès au setter $\lambda\_.\texttt{x} := \texttt{1}$ lorsque M_1 est évalué, mais il n'y a pas de manière facile de le spécifier avec la sémantique des jeux.

Pour dépasser ces limitations de la sémantique des jeux, nous avons choisi de définir un nouveau modèle, inspiré par les travaux antérieurs de J. Laird [Lai07]. Cette sémantique dispose des propriétés dénotationnelles (ou catégoriques) de la sémantique des jeux, tout en ayant des bonnes propriétés opérationnelles. L'idée de base de cette sémantique est d'interpréter les termes comme des ensembles de *traces*, représentant les interaction entre un terme et n'importe quel contexte, comme en sémantique des jeux, mais ici cette interprétation peut être calculée opérationnellement grâce à une *réduction interactive*.

En raffinant ensuite ce modèle avec des mondes pour contraindre la forme des tas, on peut définir une *sémantique des traces à la Kripke*. Cela nous permet d'établir un lien forme avec les relations logiques à la Kripke et ainsi de prouver leur correction et complétude.

3. "bonne" s'oppose ici aux "bad-variables" qui proviennent de la représentation sous forme d'objets des références, inspirée par les travaux sur Idealized Algol par Reynolds.

Une logique temporelle pour l'équivalence de programmes

Des logiques ont été conçues pour raisonner sur la corrections de programmes, fournissant des principes généraux pour prouver de tels résultats. La plus utilisée est certainement la *logique de Hoare*, qui fut ensuite étendue avec la *logique de séparation* [Rey02] pour raisonner aisément sur les propriétés du tas.

Dans cette thèse, nous nous intéressons particulièrement à des logiques pour raisonner directement sur l'équivalence de programmes. La logique de Plotkin et Abadi pour le polymorphisme paramétrique [PA93] fut la première logique introduite dans ce but, pour le Système F. Puis Dreyer et al. ont introduit LSLR [DAB09] et LADR [DNRB10], deux logiques pour des langages plus complexes, avec types récursifs et références d'ordre supérieur. Cependant, toutes ces logiques utilisent le λ-calcul comme objet de base, ce qui leur permet ensuite d'y définir les relations logiques. Si ces logiques conviennent parfaitement pour formaliser des preuves d'équivalence, il semble très difficile de les utiliser pour prouver *automatiquement* l'équivalence de programmes. Dans cette thèse, nous suivons une approche différente, en définissant une logique beaucoup plus simple qui satisfait de bonnes propriétés de "model-checking" pour abstraite le raisonnement sur les mondes utilisés dans notre définition des relations logiques à la Kripke. Cependant, comparé au step-indexing, cette abstraction ne rentre pas aisément dans le cadre de la traduction de préfaisceaux, car certaines des constructions utilisées pour définir les relations logiques à la Kripke ne sont pas monotones par rapport aux mondes.

Ainsi, on définit plutôt une *logique temporelle* pour raisonner sur les systèmes de transitions des mondes. La sémantique des différents connecteurs de cette logique temporelle est alors définie de manière usuelle à l'aide d'une sémantique de Kripke. Comparé à [DNB12], nous utilisons des systèmes de transition étiquetées (Labeled Transition Systems) plutôt que des systèmes de transitions d'états. Les invariants sont alors spécifiés directement sur les transitions, en utilisant une notion de pre- et post-condition dans l'esprit de la logique de Hoare. Cella est particulièrement utile pour garder les systèmes de transitions finis.

Dans les travaux antérieurs [PS98, ADR09, DNB10], un monde futur w' de w pouvait être étendu avec de nouveaux invariants (ou de nouveaux systèmes de transitions d'invariants) sur la partie des tas qui sont disjoints de ceux spécifiés par w. Ainsi, les mondes étaient partitionnés en *îles*, chacune correspondant à une partie du tas disjointe des autres. Cependant, cela nécessite une trop grande liberté pour être définissable dans notre logique restreinte, où l'on ne peut quantifier sur des objets complexes comme des invariants. Ainsi; on interdit ici cet usage, le système de transitions étant fixé au départ, et ne peut changer au fur et à mesure que le monde évolue. Cela veut dire qu'il doit prévoir les création de locations futures effectuées par le terme: on appelle un tel comportement *omniscient*.

Pour raisonner de manière abstraite sur les variables libres de base, on introduit une *exécution symbolique* pour notre langage qui génère toutes les réductions possibles des termes. Il génère également un ensemble de prédicats qui contraignent les valeurs des vari-

ables de base, de telle manière que l'ensemble des prédicats est satisfiable si et seulement si la réduction est effectivement possible. Ensuite, on définit les *relations logiques temporelles* $\mathbb{E}[\![\tau]\!]$, qui sont des formules de notre logique temporelle. A partir de deux termes M_1, M_2, la formule $\mathbb{E}[\![\tau]\!](M_1, M_2)$ est générée automatiquement en utilisant l'exécution symbolique. On montre ensuite qu'il existe un monde qui valide cette formule si et seulement si les deux termes appartiennent à la relation logique "concrète" vu précédemment. De la correction et la complétude des relations logiques concrètes, on réduit le problème de l'équivalence contextuelle de deux termes M_1, M_2 au model-checking de $\mathbb{E}[\![\tau]\!](M_1, M_2)$, c'est à dire:

M_1, M_2 sont contextuellement équivalents si et seulement si il existe un monde w tel que
$$w \models \mathbb{E}[\![\tau]\!](M_1, M_2)$$

Notre logique temporelle combine l'arithmétique de Presburger, utilisée pour énoncer des contraintes arithmétiques sur les valeurs de base, avec des modalités temporelles inspirées de la *Computational Tree Logic* pour contrôler l'évolution de ces contraintes. Par conséquent, elle n'a pas de bonne propriétés de décidabilité du model-checking que les logiques temporelles ont usuellement, à cause de l'interaction avec l'arithmétique.

Cependant, sous certaines hypothèses raisonnable sur la forme des mondes, on obtient la décidabilité du model-checking, c'est à dire que l'on peut vérifier automatiquement si un monde valide une formule temporelle. Pour ce faire, on traduit la formule temporelle vers une formule plus simple, qui ne contient pas de modalité temporelle, et qui peut ensuite être vérifiée par un solveur SMT.

Il semble être possible, sous une hypothèse de borne de la taille du tas, de construire automatiquement un monde qui valide la formule temporelle si et seulement si les deux programmes sont effectivement contextuellement équivalent. Cela nous permettrait d'obtenir un résultat de décidabilité[4] pour l'équivalence contextuelle pour des termes satisfaisant cette hypothèse. Nous pensons que cela permettrait d'obtenir un fragment décidable qui inclut celui de la sémantique des jeux algorithmique [MT11a], qui est lui caractérisé par une restriction sur les types.

A la suite de ces résultats théoriques, une implémentation d'un prototype a été réalisée en Haskell, qui permet de:

— générer la formule temporelle correspondant à l'équivalence de deux termes,
— traduire cette formule en une formule plus simple, qu'un solveur SMT peut valider, une fois que le monde est fourni.
— générer, sous une hypothèse de borne de la taille du tas, un monde qui valide la formule si et seulement si les deux termes sont équivalents.

4. Notons que, comme prouvée dans [Mur05, MT12], l'équivalence contextuelle est en général indécidable pour un langage avec références, même en absence de récursion ou de types de donnés infinis.

Chapitre 3: Préliminaires

Dans ce chapitre, on introduit les prérequis utilisés tout au long de cette thèse.

Théorie des Types de Martin-Löf

Tout d'abord, nous introduisons dans la section 3.1 la théorie des types qui est utilisée comme "métathéorie" pour représenter nos preuves: la *théorie des types de Martin-Löf*.

Suivant l'idée sous-jacente de la correspondance de Curry-Howard, les preuves peuvent être représentées comme des termes du λ-calcul, et les formules comme des types. Cette idée fut développée par P. Martin-Löf pour construire un formalisme constructif aux mathématiques. Comme nous l'avons expliqué précédemment, il a introduit les types dépendants comme représentation des quantificateurs existentiels et universels.

Cette section introduit $\text{MLTT}_{\mathcal{U}}$, la théorie des types de Martin-Löf avec une hiérarchie d'univers, inspirée par [Hof97] pour la théorie générale et par [Uni13] pour la présentation des univers et du type identité.

La figure 3.1 de la section 3.1.1 définit les règles de typage de base de $\text{MLTT}_{\mathcal{U}}$, correspondant à la bonne formation des contextes $\mathbf{wf}(\Gamma)$, aux univers $\mathcal{U}$ (à la Russell) et aux produits dépendants $\Pi x : T.U$. L'égalité jugementale (aussi appelée conversion) est introduite ensuite. Dans la section 3.1.2, on introduit le type identité $M_1 =_T M_2$ dont les termes sont les "preuves" d'égalité de M_1 et M_2. Ces règles de typage se trouvent dans la figure 3.2. Ce type permet ensuite de formaliser la notion de d'"irrelevance" des preuves, introduite dans la section 3.1.3. Les sections 3.1.4,3.1.5,3.1.6 étendent $\text{MLTT}_{\mathcal{U}}$ avec de nouveaux constructeurs, les sommes dépendantes et les types sous-ensemble, le type vide et singleton, et les coproduits et les entiers naturels.

Enfin, la section 3.1.7 se concentre sur l'unicité des preuves de typage, en coerçant explicitement l'utilisation de la règle de conversion. On compare alors dans la section 3.1.8 les différences entre le système de types présenté et celui derrière l'assistant de preuve Coq, qui est utilisé pour implémenter les techniques du chapitre 4.

Langages en Appel par Valeur avec Références

On introduit ensuite dans la section 3.2 les langages de programmation qui sont les objets d'études de cette thèse. Il s'agit de fragments de ML, c'est à dire des langages fonctionnels typés avec une réduction en appel par valeur et des *références nominales*.

Dans sa forme la plus générale, le langage que l'on considère dispose de références d'ordre supérieur, c'est à dire qu'il peut stocker des fonctions dans les références. On appelle ce langage RefML. Sa syntaxe est introduite en section 3.2.1, avec notamment la définition de sa sémantique opérationnelle dans la figure 3.8. Les règles de typage de RefML sont introduite en section 3.2.2 dans la figure 3.9. L'équivalence contextuelle entre deux programmes, qui est la notion centrale de cette thèse, est alors introduite dans la sec-

tion 3.2.3. Finalement, GroundML, une restriction du langage où les références ne peuvent stocker que des valeurs de types de base (Int ou ref τ) est introduite en section 3.2.4.

Raisonnement Nominal

Lorsque l'on formalise des preuves sur le λ-calcul, on a souvent à raisonner à α-*équivalence* près. C'est à dire, deux termes qui ne diffèrent que par rapport au nom de leurs variables liées sont considérés comme étant équivalents. La même idée s'applique au locations, puisque l'arithmétique de pointeurs est interdite dans notre cadre. Pour formaliser cette situation, la *logique nominale* a été introduite par A. Pitts [Pit03]. L'idée de base est de voir les objets dont la définition utilise des variables ou des locations uniquement à permutation près de ces éléments. Cela peut être formalisé en utilisant la notion d'*ensemble de Fraenkel-Mostowski*, appelés *ensembles nominaux* dans la section 3.3.1, qui sont des ensembles équipés d'une action de permutation sur un ensemble de noms (qui sont l'ensemble des variables ou des locations ici).

On a parfois besoin de raisonner explicitement sur les permutations lorsqu'on considère deux éléments nominalement équivalents t, u d'un ensemble nominal X. Cependant, lorsque c'est le cas, il est plus aisé de travailler avec une notion de *span* qui sont introduits dans la section 3.3.2.

Chapitre 4: Forcing en Théorie des Types

Le forcing est une méthode conçu à l'origine par P. Cohen pour prouver l'indépendance de l'hypothèse du continu par rapport à la théorie des ensemble axiomatique (ZFC) [CD66]. L'intuition derrière cette méthode est d'ajouter de nouveaux objets à la théorie, qui peuvent être *approximés* dans le système de base, en utilisant ce qu'on appelle des *conditions de forcing*. Plus précisément, un modèle de base M est étendu vers un nouveau modèle M[G] en ajoutant un nouvel élément *générique* G à M. Alors que M[G] est en général très complexe à étudier directement, P. Cohen a proposé de contrôler les propositions vraies de M[G] en les traduisants vers M. Pour ce faire, il a donc utilisé les *conditions de forcing*, qui peuvent être vues comme des approximations de l'ensemble G. Ces conditions sont des éléments de M, ce qui n'est pas en général le cas de l'élément générique G. Ainsi, à partir d'une formule φ de M[G], on peut construire syntaxiquement une formule $p \Vdash \hat{\varphi}$—qui s'exprime comme "p *forces* $\hat{\varphi}$"—qui appartient à M, et qui est telle que φ est vraie lorsqu'il existe une approximation "correcte" p de G telle que $p \Vdash \hat{\varphi}$ dans M. Une des propriétés essentielles de l'ensemble des conditions de forcing est d'être ordonnée. Intuitivement, on a $p \leq q$ lorsque p est une approximation plus précise de G que q, c'est à dire qu'il contient plus d'information, de telle manière que la relation $\Vdash$ doit être monotone par rapport à cette relation d'ordre.

Ces dernières années, le forcing a reçu une attention accrue, dans le but d'étendre la correspondance de Curry-Howard à divers raisonnements classiques. Ainsi, le forcing a été généralisé par Krivine dans ses travaux sur la *réalisabilité classique* [Kri09], où les conditions de forcing deviennent des λ-termes. Il a ensuite combiné cette approche avec la technique usuelle de forcing dans [Kri11] pour donner un contenu calculatoire à des formes faibles de l'axiome du choix, tel l'existence d'un ultrafiltre sur $\mathbb{N}$ ou d'un bon ordre sur $\mathbb{R}$. À la suite de ces travaux, A. Miquel [Miq11] a étudiée la transformation de preuve induite par le forcing, dans la théorie de l'arithmétique d'ordre supérieur vu comme une extension de System F_ω. C'est à dire, il a étudié comme une preuve d'une proposition P est transformée en une preuve de $p \Vdash P$.

Dans un cadre constructif, nous avons utilisé le forcing, en collaboration avec T. Coquand [CJ10, CJ12], pour donner un contenu calculatoire à un résultat de continuité uniforme à certaines fonctionnelles définissables dans la théorie des types de Martin-Löf. Dans ces travaux, nous avons étendu MLTT avec un jugement de typage indexé par une condition de forcing. Pour cela, nous avons développé un traduction simple de MLTT, qui ne s'intègre bien qu'avec les termes que l'on souhaitait ajouter à notre théorie. Cela laisse ouvert la question d'une traduction sur l'ensemble des termes de MLTT.

Pour adapter les idées "ensemblistes" du forcing et les adapter à une théorie des types dépendante, nous avons utilisé sa reformulation catégorique que Lawvere et Tierney [Tie72] ont développée en utilisant les topos de faisceaux. Dans ce cadre, le forcing intuitionniste se définit à l'aide de la notion de préfaisceaux en théorie des catégories.

Le point de départ de ce chapitre est de connecter ces deux observations:

"Le forcing intuitionniste pour la théorie des types est une internalisation de la construction de préfaisceaux dans la théorie des types."

Plus précisément, on cherche donc à étendre une théorie des types initiale—que l'on appelle *système de base*—avec de nouveaux principes, pour obtenir une nouvelle théorie des types—que l'on appelle la *couche de forcing*. Termes et types de cette couche de forcing peuvent alors être traduits dans le système de base en utilisation la *traduction de forcing*. De cette manière, on peut développer une nouvelle génération de logiques, qui peuvent être définie de manière modulaire en utilisant des couches de forcing. La traduction de forcing s'appuie sur l'internalisation dans $\mathrm{MLTT}_{\mathcal{U}}$ de la construction de préfaisceaux sur un type particulier $\mathcal{P}$—représentant les *conditions de forcing*. Il devient alors possible d'exhiber de nouveaux principes de raisonnement dans la couche de forcing en utilisant la structure du type des conditions de forcing choisi. Mais, quel que soit les nouveaux principes logiques qui sont définis, leur consistance peut être déduite directement:

"La consistance de la logique définie dans une couche de forcing résulte de la consistance de la logique de base."

En effet, on est capable d'étendre la théorie des types avec de nouveaux principes de raisonnement et de nouveaux objets, sans les définir comme axiomes. Au delà des problèmes de consistance, éviter l'approche axiomatique nous permet de donner un contenu calculatoire à ces nouveaux principes: des programmes leurs sont associés.

Cette définition du forcing pour la théorie des types est liée aux travaux sur la sémantique de Kripke qu'Appel, Melliès, Richards et Vouillon [AMRV07] ont proposé pour voir le step-indexing— une technique pour gérer différentes formes de récursions dans la sémantique des langages de programmation—comme forcing (à la Kripke) sur l'ensemble des entiers naturels. Ces entiers peuvent être utilisés pour définir (ou forcer) une modalité de la logique, une idée auparavant introduite par Nakano in [Nak00], avec un principe d'induction obtenu directement des entiers naturels. Plus récemment, Birkedal, Møgelberg, Schwinghammer, et Støvring [BMSS11] ont montré que cette construction correspond sémantiquement à travailler à l'intérieur du topos des arbres, qui fournit une méthode générique pour définir des types récursifs généraux dans le modèle sémantique. De manière similaire, on utilise le forcing sur l'ensemble des entiers naturels pour fournir des types récursifs généraux dans $\mathrm{MLTT}_{\mathcal{U}}$, sans s'appuyer sur un critère de positivité. Cette construction permet de définir un type universel $\mathcal{D}$ pour les termes du λ-calcul pur qui induit un plongement superficiel du λ-calcul pur dans $\mathrm{MLTT}_{\mathcal{U}}$. Le fait que l'on peut utiliser la règle de conversion qui est normalisante pour décrire la β-reduction du λ-calcul pur n'est pas contradictoire. Le dépliage du type récursif $\mathcal{D}$ est géré par une égalité propositionnel et non pas par la règle de conversion, et doit donc être explicite dans le terme. Pour illustrer l'utilisation du forcing dans notre cadre, nous proposons également une reformulation de la construction de Cohen d'un modèle validant la négation de l'hypothèse du continu, en utilisant les sous-ensembles finis de $\mathbf{P}(\mathbf{P}(\mathbf{Nat})) \times \mathbf{Nat}$ comme ensemble de conditions de forcing.

Plan du Chapitre Dans la section 4.1, on explique les intuitions de la traduction de forcing à l'aide de la construction de préfaisceaux. On la définit ensuite formellement dans la section 4.2, en traitant les différents problèmes de cohérence posés par la règle de conversion. La section 4.3 présente une manière systématique d'introduire de nouveaux principes de raisonnement dans une couche de forcing. On illustre ensuite dans la section 4.4 cette traduction en choisissant comme ensemble de conditions de forcing les entiers naturels, pour fournir un cadre aux types récursifs gardés. Cela fournit une présentation syntaxique au *topos des arbres* [BMSS11]. Ensuite, en section 4.5, nous illustrons la traduction de forcing avec un autre ensemble de conditions de forcing, dans le le but de forcer la négation de l'hypothèse du continu. Finalement, les travaux reliés et de possibles extensions sont présentés en section 4.6.

Chapitre 5: Sémantique des Jeux Nominale Opérationnelle

Pour atteindre notre objectif de prouver la complétude de relations logiques à la Kripke, définies de manière directe, pour RefML, nous devons les relier à un modèle dénotationnel *pleinement abstrait*, pour lequel les dénotation de deux termes sont égales si et seulement si ils sont contextuellement équivalents. La recherche de longue date d'un tel modèle pleinement abstrait a abouti avec la *sémantique des jeux*.

La sémantique des jeux [HO00, AJM00] est une théorie puissante pour construire des modèles dénotationnel pleinement abstrait de différents langages de programmation. La dénotation d'un terme est représentée par une *strategie*, c'est à dire un ensemble de *parties* entre un terme et n'importe quel contexte, qui se jouent dans une *arène*, qui définie les règles que les parties doivent suivre. Une des contribution les plus importante de la sémantique des jeux, le fameux "cube d'Abramsky", est la caractérisation de l'absence de divers effets dans le langage en termes de conditions supplémentaires sur la dénotation des termes. Ainsi, le *bon parenthésage* correspond à l'absence d'opérateurs de contrôle, la *visibilité* à l'absence de références d'ordre supérieur, l'*innocence* aux termes purs. Ces dernières années, la sémantique des jeux a été étendue à des langages avec des aspects nominaux, du ν-calcul [AGM+04], puis à une extension avec cellules mémoires [Lai08], jusqu'à une variante de ML avec références nominales d'ordre supérieur [MT11b].

Le point de départ de cette section est la sémantique nominale des jeux de Murawski et Tzevelekos [MT11b], présentée dans la section 5.1, qui est pleinement abstraite pour RefML, mais aussi pour GroundML dès lors qu'on ajoute une condition de visibilité aux stratégies [MT12]. Au contraire des précédents modèles de jeux pour des langages avec mémoire, initiés par les travaux de Abramsky, Honda et McCusker [AHM98], ce modèle utilise des techniques nominales [Pit03], auparavant présente dans [Tze07], pour éviter le problème des "*bad variables*". Dans un cadre plus opérationnel, Laird [Lai07] a introduit une sémantique de trace pour une variante de RefML, et a prouvé sa pleine adéquation. Son modèle marie une représentation sous forme de traces inspirées de la sémantique des jeux avec une définition opérationnelle, c'est à dire que la dénotation des termes est calculée à l'aide d'un système de réécriture plutôt que définie par induction sur leur jugement de typage.

Dans cette section, nous introduisons une sémantique de trace pour RefML, dont la définition est une variante typée de celle introduite par Laird. Les traces sont générée par une *réduction interactive*, qui peut être vue comme une extension de la sémantique opérationnelle usuelle à des termes ouverts ayant des variables fonctionnelles libres. Ainsi, cette réduction peut être utilisée sur des termes comme $\lambda x.M$ or $K[f\ v]$. Ainsi, la dénotations des termes est définie via des *stratégies de traces*, c'est à dire des ensembles de traces que les termes génèrent en utilisant cette réduction.

La sémantique de traces permet, par rapport à la sémantique des jeux, une étude plus fine de l'interaction en suivant le processus de divulgation des valeurs (que ce soit des

locations ou des fonctions) entre le terme et les contextes. En effet, en sémantique des jeux, on ne peut en général décomposer les parties en "sous-parties", puisque l'on obtient alors une suite de coups où certains ne seraient plus justifiés, et ainsi laissés sans contrôle. Avec la sémantique de traces, de tels coups non justifiés sont simplement vue comme des valeurs divulguées, qui peuvent être prises en charge par la réduction interactive. En utilisant cette décomposition plus fine, on peut prouver un résultat de décidabilité de l'équivalence contextuelle pour des termes "purs" de RefML, c'est à dire des termes qui n'utilisent pas de possibilité de stockage dans le tas (mais avec des contextes non restreints). Ce résultat utilise de manière cruciale le contrôle des fonctions divulguées pour effectuée une "chirurgie" sur les traces. Dans le prochain chapitre, nous verrons comment effectuer cette chirurgie sur des termes "impurs", en définissant de nouvelles techniques pour raisonner sur la divulgation de locations.

Les traces peuvent en fait être vues comme une représentation des parties utilisée en sémantique des jeux, où la structure de pointeurs, qui représente la relation de causalité entre les différents coups, est encodé via des variables. De telles variables, qui sont de type fonctionnel, sont appelées *noms de pointers*.

Suivant cette idée, nous prouvons une correspondance entre cette dénotation des termes, définie comme ensemble de traces, et la dénotation issue de la sémantique des jeux, définie par des ensembles de parties. Pour ce faire, nous imposons sur les stratégies de traces une structure catégorique qui correspond aux langages en appel par valeur, c'est à dire une *catégorie Freyd-fermée* [PR97, PT99]. Elle consiste en:

— une catégorie symétrique prémonoidale $(\mathcal{C}, I, \otimes)$,
— une catégorie "luff" $\mathcal{C}'$ de $\mathcal{C}$, pour laquelle $\otimes$ est un produit cartésien,
— un foncteur prémonoidal strict $(.)^\dagger$ entre $\mathcal{C}'$ et $\mathcal{C}$, qui est l'idéntité sur les objets,

tel que pour tout objet A de $\mathcal{C}$, le foncteur $(\_ \otimes A)^\dagger : \mathcal{C}' \to \mathcal{C}$ dispose d'un adjoint à droite. Pour construire une telle structures sur les traces, on reformule les définitions de la sémantique des jeux de issues de [Lai08, MT11b] dans le cadre de la sémantique de traces. La difficulté principale vient de l'absence d'une définition explicite de la structure de pointeurs, qui doit être reconstruire à partir d'une étude de la fraicheur des noms de pointeurs (*i.e.* c'est à dire des variables fonctionnelles).

On relie ensuite la notion usuelle de vue de la sémantique des jeux à des ensembles de noms de pointeurs de joueur *disponibles*, de telle manière que l'on puisse importer la condition de visibilité qui capture le comportement des termes de GroundML directement sur les traces. En modifiant finalement la réduction interactive pour restreindre les questions d'Opposant à être effectuées sur de tels noms de pointeurs de joueur *disponibles*, on peut capturer l'équivalence contextuelle de GroundML avec l'équivalence de traces.

Plan du Chapitre Les notions de base de la sémantique des jeux (dans un cadre en appel par valeur) et sa structure catégorie sont introduite en section 5.1. La section 5.1.7 rappelle les principales caractéristiques de la *sémantique des jeux nominale*.

La notion de trace est définie formellement dans la section 5.2.1 et sa correspondance

avec les parties de la sémantique des jeux est esquissée dans la section 5.2.2. La réduction interactive est introduite en section 5.2.3 et l'interprétation des termes comme ensembles de traces est définie en section 5.2.4. Une des principales contributions de ce chapitre, présentée en section 5.3, est la comparaison de ce modèle de traces avec le modèle de jeux de Murawski et Tzevelekos [MT11b], et la preuve que les deux sont en fait équivalents. On répond également à une question posée par Laird à la fin de [Lai07] sur une possible sémantique de traces pour un langage avec références restreintes aux types de base, en reformulant la notion usuelle de *visibilité* dans le cadre de la sémantique de traces. Ceci est fait en section 6.7.2. Finalement, en section 5.5, on montre l'utilité de la sémantique de traces en prouvant que l'équivalence contextuelle de termes purs de RefML (et également de GroundML) est décidable. Cela donne un premier exemple des idées introduites dans les deux prochains chapitres pour raisonner sur l'équivalence contextuelle.

Chapitre 6: Relations Logiques Concrètes

Dans le chapitre 5, nous avons introduit la sémantique de traces, un modèle pleinement abstrait de RefML, comme variation du modèle standard de la sémantique des jeux. Cependant, ces deux modèles sont difficilement utilisable directement pour raisonner sur l'équivalence de termes de RefML. En effet, dans ces modèles la dénotations des termes—c'est à dire les stratégies—sont des objets infinis pour lesquels raisonner sur l'égalité est difficile.

D'autre part, il y a eu de très nombreux travaux pour développer des techniques et modèles opérationnels pour prouver l'équivalence de programmes écrits dans des langages avec références. Deux des plus importantes techniques sont les *bisimulations* et les *relations logiques*. Les bisimulations environnementales [SKS11, Sum09] étendent les bisimulations définies pour les langages du premier ordre (tel les calculs de processus) à des langages d'ordre supérieur. Elles sont pour cela définies comme des ensembles de relations, de telle manière à représenter l'évolution de l'information divulguée par le terme au contexte. La coinduction est ensuite utilisée pour raisonner sur l'équivalence résultante. Cette utilisation de la coinduction est particulièrement utile pour raisonner sur la récursion dans les termes, évitant ainsi l'utilisation du step-indexing.

Les relations logiques à la Kripke, qui ont été présentées dans l'introduction de cette thèse, sont une autre approche pour prouver l'équivalence contextuelle de deux termes. Dans un cadre en appel par valeur, elles sont définies par les relations $\mathcal{V}[\![\tau]\!]w$ et $\mathcal{E}[\![\tau]\!]w$ respectivement sur les valeurs et les termes, via une induction mutuelle sur le type τ. Elles utilisent une notion de monde w pour contraindre les tas utilisés pour réduire les termes M_1, M_2 vers des valeurs dans la définition de $\mathcal{E}[\![\tau]\!]w$. Dans un article récent [HDNV12], Hur et al. ont proposé de marier les relations logiques à la Kripke et les bisimulations environnementales pour définir les *bisimulations paramétriques*.

Cependant, tous ces travaux sont soit incomplet par rapport à l'équivalence observationnelle, où alors ils nécessitent une forme de clôture qui introduit une quantification (infinie) sur les contextes. C'est notamment le cas des relations logiques, qui utilisent une définition par biorthogonalité pour obtenir la complétude: $\mathcal{E}[\![\tau]\!] \stackrel{def}{=} \{(M_1, M_2) \mid \forall (K_1, K_2) \in \mathcal{K}[\![\tau]\!].(K_1[M_1], K_2[M_2]) \in \mathcal{O}$ et $\mathcal{K}[\![\tau]\!] \stackrel{def}{=} \{(v_1, v_2) \mid \forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!].(K_1[v_1], K_2[v_2]) \in \mathcal{O}\}$ où $\mathcal{O}$ est l'ensemble des couples de termes equi-divergents. On pourrait imaginer relier directement une définition direct à une définition par biorthogonalité d'une relation logique. Mais cela semble hors de porté pour un langage comme RefML. Plus précisément, considérons deux termes $(M_1, M_2) \in \mathcal{E}[\![\tau]\!]w$ où $\mathcal{E}[\![\tau]\!]w$ est défini par biorthogonalité, essayons alors de déplier cette définition vers un style direct: soit h_1, h_2 deux tas satisfaisant les contraintes induites par w, supposons qu'à la fois $(M_1, h_1) \mapsto (v_1, h_1')$ et $(M_2, h_2) \mapsto (v_2, h_2')$. Alors il est très dur de construire un monde w' future de w tel que $(v_1, v_2) \in \mathcal{E}[\![\tau]\!]w$ et h_1', h_2' satisfassent les contraintes induites par w'. On repose en effet sur le pouvoir discriminant des contextes dans $\mathcal{K}[\![\tau]\!]w$, mais en essayant cela, on arrive à étudier précisément l'interaction entre termes et contextes, ce qui nous amène à utiliser la sémantiques des

jeux et de traces.

Dans cette thèse, nous proposons la première preuve de complétude des relations logiques à la Kripke, définies sans aucune forme de clôture, pour des termes sans divergence de GroundML, avec des contextes dans RefML. Pour cela, nous nous appuyons de manière cruciale sur la sémantique de traces introduite dans le chapitre précédent. En pratique, nous raffinons ce modèle pour définir la sémantique de traces à la Kripke.

Pour aboutir à ce résultat, nous avons modifier la définition usuelle des relations logiques à la Kripke, pour la rapprocher de la sémantique de traces. Tout d'abord, nos relations logiques sont définies sur des termes ouverts, avec des variables libres fonctionnelles [5] (c'est à dire des noms de pointeur d'Opposant), qui sont reliés à l'aide d'un environnement relationnel e qui correspond à un span, comme introduit en section 3.3. Ainsi, nos relations logiques $\mathcal{E}\left[\!\left[\tau\right]\!\right]_e$ sont indexées par ce span, de telle manière que:

— deux λ-abstractions $\lambda x_1.M_1, \lambda x_2.M_2$ sont dans $\mathcal{V}\left[\!\left[\tau \to \sigma\right]\!\right]_e$, avec τ un type fonctionnel, lorsque $(M_1, M_2) \in \mathcal{E}\left[\!\left[\sigma\right]\!\right]_{e\cdot(x_1,x_2,\tau)}$,

— deux variables (fonctionnelles) (x_1, x_2) sont dans $\mathcal{V}\left[\!\left[\tau \to \sigma\right]\!\right]_e$ lorsqu'elles sont dans e.

Cela est particulièrement utile pour raisonner de manière abstraite sur les callbacks. On peut ensuite gérer globalement les contraintes sur les locations divulguées. En effet, considérons les deux termes suivants:

$$\begin{array}{rcl} M_1 & = & \texttt{let x = ref0 in } \lambda\texttt{f.fx;x := 1} \\ M_2 & = & \texttt{let x = ref0 in } \lambda\texttt{f.fx;x := 2} \end{array}$$

Ils ne sont pas équivalents car la location liée à x, divulguée via le callback `fx`, ne stocke pas la même valeur à la fin de l'exécution de M_1 et M_2. Habituellement, de tels invariants sur les locations divulguées sont imposées par le monde w (de la même manière que les invariants sur les locations privées) et la définition de $\mathcal{V}\left[\!\left[\text{ref}\,\tau\right]\!\right] w$. Ici, on choisit une approche différence, en imposant globalement ces invariants sur les valeurs stockées dans le tas.

Pour ce faire, on doit suivre l'évolution de la divulgation des locations. On utilise donc une notion de span $\mathcal{D}$ entre locations divulguées locations pour cela, suivant les travaux de Stark [Sta98]. Ainsi, les mondes contiennent un tel span $\mathcal{D}$, qui évolue au cours de l'execution des termes, en étant étendu avec les nouvelles locations divulguées. Par exemple, considérons maintenant les deux termes suivants:

$$\begin{array}{rcl} M_1 & = & \texttt{let x = ref0 in let y = ref0 in } \lambda\texttt{f.fx;fy} \\ M_2 & = & \texttt{let x = ref0 in } \lambda\texttt{f.fx;fx} \end{array}$$

Ils ne sont pas équivalents car M_1 divulgue deux locations différentes via deux callbacks à `f`, alors que M_2 divulgue la même location. Si l'on essaye de prouver cette équivalence en utilisant nos relations logiques, on se retrouve à construire un span qui contient à

5. À notre connaissance, les relations logiques sont toujours définies sur des termes clos. Cela aurait pu justifier un choix de nom différent pour notre technique.

la fois (l_x, l_x) et (l_x, l_y) (où l_x, l_y représentent les locations liées à x, y), ce qui n'est pas possible de part la définition des spans comme bijections partielles. Cette idée d'utiliser des spans pour relier les locations divulguées est déjà présent dans les travaux de Dreyer et al. [DNB10] mais quelque peu caché. En effet, dans leur approche, chaque état du système de transitions encode une bijection partielle entre les locations, qui peut alors être utilisée dans le même but. Cependant, cette bijection partielle y est utilisée localement dans la définition de $\mathcal{V}\,[\![\text{ref}\,\tau]\!]\,w$, plutôt que globalement dans la définition des contraintes déduites de w sur les tas, notée $(h_1, h_2) : w$.

Pour obtenir des résultats de model-checking pour nos relations logiques, qui sont introduites dans le chapitre 7, nous devons contraindre la notion de mondes futures induite par le système de transitions. Dans les travaux précédents [PS98, ADR09, DNB10], un monde futur w' de w peut être étendu avec de nouveaux invariants (ou de nouveaux systèmes de transitions d'invariants) sur la partie des tas qui est disjointes des tas décrits par w. Cela découle de l'idée que les mondes sont partitionnés en *îles*, chacune correspondant à une partie du tas disjointe des autres. Ici, nous interdisons cette possibilité. Ainsi, le système de transitions doit rester fixe dès le départ. Cela suppose que l'on doit prévoir la création des locations futures effectuée par le terme: on appelle de systèmes de transitions *omniscient*, qui sont donc entièrement défini dès le départ. Ainsi, on établit une distinction claire entre les mondes w, qui sont de simples invariants, et le système de transitions $\mathcal{A}$, qui contraint l'évolution de ces mondes. La définition de nos relations logiques $\mathcal{E}_{\mathcal{A}}\,[\![\tau]\!]\,w$ dépend alors à la fois du système de transitions $\mathcal{A}$, qui est fixe de manière globale dans la définition inductive des relations logiques, et le monde w qui évolue au sein de la définition.

Par ailleurs, nous avons choisi pour représenter l'évolution des mondes de travailler avec des *systèmes de transitions étiquetées* (*Labeled Transition Systems*, abrégé en LTS) plutôt que des systèmes de transitions d'états. Cela permet de garder les systèmes de transitions finis dans de nombreux exemples, ce qui sera explicité dans le chapitre 7. Ces systèmes de transitions sont principalement utilisés pour représenter les contraintes sur les tas a l'aide de couples de pair of pre- et post-conditions entre les tas pris en entrée et les tas obtenus en sortie.

Comme nous allons le voir dans la définition de l'équivalence $\Sigma; \Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$ induite par nos relations logiques, deux termes sont équivalents s'il existe un LTS $\mathcal{A}$ contraignant les mondes tel que le couple (M_1, M_2) est dans $\mathcal{E}_{\mathcal{A}}\,[\![\tau]\!]\,w$ pour w "initial" (c'est à dire pour lequel toutes les locations apparaissant dans M_1, M_2 sont divulguées). Dans le but de prouver la complétude, nous devons alors construire un tel LTS lorsque deux termes M_1, M_2 ont leur dénotation, issue de la sémantique de traces, égale.

Ainsi, nous définissons une notion plus fine, la *sémantique de traces à la Kripke*, qui utilise également les mondes pour contraindre les tas apparaissant dans les configurations de la réduction interactive. Cela est particulièrement utile pour contrôler la divulgation des locations, puisque la définition de $[\Sigma; \Gamma \vdash M : \tau]$ présuppose que toutes les locations sont divulgués initialement, ce qui n'est pas préservé par réduction. Cependant, cette

utilisation de mondes pour affiner la sémantique de traces est indépendante des LTS. Cela veut dire que nous devons caracteriser les mondes w et les LTSs $\mathcal{A}$ ayant les propriétés suffisantes pour s'assurer, à partir du fait que M_1, M_2 ont des dénotations de traces équivalentes pour le monde w, que $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!] w$. Nous appelons de tels mondes *adéquats*, dont la définition peut être vue, en première approximation, comme le dual des relations logiques. Plus précisément, nous définissons l'ensemble des mondes adéquats pour deux termes M_1, M_2 via un prédicat sur les mondes, noté $\bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!](M_1, M_2)$. Ce prédicat impose essentiellement l'existence de tous les mondes futurs de w nécessaires (par rapport à $\mathcal{A}$) pour décrire les évolutions possibles des tas que l'on obtient au cours de la réduction interactive de M_1, M_2. Ainsi, à partir de deux termes M_1, M_2, on construit un LTS $\mathcal{A}$, que l'on appelle *LTS exhaustif*, pour lequel le monde initial est effectivement dans $\bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!](M_1, M_2)$. Il est construit en utilisant l'ensemble des traces générées par la réduction interactive, en ajoutant un nouvel état pour chaque action de ces traces.

Notons finalement que l'on appelle ici nos relations logiques *concrètes* pour les distinguer des relations logiques *temporelle* qui sont introduite dans le prochain chapitre.

Plan du Chapitre Nous commençons par introduire dans la section 6.1 la notion de LTS et de mondes. Ensuite, les relations logiques concrètes pour le fragment formé des termes sans divergence de GroundML, considéré avec l'équivalence contextuelle de RefML, sont définies en section 6.2. Dans la section 6.3, nous introduisons la sémantique de traces à la Kripke, un raffinement du chapitre précédent où les mondes sont utilisés pour contraindre les tas dans la définition de la sémantique interactive. Cette notion est utilisée en section 6.4 pour prouver la la correction des relations logiques concrètes par rapport à la sémantique de traces.

Pour s'attaquer au problème de la complétude, nous introduisons la notion de *mondes adéquats* dans la section **??**, définie de manière duale aux relations logiques, pour contraindre la forme des LTSs à suivre le flot de contrôle des termes. Cette notion est centrale dans la preuve de complétude de la section 6.6, où on utilise un *LTS exhaustif*, défini en utilisant la réduction interactive, et qui est bien adéquat.

Dans la section 6.7.1, nous esquissons une définition possible des relations logiques concrètes pour l'ensemble de RefML, en utilisant une définition coinductive qui peut être formalisée en utilisant les types récursifs gardés. Il est intéressant de noté que, en travaillant avec une notion fixe de LTS, la circularité usuelle entre la définition des mondes et des relations logiques disparaît. Nous esquissons également dans la section 6.7.2 la définition des relations logiques concrètes pour GroundML, où la notion de *backtracking* entre états introduite dans [DNB10], pour relacher la définition des monde futurs, apparait.

Chapitre 7: Relations Logiques Temporelles

Disposant de relations logiques complètes définies de manière directe, on peut imaginer être maintenant capable de les utiliser pour prouver n'importe quelle équivalence. Cependant, en examinant avec attention la preuve de complétude, on peut se rendre compte qu'elle nécessite une notion de *LTS exhaustif* qui n'est clairement pas calculable.

Cependant, il est souvent possible de définir soi-même des mondes en inspectant la forme de mondes, cherchant ainsi à définir de judicieux invariants sur les tas. Cela ouvre la possibilité de *model-checker* l'appartenance de deux termes M_1, M_2 de type τ à la relation logique $\mathcal{E}[\![\tau]\!]$. Cela signifie que, une fois équipée d'un $\mathcal{A}$ representant l'évolution des invariants sur les tas pour M_1 et M_2, on pourrait *decider* si M_1, M_2 sont effectivement dans $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]\, w$, avec w le monde initial.

En effet, de nombreux choix suivis dans le chapitre 6 pour définir $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]\, w$ étaient faits dans ce but. Le plus important étant de travailler avec un LTS fixe $\mathcal{A}$. Cela ouvre effectivement la possibilité de travailler dans une logique simple dont la sémantique est définie par un modèle de Kripke. L'utilisation d'une telle logique équipée d'un modèle de Kripke a également été utilisé par Dreyer et al. dans [DNRB10], cependant, leur logique est plutôt complexe, à la fois car elle permet de quantifier directement sur les λ-termes, et car l'évolution des mondes y est peu contrainte.

Dans ce chapitre, nous travaillons plutôt dans une logique qui mélange l'*arithmétique de Presburger*, pour définir les contraintes sur les tas, et la *logique temporelle*, pour raisonner sur l'évolution des LTS, c'est à dure sur le flot de contrôle. La logique temporelle est d'une grande importance dans le domaine du model-checking, pour prouver des propriétés de sûreté et de vivacité, suivant les travaux précurseurs de A. Pnuely [Pnu77]. À notre connaissance, le cadre présenté dans ce chapitre est le le premier à utiliser la logique temporelle pour raisonner sur le flot de contrôle d'un langage fonctionnel impur, dans le but de prouver l'équivalence de programmes.

On définit ensuite les *relations logiques temporelle* dans cette logique. Pour ce faire, nous commençons par introduire une notion d'*exécution symbolique*, qui est utilisée pour réduire les termes ayant des variables libres de bases. En pratique, elle effectue simplement une réduction exhaustive, en générant des prédicats qui contraignent les valeurs de valeurs de bases, de telle manière que ces prédicats soient satisfiables tous ensembles si et seulement la réduction est effectivement possible opérationnellement. Grâce à elle, on peut définir les relations logiques temporelles, et établir une correspondance exacte avec les relations logiques concrètes du chapitre précédent.

En toute généralité, le problème du model-checking reste indécidable pour notre logique, et ce pour deux raisons:

— la présence de quantifications sur les tas, qui peuvent être vus comme des listes de taille non bornée,

— le mélange entre arithmétique de Presburger et logique temporelle.

Cependant, sous des hypothèses réalistes sur l'arité des fonctions de transitions, qui per-

mettent de contrôler la quantification sur les tas et les locations, et en supposant que la cloture transitive du LTS est fournie et s'exprime dans l'arithmétique de Presburger, on obtient un model-checking décidable.

Ce travail a donné lieu à une implémentation réalisée dans le langage Haskell. Ce prototype calcule la relation logique temporelle $\mathbb{E}[\![\tau]\!](M_1, M_2)$ associée à deux termes, mais est également capable, dans certains cas, de générer un LTS qui potentiellement valide cette formule. Il effectue ensuite un traduction de la validité de la formule temporelle par rapport à ce LTS vers SMT-LIB2, le langage commun aux SMT-solveurs. En utilisant le solveur Z3 (`http://z3.codeplex.com/`), nous avons ainsi été capable de décider de nombreux exemples issus de la littérature. Nous laissons ouvert la possibilité de résultats théoriques sur la génération de LTS. Si nous réussissons à prouver que l'implémentation génère un LTS qui, sous certaines hypothèses sur les termes, est *adéquat* (comme défini en section 6.5), cela fournirait des résultats de décidabilité pour des fragments de RefML et GroundML.

De tels résultats de décidabilité ont auparavant été obtenus en *sémantique des jeux algorithmique*, qui se base sur les résultats de pleine adéquation des modèles de jeux. En théorie n'importe quelle équivalence de programme peut être prouvée en regardant simplement l'égalité des stratégies. Cependant, raisonner sur l'égalité des stratégies dans le cadre fonctionnel est non trivial (voir même indécidable[Loa01]) car le modèle est quotienté par une relation complexe. Pour un langage avec références, il existe une bonne caractérisation du quotient, en utilisant les stratégies *complètes*, comme nous l'avons vu au chapitre 5.

La sémantique des jeux algorithmique se base sur ce résultat, et utilise la théorie des automates pour représenter les stratégies. Dans un cadre en appel par valeur, un classification complète de la décidabilité de l'équivalence de programmes a été réalisée par Ghica and McCusker [GM00] pour le langage Idealized Algol du premier order. En appel par valeur, une étude récente d'un fragment de ML avec "bad variables" a été développé en utilisant une variante des automates à piles [HMO11]. Et en utilisant des *automates à registres frais* [Tze11], cette méthode a été étendue aux "bonnes" references [MT11a, MT12]. La principale difficulté à laquelle la sémantique des jeux algorithmique fait face est l'encodage de la structure de pointeurs, dont les parties disposent, vers un langage reconnaissable par des automates. Pour ce faire, le type des termes considérés doit être restreint pour obtenir des résultats de décidabilité. Il serait intéressant de comparer ces résultats à ceux obtenus avec notre méthode, qui nécessite elle un contrôle sur la création des locations plutôt qu'une restriction sur les types.

Plan du Chapitre Nous commençons par définir dans la section 7.1 notre logique HeapTL, qui marie l'arithmétique de Presburger et la logique temporelle. Elle utilise les LTS introduits dans le chapitre 6 pour définir la sémantique des modalités temporelles, et les mondes pour la sémantique des prédicats sur le tas. L'exécution symbolique est introduite en section 7.2, qui joue un rôle crucial dans la possibilité de générer automatiquement

les relations logiques temporelles, introduites dans la section 7.3. On étudie ensuite dans la section 7.5 les propriétés de model-checking de ces relations logiques temporelles, qui ont été esquissées dans l'introduction. Ces résultats reposent sur la correction et la complétude des relations logiques temporelle (par rapport aux relations logiques concrètes), qui sont prouvés respectivement dans les sections 7.6 et 7.7.

2

Introduction

While computer programs play an ever broader role in critical systems, from airplanes to medical equipments, the need for formal methods to ensure their correctness becomes unavoidable. Among all formal methods, mathematical proofs of correctness of programs give the highest level of trust we can hope for. But such proofs can be really complex, requiring an automatic check to be performed. This can be done using *proof assistants*, which are themselves computer programs designed to automatically check if a proof of a theorem is correct. But, to reason on programs, it is necessary to first give them a precise meaning, which is achieved by defining the semantics of the programming language in which they are written. In this thesis, both the language of the proof assistant and the language of the programs to be analyzed come from the same central notion: the λ-calculus.

2.1 Functions and Types

The λ-calculus has been introduced by Alonzo Church as a computational model in which "everything is a function", where by functions we mean a syntactical object where the binding of a variable is handled explicitly. Application is then handled by *substitution* of variables. Such functions, called λ-abstractions, can be annotated with types to abstract over their computation. More precisely, a function is given a type $A \to B$ to say that it takes as argument elements of type A and returns elements of type B. This is the so called *simply-typed λ-calculus*. This language has taken a crucial importance in the field of both proof theory and programming languages.

Indeed, in proof theory, this language is used to represent proofs and formulas via the

famous *Curry-Howard correspondence* . The idea is that a term of type $A \to B$ represents a proof of the implication $A \Rightarrow B$. Thus, checking the correctness of a proof amounts to typechecking the corresponding term. Of particular importance is the work of P. Martin-Löf, who has extended this correspondence to predicate logic. His idea was to represent a universally quantified formula $\forall x \in A.P(x)$ as a *dependent type* $\Pi x : A.P$. To define such dependent types, we must allow terms to appear in types. For example, one can define a type $\mathcal{M}_{m,n}$ of rectangular matrix of size $m * n$, such that the multiplication of matrices would be of type $\mathcal{M}_{m,l} \to \mathcal{M}_{l,n} \to \mathcal{M}_{m,n}$. Here, we used numbers l, m, n (which are terms) to refine the type of matrices.

This system, called *Martin-Löf Type Theory*, is the basis of proof assistants such as Coq or Agda. These systems satisfy the so called *De Bruijn criterion*, *i.e.* they generate a witness of the proof, namely a λ-term, which can be checked by others means, so that we do not blindly rely on the proof assistant.

λ-calculus also had been at the origin of *functional languages*. The idea was to come back to the original concept of λ-calculus as a computational model, in order to derive a programming language. This was done firstly in its untyped version by J. McCarthy with LISP and later *typed functional languages* were conceived, like ML or Haskell. Types are particularly useful to specify programs, following the well-known slogan of R. Milner:

"*Well-typed programs can't go wrong.*"

This is the so called *type-safety* property. It can be proven in two different ways, either purely syntactically via the "progress and preservation" method, or by associating to each types τ a set of terms $[\![\tau]\!]$ which are "safe", the so called *semantic types* (or *realizability*) approach.

The λ-calculus will thus be used to build the objects of study of this thesis—functional programs—but also to represent proofs on these objects, via the Curry-Howard correspondence, using a rich (dependent) type theory. However, while the programs we consider are *impure*—in that they can perform side-effects like divergence or use a persistent memory—the proofs are represented by *pure* λ-terms which are among others terminating [1].

As previoulsy explained, a way to specify a functional program is via its type. But the type system must be rich enough if we want to be able to give a precise specification. For instance, this can be achieved by using refinement types [FP91] or even dependent type [NMB08].

Here, we argue that in many cases, the best language to write the specification of a program is its programming language itself. Indeed, when writing a program, it is often the case that we first design a simple version that we trust, then we optimize it. In this case, proving that the simple version is equivalent to the optimized one would give a correctness result for the optimized version.

Of course, for more complex programs, this alternative is not always possible. Particularly interesting examples are compilers. Their correctness is crucial, since they produce

1. By terminating, we mean that there are no terms which can be infinitly reduced.

low-level code that will be actually run on computers. Indeed, a bug in a compiler could unravel all the efforts done to prove that a program satisfies its specification, by generating incorrect low-level code. This is not a figment of the imagination, as shown in [YCER11], where dozens of bugs have been found in GCC or LLVM. There have been many works on formalization of correctness of compilers. One of the most impressive is the work of X. Leroy et al. with CompCert [Ler09]. In this formalization, the specification of the compiler corresponds to a simulation between the semantics of the high and the low-level language. However, one can imagine more "modular" specifications for compilers, which allow the interaction of low-level codes produced by two different compilers, or even hand-written. Following ideas from N. Benton et al. [BT09, BH09], a possible specification satisfying this modular property can be the conservation of the equivalence of high-level programs into low-level code. This is particularly interesting in order to prove correctness of optimization performed by compilers. While working on this idea [JT10, JT11], it appeared clear to us that we need a good logical framework to formalize proof of equivalence of programs. This was the starting point of this thesis.

2.2 Contextual Equivalence

We have chosen to focus on the notion of *contextual equivalence*, also called *observational equivalence*, for impure functional languages. In this setting, two programs are said to be equivalent if no context can distinguish them. Besides its practical interest presented before, this notion is fundamental in denotational semantics, where the goal is to build *fully-abstract* models. In such models, the denotation of two terms are equal if and only if the two terms are contextually equivalent.

As one can imagine, the expressive power of contexts, seen as programs with "holes", plays a major role in the study of contextual equivalence. In the degenerate case where contexts can inspect the source code of programs, as with low-level programming languages or assembly code, this notion of contextual equivalence completely collapses to syntactical equality: a program can only be equivalent to itself.

In the opposite direction, when contexts are suppose to be "pure", the study of contextual equivalence become quickly undecidable (even without any kind of recursion or infinite types), as shown by Loader in [Loa01]. In a way, this explains why the quest for a fully-abstract denotational model for PCF [2] was a lot harder than for PCF with some side effects.

Here, we are interested in contexts and programs which have a notion of mutable memory. More precisely, we consider the notion of *references*, which are the standard way to implement memory cells in ML-derived languages such as OCaml. A reference is defined via the constructor ref v, to store the value v in the heap (whereas the arguments given to functions are, as usual, stored in the stack). These references are then reduced

2. a simply-type λ-calculus with natural numbers and recursion

to locations, which can be seen as memory addresses. However, compared to low-level languages like C, one cannot perform any pointer arithmetic on locations: they have to be seen as abstract names for memory cells. Values stored in references can be of a higher-order kind, *i.e.* closures. This allows the performance of recursion.

Using references as a global –persistent– memory, a function can track how many times it is called. Hence, when calling it two times with the same argument, it can possibly return two different results. But contexts can also use references, so that two terms are equivalent only if they perform calls of functions, provided by contexts, in an equivalent way, and with equivalent arguments. This is the idea of *synchronization of callbacks*, which is false when contexts are pure.

2.3 Kripke Logical Relations

In general, contextual equivalence is hard to reason on due to the quantification over all contexts. Some tools have been introduced to avoid this quantification and so to prove contextual equivalence in a simpler way. One of the most used frameworks is that of logical relations, which are of particular importance to define the notion of parametricity following the seminal work of Reynolds [Rey83]. Logical Relations are a generalization of semantic types, used to prove type safety, to a relational (as opposed to predicate) setting. They are binary relations on values $\mathcal{V}[\![\tau]\!]$ and on terms $\mathcal{E}[\![\tau]\!]$ of type τ. The relation $\mathcal{V}[\![\tau]\!]$ is defined by induction over τ. On ground types, it simply imposes equality of value. On functional types $\tau \to \sigma$, two values $\lambda x_1.M_1, \lambda x_2.M_2$ are in $\mathcal{V}[\![\tau \to \sigma]\!]$ when :

$$\forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!], ((\lambda x_1.M_1)v_1, (\lambda x_2.M_2)v_2) \in \mathcal{E}[\![\sigma]\!]$$

Then, two terms are in $\mathcal{E}[\![\tau]\!]$ when they reduce to two values which are in $\mathcal{V}[\![\tau]\!]$. Using these definitions, one can show that two terms of type τ are contextually equivalent when they are in $\mathcal{E}[\![\tau]\!]$.

However, one cannot extend this definition directly to a language with references. Indeed, in this case, the reduction of a term M depends on the values stored by its references, *i.e.* in the heap. Thus, A. Pitts and I. Stark [PS98] have proposed to parametrized the definition of $\mathcal{E}[\![\tau]\!]$ and $\mathcal{V}[\![\tau]\!]$ by invariants on heaps. Such invariants are called *worlds*, following the intuition of Kripke models. The notion of worlds has later been refined by A. Ahmed [ADR09], and then by D. Dreyer et al. [DNB12] to state transition systems of heap invariants, to be able to prove equivalences which was out of reach of the definitions of Pitts and Stark.

2.4 The Step-Indexing Technique

Kripke Logical Relations given by Pitts and Stark were defined for a language with "ground references", that is references which can store ground values but not functions.

In her PhD thesis [Ahm04], A. Ahmed extends these definitions to recursive types and higher-order references. To do so, she had to deal with a circularity problem between the definition of worlds and logical relations. Appel and McAllester [AM01] already ran into this kind of circularity issue while working on type safety for low-level code with recursive types. Semantic types TypeSem, which are sets of "well-behaved" values, depend on the definition of worlds World which are semantic invariants on what locations can store. Written as recursive equations, this gives

$$\text{World} \stackrel{def}{=} \text{Loc} \to \text{TypeSem} \text{ and } \text{TypeSem} \stackrel{def}{=} \text{World} \to \text{Value}$$

Then, unwinding the definition of TypeSem, we get that it is equal to (Loc $\to$ TypeSem) $\to$ Value. So we would like to define TypeSem as a fixpoint $\mu X.(\text{Loc} \to X) \to \text{Value}$. However, X appears in a negative occurence in this definition, so in general such fixpoints does not exists. Indeed, one cannot apply the *Knaster-Tarski* theorem.

This kind of recursive equation can be solved while switching from set-definition to categorical definition. This is what is usually done while working in denotational semantics [SS71]. More precisely, in *Domain theory* [Sco82], we work in the category of *complete partial orders* where such equations can be solved. One of the most famous one being $D = D \to D$ for pure λ-calculus.

But in this thesis we want to reason *operationally* on programs, while denotational semantics interprets programs as mathematical objects. So staying in the operational world, Appel and McAllester[AM01] have introduced step-indexes to solve this circularity issue. The idea is to stratify the definition of semantic types with a natural number representing roughly the number of steps for which the programs in question behave properly. Thus, the definition of TypeSem and World is now indexed by an integer n, with the following definition :

$$\text{World}_{n+1} \stackrel{def}{=} \text{Loc} \to \text{TypeSem}_n \text{ and } \text{TypeSem}_n \stackrel{def}{=} \text{World}_n \to \text{Value}_n$$

This was then used by Ahmed to define logical relations first for languages with recursive types [Ahm06] then for languages with higher-order references, in a work in collaboration with Dreyer and Rossberg [ADR09]. In these works, step-indexes used in the definition of logical relations represent the number of steps for which the two programs in question are equivalent. In this way, it becomes possible to prove the equivalence of many programs that make use of recursive features by performing a simple induction on step-indexes.

But the management of step indexes during a proof appears to be—borrowing a word from Benton—"ugly". In their work on a "very modal model" [AMRV07], Appel, Melliès, Richard and Vouillon have proposed to reason abstractly on step-indexing using Gödel-Löb logic. To do so, they introduce a modality "later", written $\triangleright$, whose meaning (defined via a *Kripke semantics*) is roughly "true in the future". Then, the so called Löb rule $(\triangleright P \to P) \to P$, which can be seen as an induction principle, is valid in this logic. It is

then used to reason smoothly on recursive programs, as it would be done with explicit step-indexes.

This modality was in fact already used by Nakano [Nak00, Nak01] to define a λ-calculus with general recursive types which is strongly normalizing. To enforce this normalization property, this modality $\triangleright$ (which is written $\bullet$ in these works) is used to "guard" recursive types.

2.5 Forcing Translation

While working on step-indexing and its rephrasing in terms of Gödel-Löb logic, it appeared clear to us that this was a special instance of *forcing*. By forcing, we mean a general method, originally introduce by P. Cohen [CD66], to prove the independence of the continuum hypothesis. This technique has later been restated in many different forms. The most interesting presentation for our purpose is in terms of *topos of sheaves* or *presheaves* done by Lawvere and Tierney [Tie72].

This intuition was then confirmed by the work of Birkedal et al. on the *topos of tree* [BMSS11], where they have built a model of a type theory with *guarded recursive types* in terms of presheaves over natural numbers. The way to define Kripke Logical Relations as Logical Relations indexed by a world is also, as we can expect, an example of forcing, via the so called Kripke semantics of modal logic. This idea was explored by Dreyer et al. in [DNRB10], where they define a modal logic to reason on Kripke Logical Relations.

Our goal was then to be able to extend a type theory modularly—more precisely the type theory behind the Coq proof assistant—using this kind of forcing constructions. This has led us to build a general method to define a forcing translation of the whole type theory using an internalization of the presheaf construction, parametrized by a set of *forcing conditions*. Such a construction allows us to define *forcing layers*, that is an extension of the ground theory with new logical principle, while keeping the following important properties:

— the consistency of the theory,
— the decidability of the type-checking,
— the canonicity property, *i.e.* the fact that closed terms reduce to closed values.

Such conservation results are not in general available from semantics methods.

Instantiating the set of forcing conditions, we are then able to build what we call the *step-indexed layer*, where the construction of guarded recursive types from [BMSS11] can be defined. Working with a type theory with universes, we are additionally able to define a fixpoint operator on functions over types, which allows an internal way of defining guarded recursive types.

2.6 Biorthogonal versus Direct Definitions

An other main technique to define logical relations is biorthogonality, or $\top\top$-closure. Biorthogonality has a long history in proof theory. Indeed, it has been used by J.-Y. Girard to prove cut-elimination of linear logic, and by J.-L. Krivine in his work on classical realizability, to extend the Curry-Howard correspondence to classical logic with the axiom of choice.

It is used with logical relations to define $\mathcal{E}[\![\tau]\!]$ not directly in terms of $\mathcal{V}[\![\tau]\!]$, but rather as an indirect closure of $\mathcal{V}[\![\tau]\!]$ via an auxilliary relation $\mathcal{K}[\![\tau]\!]$ on contexts.

This definition as a closure is used for three different purposes:

— to reason on languages with context-dependent control-flow features, like the `call/cc` or `jump` instructions in assembly code,

— to build operational models of both polymorphism and recursion [Pit00], enforcing the *admissibility* of logical relations,

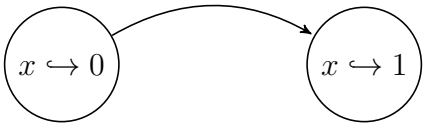
— to get completeness of logical relations, *i.e.* two terms of type τ which are contextually equivalent are in $\mathcal{E}[\![\tau]\!]$.

In this thesis, we do not consider control operators like `call/cc`, so that control flow is always sequential. To deal with recursive features of the language, we have seen that step-indexing is in practice enough, so we do not need any admissible condition. So the only remaining reason to use biorthogonality would be to get completeness of logical relations with respect to contextual equivalence.

However, when working with biorthogonal logical relations, we rely in fact on a *principle of local invariance* [PS98] which amounts to unwinding the biorthogonality to a direct definition. In fact, the completeness result obtained using biorthogonality is a bit spurious, since it does not mean that we are able to prove in practice any equivalence using these logical relations. This is exemplified by the so called "awkward example" :

$$\begin{aligned} M_1^{awk} &= \texttt{let x} = \text{ref}\,\texttt{0 in}\ \lambda\texttt{f.x} := \texttt{1}; \texttt{f}(); !x \\ M_2^{awk} &= \lambda\texttt{f.f}(); 1 \end{aligned}$$

To prove that these two terms are equivalent, the state invariant which constrains x to point either to 0 or to 1 is not enough to deduce that the first program returns 1. Thus, the model of Pitts and Stark, which uses worlds as invariants, cannot prove this equivalence, which can seem strange at first sight since it is complete. This is because using the biorthogonality closure to prove equivalence is as hard as reasoning directly on contextual equivalence, when one cannot use the principle of local invariance. Actually, the impossibility to prove this equivalence with the definitions of Pitts and Stark led Ahmed to refine their model, to allow heaps invariants to evolve during the execution [ADR09]. Using the subsequent work of Dreyer et al. [DNB12], the world used to prove this equivalence is the following state transition system:



which states that once x points to 1, it will never points to 0 again. Using this refined principle, it is then possible to show that M_1 is equivalent to M_2. Such STSs can in fact be seen as an abstraction of the control flow of the program, only taking into account the modification of the heap.

2.7 Operational Nominal Game Semantics

Without biorthogonality, proving completeness of logical relations for a language with references is an open problem. Indeed, one has to reason on the shape of contexts, which is a priori very hard. To tackle this problem, we choose to link logical relation to a denotational model of our language. The obvious choice is game semantics, and more precisely the model developed by A. Murawski and N. Tzevelekos for "good general references" [MT11b] [3] which is indeed fully-abstract for RefML. This model uses *nominal sets* [Pit13] to abstract reasoning on "names" of locations. This idea will also be used throughout this thesis.

This game semantics model does not however fit well with logical relations, as we can see with the following term M:

$$\lambda\, \texttt{f.let x} \;=\; \texttt{ref0 in f}(\lambda\_.\texttt{x} := \texttt{1})\ \texttt{in}\ M_1$$

This term keeps the reference x private, and gives to the context the possibility to change it to 1 via a setter function. But then, it is not possible to relate M_1 to M for two reasons :

— x is private to the term, but the denotation of M_1 is defined as if all its locations were disclosed to the context,

— the context has access to the setter $\lambda\_.\texttt{x} := \texttt{1}$ when M_1 is evaluated, but there is no easy way to specify this with game semantics.

To solve these limitations of game semantics, we need to define a new model, following previous work of J. Laird [Lai07]. It shares the good denotational (*i.e.* categorical) properties of game semantics. with operational ones. The idea is to interpret terms as sets of traces, which represent interactions between a term and any contexts, as in game semantics, but which can be computed operationally via an *interactive reduction*.

Then, refining this model with worlds to constrain the shape of heaps, we define a notion of *Kripke trace semantics*. This allows us to make a formal link with Kripke logical relations and therefore to prove their soundness and completeness.

3. "good" as opposed to the so called "bad-variables" which arise from the "object"-representation of references inspired by the work on Idealized Algol by Reynolds.

2.8 A Temporal Logic to Reason on Equivalence of Programs

To reason on correctness of programs, some logics have been designed, which provide general principles to prove such correctness results. The most famous one is *Hoare Logic*, which has then be refined with *separation logic* [Rey02].

Here, we are interested in program logics to reason directly on the equivalence of programs. Plotkin and Abadi logic for parametric polymorphism [PA93] was the first logic introduced to this end, for System F. Then, Dreyer et al. introduced LSLR [DAB09] and LADR [DNRB10], two logics for richer languages with recursive types and higher-order references. However, all of these logics used λ-calculus as built-in object, and then defined logical relations inside it. If they are well-suited to formalize proof of equivalence, it seems hard to use them to automatically prove equivalence of programs.

In this thesis, we follow a different approach, by defining a simpler logic with good model-checking properties to abstract reasoning on worlds used in our definition of Kripke logical relations. However, compared to step-indexing, this abstraction does not fit well within the framework of the presheaf translation, because some of the constructions used to define Kripke logical relations are not monotonic *w.r.t.* worlds.

So we define rather a *temporal logic* to reason on the transition system of worlds. The meaning of the various connectives of this temporal logic are defined via a usual Kripke modal semantics. Compared to [DNB12], we switch from state transition systems to labeled transition systems, where invariants are specified on transitions using a notion of pre- and post-condition in the spirit of Hoare logic. This is particularly useful to keep transition systems finite.

In previous works [PS98, ADR09, DNB10], a future world w' of w could have been extended with new invariants (or new STSs of invariants) on the part of heaps which are disjoint from the one w was specifying. This follows the idea that worlds were partitioned into *islands*, each one corresponding to a part of the heap disjoint from the others. However, this gives too much freedom to be definable in our restricted logic, where we can quantify only in simple objects. So here, we forbid this usage: the transition system stays fixed from the beginning, which does not change while worlds are evolving. This means that it must have foreseen the creation of future locations made by the term: we call such behavior *omniscient*.

To reason abstractly on free ground variables, we introduce a *symbolic execution* for our language which generates all the possible reductions of terms. It also generates a set of predicates which constrain values of ground variables, such that this set of predicates is satisfiable if and only if the reduction is indeed possible. Then, we define *temporal logical relations* $\mathbb{E}[\![\tau]\!]$, which are formulas of our temporal logic. From two terms M_1, M_2, $\mathbb{E}[\![\tau]\!](M_1, M_2)$ can be generated automatically using the symbolic execution. One then shows that there exists a world which validate this formula if and only if the two terms are in the usual, "concrete" logical relations. From the soundness and completeness of

the concrete logical relation, we reduce the problem of the contextual equivalence of two terms M_1, M_2 to the model checking of $\mathbb{E}[\![\tau]\!](M_1, M_2)$, that is:

M_1, M_2 are contextually equivalent if and only if there exists a world w such that
$$w \models \mathbb{E}[\![\tau]\!](M_1, M_2)$$

Our temporal logic mixes Presburger arithmetic, used to write arithmetic constrains on stored values, with temporal modalities inspired by the *Computational Tree Logic* to control the evolution of these constrains. Therefore It does not have in general the good properties of decidability of model checking that temporal logic usually has, due to the interaction with arithmetic.

However, under some reasonable hypothesis on the shape of worlds, we get the decidability of the model checking, that is we can automatically check if a world validates a temporal formula. This is done via translating the formula to a simpler one, which does not contain any temporal modality, which can then be checked by a SMT solver.

It seems possible, under a bounded heap hypothesis, to build automatically a world which validates the temporal formula if and only if the two programs are indeed contextually equivalent. This would give a decidability result [4] for the contextual equivalence for terms satisfying this hypothesis. We believe that this would encompass the decidability fragment of algorithmic game semantics [MT11a], which is characterize by a restriction on types.

Following these theoretical results, a prototype has been implemented in Haskell which:
— generate the temporal formula corresponding to the equivalence of two terms,
— translate this formula to a simpler one that a SMT-solver can validate, once a world is provided
— under a bounded heap hypothesis, generates a world which validates the formula if and only if the two terms are equivalent.

2.9 Overwiew of the Manuscript

In Chapter 3, we present the basic notions used throughout this thesis. We begin in section 3.1 by defining *Martin-Löf Type Theory* with a *hierarchy of universes*. Section 3.1.2 introduces *identity types* following ideas from *Homotopy Type Theory*. Section 3.1.7 focus on how to get uniqueness of proofs of typing judgments, by explicitly coercing uses of the conversion rule. Section 3.1.8 presents the difference between the type system we have presented and the one behind the proof assistant Coq, which has been used to implement techniques from Chapter 4. Then, section 3.2 presents RefML, a call-by-value language with higher-order references, which is our object of study. This Chapter ends with section 3.3 which introduces *nominal logic*, that we use to study RefML, more precisely to reason abstractly on names of variables and locations.

4. As proven in [Mur05, MT12], contextual equivalence is in general undecidable for a language with references, even in absence of recursion or infinite types

Chapter 4 introduces a new modular way to extend type theory using ideas from *forcing*, while keeping the decidability of type checking and the consistency of the theory. It uses a *presheaf translation* of type theory presented in section 4.1, whose formal study is performed in section 4.2. The notion of *forcing layer*, where the theory can be extended with new principles, is defined in section 4.3 Then section 4.4 introduces guarded recursive types, which are built in the *forcing layer* using *step-indexing*.

Chapter 5 defines *operational nominal game semantics*, a semantics which marries the good denotational properties of *game semantics* with an operational representation of terms using *traces*, to be able to perform modular reasoning. In section 5.1, we briefly recall notions of game semantics for RefML. Then, in section 5.2 we introduce trace semantics, firstly with definitions imported from game semantics, then by introducing the interactive reduction which allows us to generate traces from configurations. Section 5.3 provides a formal correspondence between trace semantics and game semantics, and show that the two models gives the same denotation to terms of RefML. We show in section 5.4 that a tweaked version of the interactive reduction, with an explicit management of scopes, captures exactly GroundML. This proof is again done via a correspondence with the game model of RefML, *i.e.* visibility. Finally, we exemplify in section 5.5 the use of trace semantics for operational reasoning by studying the equivalence of pure terms of RefML (with contexts in RefML).

We introduce concrete logical relations in Chapter 6. They are parametrized over a notion of worlds, whose evolution is governed by a notion of Labeled Transition Systems, introduced in section 6.1. The definition of concrete logical relations is given in section 6.2. To relate these concrete logical relations with trace semantics, we also parametrized traces with world, via a notion of Kripke trace semantics defined in section 6.3. This notion is crucial in section 6.4 to prove soundness of concrete logical relations. To prove completeness we first introduce in section 6.5 a notion of adequate worlds, which are worlds which respect the callback structure. Then, the proof of completeness is done in section 6.6. Finally, we briefly sketch in section 6.7.1 how to extend this work to full RefML, *i.e.* with higher-order references, and in section 6.7.2 how to restrict it to GroundML.

Finally in Chapter 7 we introduce a temporal logic in section 7.1 to build *temporal logical relations*. Its definition relies on a symbolic execution for RefML, defined in section 7.2. The definition of temporal logical relations is given in section 7.3, and we present some examples in section 7.4. Its model-checking properties are studied in section 7.5. We prove the soundness and the completeness of these relations with respect to concrete logical relations respectively in section 7.6 and 7.7.

3

Preliminaries

In this chapter, we introduce the technical background used throughout this thesis. Firstly, we introduce the type theory used as a metatheory to represent our proofs, namely Martin-Löf Type Theory. Then, we present RefML, a call-by-value language with higher-order references. This is the object of studies of this thesis. We also define a restriction

of it, namely GroundML, where references can only store ground values, i.e. natural numbers, booleans or locations. Finally, we introduce the basic ideas of *Nominal Logic*, to abstract the reasoning on locations and variables of RefML.

3.1 Martin-Löf Type Theory

Following the idea of the Curry-Howard correspondence, proofs can be represented as terms of λ-calculus, and formulas as types. This idea was pursue by Martin-Löf to give a constructive formalism to mathematics. As we have explained in the introduction, he has introduced dependent types to be able to represent existential and universal quantifiers.

This section introduces *Martin-Löf Type Theory*. Our presentation is highly inspired by [Hof97] for the general theory and [Uni13] for the presentation of universes and identity types.

3.1.1 Dependent Type Theory

The fragment of $\mathrm{MLTT}_{\mathcal{U}}$ restricted to dependent products is given by the following grammar

$$T, U, M, N \quad := \quad \mathcal{U}_i \mid \Pi x :: .T.U \mid x \mid \lambda x : T.M \mid \mathbf{App}_{(x:T)U}(M, N)$$

with $i \in \mathbb{N}, x \in \mathrm{Var}$ where Var is a fixed (infinite) set of variables. This set of terms will be expanded as we introduce new type constructions in our theory.

In $\mathrm{MLTT}_{\mathcal{U}}$, there is no syntactical distinction between terms and types, since terms can be used in dependent types. The term $\mathbf{App}_{(x:T)U}(M, N)$, which represent the usual application, is usually written MN, omitting the typing informations ${}_{(x:T)}U$. It is used to keep the proof of a typing judgment unique.

A *typing context* Γ is an *ordered* list $(x_1 : T_1), \ldots, (x_n : T_n)$ of pairs of distinct variables and types. The empty context is written $\diamond$. Adding a new element $(x : T)$ to a context Γ is written $\Gamma, (x : T)$, assuming implicitly that x is not already in Γ.

We introduce two judgments :

— $\Gamma \vdash M : T$, which states that M is of type T w.r.t. the typing context Γ of its free variables.

— $\mathbf{wf}(\Gamma)$, the fact that Γ is *well-founded*, i.e. writing Γ as $\Gamma_1, (x : T), \Gamma_2$, T is a type in the context Γ.

They are defined using *inference rules* in Figure 3.1, by a mutual induction. The rules for other constructors of $\mathrm{MLTT}_{\mathcal{U}}$ will be introduced in the next sections.

Even if there is no syntactic distinction between terms and types, when considering the typing judgment $\Gamma \vdash M : T$, we say that M is a term and T is a type. So types always live in some universe $\mathcal{U}_i$.

To stress the fact that contexts are *ordered* lists of pairs of terms and contexts, they are sometimes called *telescopes*. Indeed, considering $\Gamma = (x_1 : T_1) \ldots (x_n : T_n)$, we have

$$\textsc{WF-Empty}\ \frac{}{\mathbf{wf}(\diamond)} \qquad \textsc{WF-Ctx}\ \frac{\Gamma \vdash T : \mathcal{U}_i}{\mathbf{wf}(\Gamma, (x : T))} \qquad \textsc{Var}\ \frac{\mathbf{wf}(\Gamma) \qquad (x : T) \in \Gamma}{\Gamma \vdash x : T}$$

$$\textsc{Univ}\ \frac{\mathbf{wf}(\Gamma)}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}} \qquad \textsc{Univ-Cumul}\ \frac{\Gamma \vdash M : \mathcal{U}_i \qquad j > i}{\Gamma \vdash M : \mathcal{U}_j}$$

$$\Pi\textsc{-Univ}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma, x : T \vdash U : \mathcal{U}_j}{\Gamma \vdash \Pi x : T.U : \mathcal{U}_{max(i,j)}}$$

$$\Pi\textsc{-Intro}\ \frac{\Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x : T.M : \Pi x : T.U} \qquad \Pi\textsc{-Elim}\ \frac{\Gamma \vdash M : \Pi x : T.U \qquad \Gamma \vdash N : T}{\Gamma \vdash \mathbf{App}_{(x:T)U}(M, N) : U\,\{N/x\}}$$

Figure 3.1: $\mathrm{MLTT}_{\mathcal{U}}$ Type System: Contexts, Universes, Dependent Products

$\mathrm{FV}(T_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ for all $i \in \{2, \ldots n\}$. This property is enforced by the inductive definition of $\mathbf{wf}(\Gamma)$.

The usual presentation of MLTT is done with only two universes, $\mathcal{U}_0$ being written as Set, and $\mathcal{U}_1$ as Type. Having no universe $\mathcal{U}$ such that $\mathrm{Type} : \mathcal{U}$, we cannot write any types $\Pi x : \mathrm{Type}.U$. This means that this theory is *predicative*.

As in [Uni13] (and contrary to [Hof97]), we used *Russell-style* universes, as opposed to *Tarski-style* (using the terminology of [MLS84]) where from $\Gamma \vdash T : \mathcal{U}$, we cannot directly consider terms M such that $\Gamma \vdash M : T$, but we rather have a new constructor $\mathbf{Elem}\,T$ so that we can have $\Gamma \vdash M : \mathbf{Elem}\,T$. We can see $\mathbf{Elem}\,T$ as an explicit coercion, which is left implicit when working with Russell-style universes.

Tarski-style universes are well-suited when working on categorical models of dependent type theory, since it makes a distinction between the term T of type $\mathcal{U}$, which is interpreted as a morphism of the category, and the type $\mathbf{Elem}\,T$ which is interpreted as an object of the category. However, in the following we are interested only in the syntactic approach, so that Russell-style universes are convenient and more simple to work with.

The hierarchy of universes is cumulative, as shown by the rule Univ-Cumul which states that a term M of type $\mathcal{U}_i$ can be seen as of type $\mathcal{U}_j$ for $j > i$.

There are three typing rules for the dependent product:

— Π-Univ, which states in which universe $\Pi x : T.U$ lives.

— Π-Intro, which is the usual (dependent) rule for typing λ-abstraction. It is used to *introduce* elements of type $\Pi x : T.U$.

— Π-Elim, which is the usual rule for typing applications. It is used to *eliminate* elements of type $\Pi x : T.U$.

This schemata with three kind of typing rules will be used when introducing new type constructors to the type system.

As explained in the introduction, taking the Curry-Howard glasses, $\Pi x : T.U$ can be seen as the proposition $\forall x : T.U(x)$. When U does not contains the variable x, the type $\Pi x : T.U$ corresponds to the usual functional type $T \to U$.

Suppose that we are able to define the type $\mathbf{List}(n)$ of list of length n, which can be done using inductive types (or W-types) as present in Coq (Section 3.1.8). Then, $\mathbf{List}(1+2)$ and $\mathbf{List}(3)$ would not be equal. To solve this problem, we introduce a notion of *judgmental equality* (also called *conversion*) between terms M, N sharing the same type T, written $\Gamma \vdash M \equiv N : T$. Using it, we can state that if two types T, U are judgmentally equal and M is of type T, then M can be considered as of type U :

$$\textsc{Conv}\ \frac{\Gamma \vdash M : T : \qquad \Gamma \vdash T \equiv U : \mathcal{U}}{\Gamma \vdash M : U}$$

The judgmental equality identifies two β-equivalent terms, as stated with the following rule:

$$\Pi\text{-}\textsc{Conv}\ \frac{\Gamma, x : T \vdash M : U : \qquad \Gamma \vdash N : U}{\Gamma \vdash \mathbf{App}_{(x:T)U}(\lambda x.M, N) \equiv M\{N/x\} : U\{N/x\}}$$

where $M\{N/x\}$ represent the capture-free substitution of x by N in M. Every new construction introduced in our language comes with a new rule for judgmental equality representing the computational content of the construction.

The judgmental equality is an equivalence relation, but we omit the typing rules which enforce it (cf. [Hof97, Uni13]). It is also a congruence, meaning that from $\Gamma, x : T \vdash M \equiv M' : U$ and $\Gamma \vdash T \equiv T' : \mathcal{U}$ we get that $\lambda x : T.M \equiv \lambda x : T'.M' : U$. Once again, we omit these rules, which can also be found in [Hof97, Uni13]. Notice that each time we add a new construction in the type system, we need to add a rule enforcing that $\equiv$ stays a congruence for this new construction. We will come back to this conversion rules in section 3.1.7, since it allows different proofs of the same typing judgment.

Notice that we work here with a *typed* judgmental equality. An other possible presentation is to used an untyped conversion $T \equiv U$ with a rule

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash M : U}\ T \equiv U$$

where $T \equiv U$ can be seen as a judgment at a meta-level. In practice, the two presentation are essentially equivalent, as proven by Siles and Herbelin in [SH12].

One of the crucial property of $\mathrm{MLTT}_{\mathcal{U}}$ is the decidability of the typing checking.

Theorem 1 ([ML73, Hof95a]): *It is decidable in* $\mathrm{MLTT}_{\mathcal{U}}$ *whether a judgment* $\Gamma \vdash M : T$ *is derivable.*

$$\textsc{Id-Univ}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma \vdash M, N : T}{\Gamma \vdash M =_T N : \mathcal{U}_i} \qquad \textsc{Id-Intro}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma \vdash M : T}{\Gamma \vdash \mathbf{refl}_M : M =_T M}$$

$$\textsc{Id-Elim}\ \frac{\begin{array}{c}\Gamma, x, y : T, z : x =_T y \vdash V : \mathcal{U}_i \\ \Gamma, w : T \vdash M : V\,\{w/x\}\,\{w/y\}\,\{\mathbf{refl}_w/z\} \qquad \Gamma \vdash N_1, N_2 : T \qquad \Gamma \vdash p : N_1 =_T N_2\end{array}}{\Gamma \vdash \mathbf{ind}^{=}_{(x,y:T,z:x=_T y)V,\,(w:T)M}(N_1, N_2, p) : V\,\{N_1/x\}\,\{N_2/y\}\,\{p/z\}}$$

$$\textsc{Id-Conv}\ \frac{\Gamma, x, y : T, z : x =_T y \vdash V : \mathcal{U}_i \quad \Gamma, w : T \vdash M : V\,\{w/x\}\,\{w/y\}\,\{\mathbf{refl}_w/z\} \qquad \Gamma \vdash N : T}{\Gamma \vdash \mathbf{ind}^{=}_{(x,y:T,z:x=_T y)V,\,(w:T)M}(N, N, \mathbf{refl}_M) \equiv M\,\{N/z\} : V\,\{M/x\}\,\{M/y\}\,\{\mathbf{refl}_M/z\}}$$

Figure 3.2: MLTT$_{\mathcal{U}}$ Type System: Identity Types

3.1.2 Identity Type

We now present the Identity Type, as introduced by Per Martin-Löf [ML73]. For every pair of terms M_1, M_2 of type T, we introduce a new type $M_1 =_T M_2$ whose terms are "proofs" that M_1 is equal to M_2. When $M_1 =_T M_2$ is inhabited, we say that they are *propositionally equal*, as opposed to judgmentally equal.

This notion of propositional equality has been widely studied but it has gain recent tractions through the advent of Homotopy Type Theory [Uni13]. In that setting, the identity type has been shown to satisfy all the law of a groupoid (actually a weak ∞-groupoid) [HS98, Lum09, GvdB11]. This new insight will be used in Chapter 4 in which we need a careful management of equality.

The typing rules for the Identity Type are given in Figure 3.2. The introduction rule Id-Intro says that two terms that are judgmentally equal are propositionally equal. The elimination rule Id-Elim can be seen as a generalization of Leibniz's principle which states the indiscernability of identicals. Notice that V has x, y, z and T has w as free variables. So $\mathbf{ind}^{=}_{(x,y:T,z:x=_T y)V,\,(w:T)M}(N_1, N_2, p)$ has to bound x, y and z in V and w in M. This is represented respectively by the notation ${}_{(x,y:T,z:x=_T y)}V$ and ${}_{(w:T)}M$.

The rule Id-Conv just state that we can reduce $\mathbf{ind}^{=}_{(x,y:T,z:x=_T y)V,\,(w:T)M}(N_1, N_2, p)$ when N_1 and N_2 are convertible (i.e. judgmentally equal) and p, their proof of equality, is in fact $\mathbf{refl}_M$.

Notice that for closed terms M_1, M_2, propositional and judgment equality coincide. We can impose that they coincide on any pair of terms with the following *reflection rule*

$$\textsc{Ext}\ \frac{\Gamma \vdash N : M_1 =_T M_2}{\Gamma \vdash M_1 \equiv M_2 : T}$$

Such a theory is called *extensional* [MLS84], as opposed to *intentional*. The problem with

this rule is that checking the correctness of a judgment $\Gamma \vdash M : T$, i.e. the type-checking, becomes undecidable. This is why, in the following, we only consider the intentional version of $\mathrm{MLTT}_{\mathcal{U}}$, *i.e.* without the reflection rule.

We now introduce some basic constructions on Identity types that are used in Chapter 4. First, as mentioned above, the Identity Type has the structure of an ∞-groupoid and as such, their exists an inverse for p^{-1} of type $N =_T M$ for every equality proof p of type $M =_T N$, defined as $\mathbf{ind}^{=}_{(x,y:T)y =_T x, (w:T)\mathbf{refl}_w}(N_1, N_2, p)$. Similarly, propositional equalities can be composed, and satisfies various laws. We refer the reader to [Uni13] for details.

Taking $V : T \to \mathcal{U}_i$ a type family over T, we can build a function $\mathbf{transport}^V_T$ of type $\Pi u, v : T.(u =_T v) \to Vu \to Vv$ as

$$\lambda u, v : T.\lambda p : (u =_T v).\mathbf{ind}^{=}_{(x,y:T)(Vx \to Vy), (w:T)\lambda x : (Vw).x}(u, v, p)$$

This is useful to rewrite in V an element M into an element N as soon as we have a element of $M =_T N$.

An other interesting construction is $\mathbf{ap}_f$ which take a function $f : T \to U$, two elements u, v of T and an element of $u =_T v$ and return an element of $fu =_U fv$. It is defined as

$$\lambda u, v : T.\lambda p : (u =_T v).\mathbf{ind}^{=}_{(x,y:T).(fx =_U fy), (w:T).\mathbf{refl}_{fw}}(u, v, p)$$

3.1.3 Proof Irrelevance

Type Theory is used to formalize mathematics, by formalizing objects of a theory and properties between those objects. Following the Curry-Howard correspondence, elements which inhabit properties on objects represent proofs. However, in mathematics, the proof of a property does not depend in general on the proof of others properties, but just on the fact that they are true, i.e. inhabited. This idea is called *proof-irrelevance*. Thus, when we have $\Gamma \vdash M : P$ where P represents a property (for example that the set of prime numbers is infinite), the only important fact is that P seen as a type is inhabited. This means that if we have an other term N such that $\Gamma \vdash N : P$, we would like to get that M and N are propositionally equal. To do so, we define the predicate $\mathrm{isProp}_{\mathcal{U}}$ as

$$\lambda P : \mathcal{U}.\Pi x, y : P.\ x =_P y$$

so that when $\Gamma \vdash P : \mathcal{U}$, we have $\mathrm{isProp}_{\mathcal{U}}(P)$ which represents exactly this idea. The types P which satisfies $\mathrm{isProp}_{\mathcal{U}}(P)$ are called *mere propositions*.

Using this predicate, we can define the notion of *sets*, as type whose equality on elements is a proposition, *i.e.* which satisfies the so-called *Uniqueness of Identity Proofs* (UIP). It is represented by the predicate $\mathrm{isSet}_{\mathcal{U}}$ defined as

$$\lambda A : \mathcal{U}.\Pi a, b : A.\ \mathrm{isProp}_{\mathcal{U}}(a =_A b).$$

$$\Sigma\text{-Univ}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma, x : T \vdash U : \mathcal{U}_i}{\Gamma \vdash \Sigma x : T.U : \mathcal{U}_i} \qquad \Sigma\text{-Intro}\ \frac{\Gamma \vdash M : T \qquad \Gamma \vdash N : U\{M/x\}}{\Gamma \vdash \langle M, N\rangle_{(x:T)U} : \Sigma x : T.U}$$

$$\text{Proj-1}\ \frac{\Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \pi_1 M : T} \qquad \text{Proj-2}\ \frac{\Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \pi_2 M : U\{\pi_1 M/x\}}$$

$$\text{Conv-Proj1}\ \frac{\Gamma \vdash \langle M, N\rangle_{(x:T)U} : \Sigma x : T.U}{\Gamma \vdash \pi_1\langle M, N\rangle_{(x:T)U} \equiv M : T} \qquad \frac{\Gamma \vdash \langle M, N\rangle_{(x:T)U} : \Sigma x : T.U}{\Gamma \vdash \pi_2\langle M, N\rangle_{(x:T)U} \equiv N : U\{N/x\}}\ \text{Conv-Proj2}$$

Figure 3.3: $\text{MLTT}_{\mathcal{U}}$ Type System: Dependent Sums

Notice that $\text{isProp}_{\mathcal{U}}(P)$ and $\text{isSet}_{\mathcal{U}}(A)$ are themselves mere propositions, as proven in [Uni13].

Homotopy Type Theory [Uni13] is interested in studying higher-order generalization of this idea, called n-Types. For example, we could consider the predicate $\text{is1-Type}_{\mathcal{U}}$ defined as $\lambda A : \mathcal{U}.\Pi a, b : A.\ \text{isSet}_{\mathcal{U}}(a =_A b)$. The study of such higher types, and more particularly the status of the *axiom of univalence*, goes beyond the scope of this thesis.

The goal of this work is to reason on Programming Language, for which the notion of sets seems enough for the formalization. In particular, in Chapter 4, we work with $\text{MLTT}_{\mathcal{U}}$ + UIP and thus is incompatible with univalence and implies that all types can be seen as sets. Nevertheless, we have tried to be as independent from UIP as possible, so that the work could be generalized to $\text{MLTT}_{\mathcal{U}}$.

3.1.4 Dependent Sums and Subset Types

We now introduce the notion of product between two types, such that the second component of the product can depend on the first component. Such a type is called a *Dependent Sum*.

Taking a type $T : \mathcal{U}_i$ and a type family $U : T \to \mathcal{U}_i$, we can form a new type, called a *dependent sum* and written $\Sigma x : T.U$. The inhabitants of this types are pairs $\langle M, N\rangle_{(x:T)U}$ with $M : T$ and $N : T\{N/x\}$. Moreover, from a pair $\langle M, N\rangle_{(x:T)U}$, we wish to retrieve M and N. This is done using the usual projections π_1, π_2, such that if $M : \Sigma x : T.U$ then $\pi_1 M : T$ and $\pi_2 M : U\{\pi_1 M/x\}$. And since we want to compute with pairs, we need the conversion rules $\pi_1\langle M, N\rangle_{(x:T)U} \equiv_T M$ and $\pi_2\langle M, N\rangle_{(x:T)U} \equiv_{U\{\pi_1 M/x\}} N$. All of this is summed up in Figure 3.1.4.

When the type family $U : T \to \mathcal{U}_i$ is constant, i.e. it does not depend on T, the dependent sum $\Sigma x : T.U$ correspond to the usual Cartesian product, and is written $T \times U$. Then, taking $M : T \times U$ we have $\pi_2 M : U$ since there is no more dependency of

U over T.

The projections π_1 and π_2 provide a too weak elimination form because M is not convertible to $\langle \pi_1 M, \pi_2 M \rangle_{(x:T)U}$ in general. So the general elimination of a dependent sum can not be deduced from π_1 and π_2. That is, we need to introduce a stronger rule Elim-Σ for elimination of dependent sums:

$$\frac{\Gamma, z : \Sigma x : T.U \vdash V : \mathcal{U} \qquad \Gamma, x : T; y : U \vdash N : V\left\{\langle x, y\rangle_{(x:T)U}/z\right\} \qquad \Gamma \vdash M : \Sigma x : T.U}{\Gamma \vdash \mathbf{ind}^{\Sigma}({}_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, M) : V\{M/z\}}$$

Notice that N has x and y as free variables, which do not appear in Γ, so we need to bound them using the special notation ${}_{(x:T,y:U)}$. This means that $\mathbf{ind}^{\Sigma}$ actually binds z in V and x, y in M. This rule is similar to the usual elimination of existential quantification in Sequent Calculus. This is not a surprise since under the Curry-Howard correspondence, $\Sigma x : T.P$ is seen as the usual quantification $\exists x : T.P(x)$.

The conversion rule of $\mathbf{ind}^{\Sigma}$ is given by the following rule Conv-$\mathbf{ind}^{\Sigma}$:

$$\frac{\begin{array}{c}\Gamma, z : \Sigma x : T.U \vdash V : \mathcal{U} \\ \Gamma, x : T; y : U \vdash N : V\left\{\langle x, y\rangle_{(x:T)U}/z\right\} \qquad \Gamma \vdash M_1 : T \qquad \Gamma \vdash M_2 : U\{M_1/x\}\end{array}}{\Gamma \vdash \mathbf{ind}^{\Sigma}({}_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, \langle M_1, M_2\rangle_{(x:T)U}) \equiv N\{M_1/x\}\{M_2/y\} : V\left\{\langle M_1, M_2\rangle_{(x:T)U}/z\right\}}$$

The two projections can in fact be defined using $\mathbf{ind}^{\Sigma}$ as

$$\begin{array}{rcl}\pi_1 & = & \mathbf{ind}^{\Sigma}({}_{(z:\Sigma x:T.U)}T, {}_{(x:T,y:U)}x, M) \\ \pi_2 & = & \mathbf{ind}^{\Sigma}({}_{(z:\Sigma x:T.U)}U, {}_{(x:T,y:U)}y, M)\end{array}$$

We easily check that the rules Conv-Proj1 and Conv-Proj1 are then just special instances of Conv-$\mathbf{ind}^{\Sigma}$.

Taking $M : T$, it can exist different $N : U\{M/x\}$ such that $\langle M, N\rangle_{(x:T)U} : \Sigma x : T.U$. However, when U is a family of mere proposition, there is unicity of such N. So taking $\Gamma, x : T \vdash P :\to \mathcal{U}$ a family of types such that $\Gamma, x : T \vdash \mathrm{isProp}_{\mathcal{U}}(P)$ (i.e. for all $x : T$, P is a mere proposition), the dependent sum $\Sigma x : T.P$ is called a *subset type*, and is written $\{x : T \mid P\}$. By an abuse of notation, we write $\Gamma \vdash M : \{x : T \mid P\}$ when there exist N such that $\Gamma \vdash \langle M, N\rangle_{(x:T)U} : \Sigma x : T.P$. We call P a proof obligation of M, and we refer to N as $M.P$. We also write $\Sigma x : T.P, Q$ for $\Sigma x : T.P \wedge Q$.

Using subset types, we can define types $\mathrm{Prop}_{\mathcal{U}_i}$ and $\mathrm{Set}_{\mathcal{U}_i}$ as $\{P : \mathcal{U}_i \mid \mathrm{isProp}_{\mathcal{U}_i} P\}$ and $\{X : \mathcal{U}_i \mid \mathrm{isSet}_{\mathcal{U}_i} X\}$, since isProp and isSet are mere propositions.

3.1.5 Empty and Unit Types

We now introduce two new types:

$$\mathbf{0}\text{-Univ}\ \frac{\mathbf{wf}(\Gamma)}{\Gamma \vdash \mathbf{0} : \mathcal{U}_i} \qquad \mathbf{0}\text{-Elim}\ \frac{\Gamma, x : \mathbf{0} \vdash V : \mathcal{U}_i \qquad \Gamma \vdash N : \mathbf{0}}{\Gamma \vdash \mathbf{ind^0}((x{:}\mathbf{1})V, N) : V\,\{N/x\}}$$

$$\mathbf{1}\text{-Univ}\ \frac{\mathbf{wf}(\Gamma)}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i} \qquad \mathbf{1}\text{-Intro}\ \frac{\mathbf{wf}(\Gamma)}{\Gamma \vdash \star : \mathbf{1}}$$

$$\mathbf{1}\text{-Elim}\ \frac{\Gamma, x : \mathbf{1} \vdash V : \mathcal{U}_i \qquad \Gamma \vdash M : V\,\{\star/x\} \qquad \Gamma \vdash N : \mathbf{1}}{\Gamma \vdash \mathbf{ind^1}((x{:}\mathbf{1})V, M, N) : V\,\{N/x\}}$$

$$\mathbf{1}\text{-Conv}\ \frac{\Gamma, x : \mathbf{1} \vdash V : \mathcal{U}_i \qquad \Gamma \vdash M : V\,\{\star/x\}}{\Gamma \vdash \mathbf{ind^1}((x{:}\mathbf{1})V, M, \star) \equiv M : V\,\{\star/x\}}$$

Figure 3.4: MLTT$_\mathcal{U}$ Type System: Empty and Unit Types

— **0**, a type inhabited by no terms, which correspond to the empty set,
— **1**, a type inhabited by a unique term $\star$,
with their respective elimination terms $\mathbf{ind^0}$ and $\mathbf{ind^1}$. Their typing rules are given in Figure 3.4.

Seen as propositions, they correspond respectively to *True* and *False*. A proof of the non-existence of an inhabitant of **0** corresponds thus to the *consistency* of our system. Indeed, for any type $T : \mathcal{U}_i$, we can define the following term

$$\lambda x : \mathbf{0}.\mathbf{ind^0}(T, x)$$

which is of type $\mathbf{0} \to T$. So if **0** is inhabited, then any type is!

3.1.6 Coproduct and Natural Number Type

We are still missing some important basic objects in our theory, like disjoint union and natural numbers. We now introduce new types and type constructors to build them.

First, we construct *coproduct types*, which are a way to build a disjoint union of two types. Its typing rules are defined in Figure 3.5. Taking two terms $M_l : T_1 \to U_1$ and $M_r : T_2 \to U_2$, we define $\mathbf{lift}^+_{(T_1+T_2, U_1+U_2)}(M_1, M_2)$ of type $T_1+T_2 \to U_1+U_2$ as $\lambda z : T_1+T_2.\mathbf{ind}^+(U_1+U_2, (x{:}T)(M_l x), (y{:}U)(M_r y), z)$.

Using coproducts, we can define the type of *Booleans*, written **Bool**, simply as **1**+**1**. Then, defining **true** as $\mathbf{inl}(\star)$ and **false** as $\mathbf{inr}(\star)$, we can use the non-dependent version of $\mathbf{ind}^+$ to perform case analysis (Figure 3.6).

Finally, we define the type **Nat** of natural numbers. It is inhabited by a term 0, and a successor function **S** take a natural number as argument to build a new one. Thus the

$$+\text{-Univ}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma \vdash U : \mathcal{U}_i}{\Gamma \vdash T{+}U : \mathcal{U}_i}$$

$$+\text{-IntroL}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma \vdash U : \mathcal{U}_i \qquad \Gamma \vdash M : T}{\Gamma \vdash \mathbf{inl}(M) : T{+}U}$$

$$+\text{-IntroR}\ \frac{\Gamma \vdash T : \mathcal{U}_i \qquad \Gamma \vdash U : \mathcal{U}_i \qquad \Gamma \vdash M : U}{\Gamma \vdash \mathbf{inr}(M) : T{+}U}$$

$$+\text{-Elim}\ \frac{\begin{array}{c}\Gamma, z : T{+}U \vdash V : \mathcal{U}_i \\ \Gamma, x : T \vdash M_l : V\,\{\mathbf{inl}(x)/z\} \qquad \Gamma, y : U \vdash M_r : V\,\{\mathbf{inr}(y)/z\} \qquad \Gamma \vdash N : T{+}U\end{array}}{\Gamma \vdash \mathbf{ind}^{+}({}_{(z:T+U)}V, {}_{(x:T)}M_l, {}_{(y:U)}M_r, N) : V\,\{N/z\}}$$

$$+\text{-ConvL}\ \frac{\begin{array}{c}\Gamma, z : T{+}U \vdash V : \mathcal{U}_i \\ \Gamma, x : T \vdash M_l : V\,\{\mathbf{inl}(x)/z\} \qquad \Gamma, y : U \vdash M_r : V\,\{\mathbf{inr}(y)/z\} \qquad \Gamma \vdash N : T\end{array}}{\Gamma \vdash \mathbf{ind}^{+}({}_{(z:T+U)}V, {}_{(x:T)}M_l, {}_{(y:U)}M_r, \mathbf{inl}(N)) \equiv M_l\,\{N/x\} : V\,\{\mathbf{inl}(N)/z\}}$$

$$+\text{-ConvR}\ \frac{\begin{array}{c}\Gamma, z : T{+}U \vdash V : \mathcal{U}_i \\ \Gamma, x : T \vdash M_l : V\,\{\mathbf{inl}(x)/z\} \qquad \Gamma, y : U \vdash M_r : V\,\{\mathbf{inr}(y)/z\} \qquad \Gamma \vdash N : U\end{array}}{\Gamma \vdash \mathbf{ind}^{+}({}_{(z:T+U)}V, {}_{(x:T)}M_l, {}_{(y:U)}M_r, \mathbf{inr}(N)) \equiv M_r\,\{N/y\} : V\,\{\mathbf{inr}(N)/z\}}$$

Figure 3.5: $\mathrm{MLTT}_{\mathcal{U}}$ Type System: Coproducts

$$\mathbf{Bool}\text{-ConvL}\ \frac{\Gamma \vdash V : \mathcal{U}_i \qquad \Gamma \vdash M_1 : V \qquad \Gamma \vdash M_2 : V}{\Gamma \vdash \mathbf{ind}^{+}(V, M_1, M_2, \mathbf{true}) \equiv M_1 : V}$$

$$\mathbf{Bool}\text{-ConvR}\ \frac{\Gamma \vdash V : \mathcal{U}_i \qquad \Gamma \vdash M_1 : V \qquad \Gamma \vdash M_2 : V}{\Gamma \vdash \mathbf{ind}^{+}(V, M_1, M_2, \mathbf{false}) \equiv M_2 : V}$$

Figure 3.6: $\mathrm{MLTT}_{\mathcal{U}}$ Type System: Booleans

$$\textbf{Nat-}\textsc{Univ}\ \frac{\mathbf{wf}(\Gamma)}{\Gamma \vdash \mathbf{Nat} : \mathcal{U}_i}$$

$$\textbf{Nat-}\textsc{Intro0}\ \frac{\mathbf{wf}(\Gamma)}{\Gamma \vdash 0 : \mathbf{Nat}}$$

$$\textbf{Nat-}\textsc{IntroS}\ \frac{\Gamma \vdash M : \mathbf{Nat}}{\Gamma \vdash \mathbf{S}(M) : \mathbf{Nat}}$$

$$\textbf{Nat-}\textsc{Elim}\ \frac{\begin{array}{ccc}\Gamma, x : \mathbf{Nat} \vdash V : \mathcal{U}_i & & \\ \Gamma \vdash M_0 : V\,\{0/x\} & \Gamma, x : \mathbf{Nat}, y : V \vdash M_{\mathbf{S}} : V\,\{\mathbf{S}(x)/x\} & \Gamma \vdash N : \mathbf{Nat}\end{array}}{\Gamma \vdash \mathbf{ind}^{\mathbf{Nat}}({}_{(x:\mathbf{Nat})}V, M_0, {}_{(x:\mathbf{Nat},y:V)}M_{\mathbf{S}}, N) : V\,\{N/x\}}$$

$$\textbf{Nat-}\textsc{Conv0}\ \frac{\Gamma, x : \mathbf{Nat} \vdash V : \mathcal{U}_i \qquad \Gamma \vdash M_0 : V\,\{0/x\} \qquad \Gamma, x : \mathbf{Nat}, y : V \vdash M_{\mathbf{S}} : V\,\{\mathbf{S}(x)/x\}}{\Gamma \vdash \mathbf{ind}^{\mathbf{Nat}}({}_{(x:\mathbf{Nat})}V, M_0, {}_{(x:\mathbf{Nat},y:V)}M_{\mathbf{S}}, 0) \equiv M_0 : V\,\{0/z\}}$$

$$\textbf{Nat-}\textsc{ConvS}\ \frac{\begin{array}{ccc}\Gamma, x : \mathbf{Nat} \vdash V : \mathcal{U}_i & & \\ \Gamma \vdash M_0 : V\,\{0/x\} & \Gamma, x : \mathbf{Nat}, y : V \vdash M_{\mathbf{S}} : V\,\{0/x\} & \Gamma \vdash N : \mathbf{Nat}\end{array}}{\Gamma \vdash \mathbf{ind}^{\mathbf{Nat}}({}_{(x:\mathbf{Nat})}V, M_0, {}_{(x:\mathbf{Nat},y:V)}M_{\mathbf{S}}, \mathbf{S}(N)) \equiv M_{\mathbf{S}}\,\{\mathbf{S}(N)/x\}\,\{M'/y\} : V\,\{\mathbf{S}(N)/x\}}$$

$$\text{with } M' = \mathbf{ind}^{\mathbf{Nat}}({}_{(x:\mathbf{Nat})}V, M_0, {}_{(x:\mathbf{Nat},y:V)}M_{\mathbf{S}}, N)$$

Figure 3.7: MLTT$_{\mathcal{U}}$ Type System: Natural Numbers

natural number 1 is defined as $\mathbf{S}(0)$, as it is usually done in Peano Arithmetic. The typing rules are given in Figure 3.7.

Compare to the eliminator for coproducts, the one for **Nat** has one main difference, namely the fact that $M_{\mathbf{S}}$ has en extra free variable y of type V. This variable allows the induction hypothesis to build the term which inhabit V.

Using $\mathbf{ind}^{\mathbf{Nat}}$, we can define the addition function **plus** of type $\mathbf{Nat} \to \mathbf{Nat} \to \mathbf{Nat}$ as

$$\lambda n : \mathbf{Nat}.\lambda m : \mathbf{Nat}.\mathbf{ind}^{\mathbf{Nat}}({}_{(x:\mathbf{Nat})}\mathbf{Nat}, n, {}_{(x:\mathbf{Nat},y:\mathbf{Nat})}\mathbf{S}(y), m).$$

In practice, it is often easier to define new functions via *pattern-matching* rather than using $\mathbf{ind}^{\mathbf{Nat}}$. For example, the addition function **plus** can be defined as

$$\begin{array}{l}\lambda n : \mathbf{Nat}.\lambda m : \mathbf{Nat}.\texttt{match } n \texttt{ with}\\ \quad \mid 0 \texttt{ => } m\\ \quad \mid \mathbf{S}n' \texttt{ => } \mathbf{S}(\mathbf{plus}\ n'\ m)\end{array}$$

Defining such recursive terms with pattern-matching instead of the eliminator on **Nat** is not always possible. Here, it is possible because **plus** is recursively called on n' which is strictly smaller than $n = \mathbf{S}n$. However, when it is clear how to define such terms with pattern-matching with $\mathbf{ind}^{\mathbf{Nat}}$, we will allow us to use them.

3.1.7 Type Theory with Explicit Coercions

A proof of $\Gamma \vdash M : T$ is not unique in general, because the conversion rule can be applied silently at various places. This is usually not problematic but in the forcing translation defined in Chapter 4, we need to make explicit the conversion rule. This rises two issues: (i) the translation may depend on the actual choice of the application of the conversion rule, (ii) the existence of a typing judgment with explicit is not guarantee.

The problem of making the conversion explicit already has already been considered by Curien et al. when defining a categorical interpretation of Martin-Löf Type Theory [Cur93, CGH14]. It has also been studied by Geuvers et al. in [GW08, vDGW13], for Martin-Löf Logical Framework and more generally for any Pure Type System.

They both propose to introduce a new typing judgment $\Gamma \vdash_e M : T$, which is defined in the same way than $\Gamma \vdash M : T$ but for the rule Conv, which uses an explicit coercion $\mathbf{c}_{T,U}(M)$:

$$\textsc{Conv}\ \frac{\Gamma \vdash_e M : T : \qquad \Gamma \vdash T \equiv U : \mathcal{U}}{\Gamma \vdash_e \mathbf{c}_{T,U}(M) : U}$$

The introduction of this explicit coercion can be seen as a way to transform an extentional theory to an intentional one, as presented in [Hof95a, Our05].

In [Cur93, CGH14], this explicit coercion is handled by adding new judgmental equalities in the system. To be even more explicit, those new equalities should also appear explicitly in a term. So a complete language for describing coercions should be developed, as done for instance in a simpler setting in [CG94]. However, they have refrained to do so, invoking "a heavy notational apparatus".

In [GW08], the explicit coercion is handled using an heterogeneous notion of equality (a.k.a. JMEq, as introduced by Conor McBride [McB00]). Using such an heterogeneous equality only makes sense in presence of UIP.

In both work, it is possible to define a function $|\_|$ which transforms a term of $\mathrm{MLTT}^e_{\mathcal{U}}$ to a term of $\mathrm{MLTT}_{\mathcal{U}}$ by simply removing all the explicit conversion coercions, so that $\mathbf{c}_{T,U}(M)$ is transformed into M. This function is extended straightforwardly to contexts. Their system satisfy the two following theorems.

Theorem 2: *Suppose* $\Gamma \vdash_e M : T$*, then* $|\Gamma| \vdash |M| : |T|$.

Theorem 3: *Suppose* $\Gamma \vdash M : T$*, then there exists* Γ_e, M_e, T_e *a context and two terms of* $\mathrm{MLTT}^e_{\mathcal{U}}$ *such that*

— $|\Gamma_e| = \Gamma, |M_e| = M$ *and* $|T_e| = T$,
— $\Gamma_e \vdash_e M_e : T_e$.

Unfortunately, we can not reuse the work of Curien et al. out of the box because the type theory we consider is intentional and without explicit substitutions. However, a formal study of this system goes beyond the scope of this thesis and we assume that an operation $|\_|$ satisfying Theorems 2 and 3 also exists for $\mathrm{MLTT}^e_{\mathcal{U}}$. That is, we leave the expression of judgmental equality on $\mathbf{c}_{T,U}(M)$ and the proofs of the corresponding theorems as future work.

Nevertheless, we will use in Chapter 4 the fact that those equalities correspond to equality satisfies by $\mathbf{transport}^{\lambda T,T}_{\mathcal{U}}$ because $\mathbf{c}_{T,U}(M)$ is a rewriting using the judgemental equality on types in the same $\mathbf{transport}^{\lambda T,T}_{\mathcal{U}}$ is a rewriting using the propositional equality.

3.1.8 The Type System of Coq

A variant of the type system we have presented have been implemented in the proof assistant Coq. It is based on the *Calculus of Inductive Constructions* (CIC), which differs in some points to $\mathrm{MLTT}_{\mathcal{U}}$. We briefly review the differences because Coq is the proof assistant in which the forcing translation of Chapter 4 has been implemented.

First, CIC has a special universe, Prop, to represent propositions, rather than being define internally in the theory using a predicate isProp as we have done in Section 3.1.3. This universe is *impredicative* in the sense that the dependent product $\Pi P : \mathrm{Prop}.Q$ is of type Prop, when Q is of type Prop. Prop can in fact be identified with $\mathbf{Prop}_{\mathcal{U}}$ as defined in 3.1.4, and impredicativity can be recovered from a *propositional resizing axiom* [Uni13], which states that there exists an equivalence between $\mathbf{Prop}_{\mathcal{U}_{i+1}}$ and $\mathbf{Prop}_{\mathcal{U}_i}$. In Coq, propositions do not automatically satisfies proof-irrelevance, but we can suppose it using an axiom. The other universe $\mathcal{U}_i$ are noted Type_i, but for $\mathcal{U}_0$, which is noted Set.

CIC has inductive types as built-in features, which are used to define Dependent Sums, Coproducts, Empty and Unit Type, Natural numbers and more importantly, the Identity Type. The Identity Type is usually defined in Coq as going into the universe Prop, i.e. taking $T : \mathrm{Type}$ and $M, N : T$ we have $T =_N M : \mathrm{Prop}$. This is coherent with the fact that $T =_N M$ is indeed seen as a proposition, however, when working with a proof-irrelevant universe Prop, this imposes Uniqueness of Identity Proof, so that every type is a set.

3.2 Call-By-Value Languages with References

We now introduce the programming languages which are the objects of study of this thesis. They are fragments of ML, namely typed functional languages with a call-by-value reduction and *nominal references*.

In its unrestricted form, the language that we consider has higher-order references *i.e.* it can store functions into references. We call this language RefML.

3.2.1 Syntax and Operational Semantics of RefML

The grammar of types of RefML is

$$\tau, \sigma \stackrel{def}{=} \text{Unit} \mid \text{Bool} \mid \text{Int} \mid \text{ref}\,\tau \mid \tau \to \sigma.$$

We write Types for the set of types generated by this grammar. A type is said to be *ground* (or *non-functional*) if is not equal to $\tau \to \sigma$. So $\text{ref}\,(\tau \to \sigma)$ is among others ground.
The syntax of values, terms and applicative contexts of RefML is given by

$$\begin{array}{lll}
v & \stackrel{def}{=} & () \mid \mathbf{true} \mid \mathbf{false} \mid \widehat{n} \mid x \mid l \mid \lambda x:\tau.M \mid \Omega_\tau \qquad (\text{where } n \in \mathbb{Z}, l \in \text{Loc}) \\
M, N & \stackrel{def}{=} & v \mid MN \mid M + N \mid \texttt{if}\ M_1\ \texttt{then}\ M_2\ \texttt{else}\ M_3 \mid M == N \mid \\
 & & \text{ref}\,M \mid !M \mid M := N \\
C & \stackrel{def}{=} & \bullet \mid \lambda x:\tau.C \mid CM \mid MC \mid C + M \mid M + C \mid \texttt{if}\ C\ \texttt{then}\ M\ \texttt{else}\ M' \mid \\
 & & \text{ref}\,C \mid !C \mid C := M \mid M := C \mid C == M \mid M == C \\
K & \stackrel{def}{=} & \bullet \mid KM \mid vK \mid K + M \mid v + K \mid \texttt{if}\ K\ \texttt{then}\ M\ \texttt{else}\ M' \mid \text{ref}\,K \mid \\
 & & !K \mid K := M \mid v := K \mid K == M \mid v == K
\end{array}$$

where v denotes values, M denotes terms, C denotes contexts and K denotes applicative contexts. Applicative contexts are particular kinds of contexts, the ones which reduce directly terms that fill their hole $\bullet$. We use special terms Ω_τ for each type τ that always diverge.

As usual, $\texttt{let x =}\ N\ \texttt{in}\ M$ is defined as $(\lambda x.M)N$ and $M; N$ is defined as $(\lambda x.M)N$ with x fresh in M.

Locations live in sets Loc_τ where τ is the type of values they are storing. We define Loc as $\cup_\tau \text{Loc}_\tau$ and Loc_ϕ as $\cup_{\sigma,\tau} \text{Loc}_{\sigma\to\tau}$. There is no way to perform any arithmetic on locations. That is, taking $l \in \text{Loc}$, $l + 1$ is not a valid term (it is not well-typed as we will see). However, it is possible to compare locations, *i.e.* taking $l_1, l_2 \in \text{Loc}_\tau$, $l_1 == l_2$ is a valid term.

Heaps h are defined as finite partial maps $\text{Loc} \rightharpoonup \text{Val}$ respecting types, *i.e.* $h(l)$ is a closed value of type τ when $l \in \text{Loc}_\tau$. The empty heap is written ε. Adding a new element to a partial map h is written $h \cdot [l \hookrightarrow v]$, and is defined only if $l \notin \text{dom}(h)$. We

$$
\begin{array}{lcll}
(K[(\lambda x.M)v], h) & \mapsto & (K[M\{v/x\}], h) & \\
(K[\widehat{n} + \widehat{m}], h) & \mapsto & (K[\widehat{n+m}], h) & \\
(K[\widehat{n} == \widehat{m}], h) & \mapsto & (K[\mathbf{true}], h) & (n = m) \\
(K[\widehat{n} == \widehat{m}], h) & \mapsto & (K[\mathbf{false}], h) & (n \neq m) \\
(K[\Omega_\tau], h) & \mapsto & (K[\Omega_\tau], h) & \\
(K[l == l], h) & \mapsto & (K[\mathbf{true}], h) & (l = l') \\
(K[l == l'], h) & \mapsto & (K[\mathbf{false}], h) & (l \neq l') \\
(K[!l], h) & \mapsto & (K[v], h) & (h(l) = v) \\
(K[\mathrm{ref}\, v], h) & \mapsto & (K[l], h \cdot [l \hookrightarrow v]) & (l \notin \mathrm{dom}(()h)) \\
(K[l := v], h) & \mapsto & (K[()], h[l \hookrightarrow v]) & (l \in dom(h)) \\
(K[\texttt{if } \mathbf{true} \texttt{ then } M_1 \texttt{ else } M_2], h) & \mapsto & (K[M_1], h) & \\
(K[\texttt{if } \mathbf{false} \texttt{ then } M_1 \texttt{ else } M_2], h) & \mapsto & (K[M_2], h) &
\end{array}
$$

Figure 3.8: Operational Semantics of RefML

also define $h[l \hookrightarrow v]$, for $l \in \mathrm{dom}(h)$, as the partial function h' which satisfies $h'(l') = h(l')$ when $l' \neq l$, and $h'(l) = v$. The restriction of a heap h to a set of locations L is written $h_{|L}$. A heap is said to be *closed* when, for all $l \in \mathrm{dom}(h)$, if $h(l)$ is itself a location l' then $l' \in \mathrm{dom}(h)$. Taking a set L of locations and h a heap, we define the image of L by h, written $h^*(L)$ as $h^*(L) \stackrel{def}{=} \bigcup_{j \leq 0} h^j(L)$ and $h^0(L) = L$, $h^{j+1}(L) = h(h^j(L)) \cap \mathrm{Loc}$. Using it, we define $\mathbf{Cl}(L)$, the set of minimal closed heaps whose domain contains L, as $\mathbf{Cl}(L) \stackrel{def}{=} \{h \mid \mathrm{dom}(h) = h^*(L), \forall l \in \mathrm{dom}(h), \nu_L(h(l)) \subseteq \mathrm{dom}(h)\}$.

The small step operational semantics of RefML, written $(M, h) \mapsto (M', h')$, is defined in Figure 3.8. We write $M\{v/x\}$ to represent the (capture-free) substitution of x by v in M.

This reduction is deterministic, so we suppose that the reduction $(K[\mathrm{ref}\, v], h) \mapsto (K[l], h \cdot [l \hookrightarrow v])$ chooses a location $l \notin \mathrm{dom}(h)$. We also consider the non-deterministic reduction $\mapsto_{nd}$, defined in the same way but for the rule of allocation, which is such that $(K[\mathrm{ref}\, v], h) \mapsto_{nd} (K[l], h \cdot [l \hookrightarrow v])$ for any $l \notin \mathrm{dom}(h)$.

For example, the term $\mathrm{ref}\, 1 == \mathrm{ref}\, 1$ reduces to **false**. Note that, when working with references which can only store integers (or booleans), the equality on locations can be encoded using the following term (as shown for example in [PS98]):

$$\lambda \mathtt{x} : \mathrm{ref\ Int}.\lambda \mathtt{y} : \mathrm{ref\ Int}.\mathtt{let\ v} = !\mathtt{x\ in}\ ((\mathtt{x} := !\mathtt{y} + \mathtt{1});\ \mathtt{let\ b} = (!\mathtt{x} == !\mathtt{y})\ \mathtt{in}\ (\mathtt{x} := \mathtt{v}; \mathtt{b}))$$

The idea is simply to test if x and y are aliased, by modifying x and checking if y has also been modified. However, it seems impossible to encode the equality on locations of type ref Unit, as we cannot use the modification $\mathtt{x} := !\mathtt{y} + \mathtt{1}$ or any other modifications (since () is the only inhabitant of Unit) to test if x, y are indeed aliased.

With higher-order references, usual fixpoints $\mathtt{fix\ f(x).M}$ of type $\tau \to \sigma$ can be

defined using the so-called *Landin's Knot* :

$$\texttt{let y} = \texttt{ref}\ (\lambda \texttt{x}.\Omega_{\tau\to\sigma})\ \texttt{in y} := (\lambda \texttt{x.let f} = !\texttt{y in M}); !\texttt{y}$$

where $\Omega_{\tau\to\sigma}$ is a term of type $\tau \to \sigma$ which always diverges (like `undefined` in Haskell or `Obj.Magic` in OCaml).

One has $(\texttt{fix f(x).M}, h) \to (\lambda \texttt{x.let f} = !\texttt{l}_\texttt{y}\ \texttt{in M}, h \cdot [\texttt{l}_\texttt{y} \hookrightarrow \lambda \texttt{x.let f} = !\texttt{y in M}])$. So taking a value of type τ, we get the following reduction

$$((\texttt{fixf(x).M})\texttt{v}, h) \to^* (\texttt{M}\{\texttt{v/x}\}\{\texttt{l}_\texttt{y}/\texttt{f}\}, h \cdot [\texttt{l}_\texttt{y} \hookrightarrow \lambda \texttt{x.let f} = !\texttt{y in M}])$$

As we see, the term M accesses to the recursive call f via the heap.

In the following, we write E for terms which are either values or of the form $K[f\ v]$ with f a free variable and v a value. These last terms are called *callbacks*. One cannot reduce these two kinds of terms.

3.2.2 Typing Rules

We now introduce the typing judgments of RefML. To deal with creation of locations, we use a special context Σ which assign a type to the values a location can store. This is because locations, in contrast to variables, can have only one fixed type. So typing judgments are of the form $\Sigma; \Gamma \vdash M : \tau$, where Γ is a variable context and $\Sigma = (l_1, \ldots, l_n)$ is a location context. Notice that we do not need to indicate the types of locations of l in Σ, since the membership $l \in \mathrm{Loc}_\tau$ already gives its type. The typing rules of RefML are given in Figure 3.9. When Σ is empty, we simply write $\Gamma \vdash M : \tau$

We write $\mathrm{FV}(M)$ for the set of free variables appearing in M. In the following, we make a clear distinction between variables of ground types and variables of functional types. Indeed, we deal abstractly with variables of functional types in most of this thesis, beginning with Chapter 5. So we are particularly interested in terms for which all its free variables are of functional types. Such terms are called *ground-closed*. Abstract reasoning on variables of ground types will be introduced in Chapter 7 by using symbolic execution.

In the following, we write $\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau$ to distinguish between typing contexts Γ_g, Γ_f containing respectively ground type variables and functional type variables.

The relational environment associated to Γ, *i.e.* the set of (x, x, τ) such that $(x, \tau) \in \Gamma$, is written $\widetilde{\Gamma}$. The relational environment $\widetilde{\Sigma}$ is defined in the same way. From a typing context Γ and a function $\gamma : \mathrm{Var} \rightharpoonup \mathrm{Val}$, we say that γ is a *substitution* on Γ—written $\gamma : \Gamma$—if γ is defined exactly on all the variables occurring in Γ, and $\gamma(x)$ is a value of type τ whenever $(x, \tau) \in \Gamma$. Then, the action of the substitution γ on a term M, defined as $M\{\overrightarrow{\gamma(x_i)/x_i}\}$ with x_i ranging over Γ, is written $\gamma(M)$.

3.2.3 Contextual Equivalence

We now define contextual (or observational) equivalence.

$$\frac{}{\Sigma;\Gamma \vdash \Omega_\tau : \tau} \qquad \frac{}{\Sigma;\Gamma \vdash () : \text{Unit}} \qquad \frac{}{\Sigma;\Gamma \vdash \textbf{true} : \text{Bool}} \qquad \frac{}{\Sigma;\Gamma \vdash \textbf{false} : \text{Bool}}$$

$$\frac{\Sigma;\Gamma \vdash M_1 : \text{Bool} \qquad \Sigma;\Gamma \vdash M_2 : \tau \qquad \Sigma;\Gamma \vdash M_2 : \tau}{\Sigma;\Gamma \vdash \texttt{if}\ \ M_1\ \texttt{then}\ \ M_2\ \texttt{else}\ \ M_3 : \tau}$$

$$\frac{}{\Sigma;\Gamma \vdash \widehat{n} : \text{Int}} \qquad \frac{\Sigma;\Gamma \vdash M_1 : \text{Int} \qquad \Sigma;\Gamma \vdash M_2 : \text{Int}}{\Sigma;\Gamma \vdash M_1 == M_2 : \text{Bool}}$$

$$\frac{\Sigma;\Gamma \vdash M_1 : \text{Int} \qquad \Sigma;\Gamma \vdash M_2 : \text{Int}}{\Sigma;\Gamma \vdash M_1 + M_2 : \text{Int}}$$

$$\frac{(x:\tau) \in \Gamma}{\Sigma;\Gamma \vdash x : \tau} \qquad \frac{\Sigma;\Gamma, x:\tau \vdash M : \sigma}{\Sigma;\Gamma \vdash \lambda x.M : \tau \to \sigma} \qquad \frac{\Sigma;\Gamma \vdash M : \tau \to \sigma \qquad \Sigma;\Gamma \vdash N : \tau}{\Sigma;\Gamma \vdash MN : \sigma}$$

$$\frac{l \in \Sigma \cap \text{Loc}_\tau}{\Sigma;\Gamma \vdash l : \text{ref}\,\tau} \qquad \frac{\Sigma;\Gamma \vdash M_1 : \text{ref}\,\tau \qquad \Sigma;\Gamma \vdash M_2 : \text{ref}\,\tau}{\Sigma;\Gamma \vdash M_1 == M_2 : \text{Bool}}$$

$$\frac{\Sigma;\Gamma \vdash M : \text{ref}\,\tau}{\Sigma;\Gamma \vdash !M : \tau} \qquad \frac{\Sigma;\Gamma \vdash M : \text{ref}\,\tau \qquad \Sigma;\Gamma \vdash N : \tau}{\Sigma;\Gamma \vdash M := N : \text{Unit}}$$

Figure 3.9: Typing Rules of RefML

Definition 1. *Two terms M_1, M_2 of the same type τ in a context Σ, Γ, are said to be* contextually equivalent, *written $\Sigma; \Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$, when*

$$\forall h \in \mathbf{Cl}(\Sigma).\ \forall C \text{ s.t. } \Sigma; \cdot \vdash C[M_i] : \mathrm{Unit}.\Big((C[M_1], h) \Downarrow \Leftrightarrow (C[M_2], h) \Downarrow\Big).$$

where we write $(C[M_1], h) \Downarrow$ for the fact that this term reduces to $()$.

Using closed heaps containing Σ enforce the reduction of $(C[M_i], h)$ to not being stuck, *i.e.* either to always reduce to $()$ or to diverge. This is equivalent to the usual definition, which quantifies over extended location context $\Sigma' \subseteq \Sigma$ and (closed) heaps $h' : \Sigma$. We prefer here to with $\mathbf{Cl}(\Sigma)$ since it allows us to remove the quantification over $\Sigma' \subseteq \Sigma$.

When studying contextual equivalence of two terms, the notion of *disclosure* is central. Let consider the term

$$M_1 = \texttt{let x}_1 = \texttt{ ref0 in let x}_2 = \texttt{ ref0 in } \lambda \texttt{f} : (\mathrm{ref\,Int} \to \mathrm{Unit}).\texttt{f x}_1; \texttt{x}_1 := \texttt{1})$$

then we say that M_1 *discloses* the location associated to x_1 to the context. Notice that M_1 is not equivalent to

$$M_2 = \texttt{let x}_1 = \texttt{ ref0 in let x}_2 = \texttt{ ref0 in } \lambda \texttt{f} : (\mathrm{ref\,Int} \to \mathrm{Unit}).\texttt{f x}_2; \texttt{x}_1 := \texttt{1})$$

even if the two terms reduces to Unit if the callback f does not diverge. This is because M_2 discloses the location associated to x_2, and that the context can check if the location disclosed via f stores 0 or 1. Such a context which discriminates the two terms is

$$\texttt{let g} = \bullet \texttt{ in g } (\lambda \texttt{y} : \mathrm{ref\,Int}.\texttt{g } (\lambda\_.\texttt{if !y == 0 then } \Omega_{\mathrm{Unit}} \texttt{ else } \mathrm{Unit}))$$

One can also consider *indirect* disclosures, when terms provide functions to callbacks which allows the context to modify the heap. For example, the following term

$$\texttt{let x} = \texttt{ ref0 in } \lambda \texttt{f} : ((\mathrm{Int} \to \mathrm{Unit}) \to \mathrm{Unit})\mathrm{ref\,Int} \to \mathrm{Unit}).\texttt{f } (\lambda \texttt{n} : \mathrm{Int}.\texttt{x} := \texttt{n})$$

which disclose to the context the setter $\lambda \texttt{n} : \mathrm{Int}.\texttt{x} := \texttt{n}$. Using it, the context can modify as it wants the location linked to x even if it has not been directly disclosed.

3.2.4 GroundML

In this thesis, we also consider GroundML, a restriction of RefML with only full ground references, *i.e.* references which can store integers or other full ground references. It is formally defined as the set of terms of RefML whose type do not contain any subtypes $\mathrm{ref}\,(\tau \to \sigma)$, and which do not contain any subterms of the form $\mathrm{ref}\,M$ with M functional.

In GroundML, we cannot define diverging terms anymore via higher-order references, so we rely on the terms Ω_τ for each type τ that always diverge. It is particularly important to have such diverging terms when studying contextual equivalence. Indeed, without it, one cannot define the contextual equivalence of M_1, M_2 as equi-termination of $C[M_1]$ and $C[M_2]$ for all (well-type) contexts C, since contexts cannot diverge anymore. We sometimes need to consider divergent-free terms of GroundML, *i.e.* terms where no Ω_τ appear, but their contextual equivalence is still defined with possibly diverging contexts.

The contextual equivalence for GroundML is coarser than the one for RefML. Indeed, contexts of GroundML have less power than the one of GroundML. This is due to the fact that contexts cannot store higher-order values, like setter or getter, disclosed by the term. So they can only used such disclosed functions in their scope. This intuition is formally expressed by the so-called "differed divergence" example, taken from [DNB12], which exhibits two terms which are equivalent in GroundML, but not in RefML:

$$\begin{cases} \texttt{let y} \;=\; \text{ref}\,\textbf{false}\;\texttt{in}\;\lambda\texttt{f.f}(\lambda\_.\texttt{y} := \textbf{true}); \texttt{if}\;!\texttt{y}\;\texttt{then}\;\Omega_{\text{Unit}}\;\texttt{else}\;() \\ \lambda\texttt{f.f}(\lambda\_.\;\Omega_{\text{Unit}}). \end{cases}$$

3.3 Nominal Reasoning

When formalizing proofs on λ-calculus, one always reasons up-to α-*equivalence*. That is, two terms which differ only w.r.t. the name of their bound variables are considered to be equivalent. The same is true for name of locations, since no pointer-arithmetic is allowed on them. To formalize the logic behind this situation, *Nominal Logic* has been introduced by A. Pitts [Pit03]. The idea is to see objects which are defined using variables or locations only up-to permutation. This is formalized using the notion of *Fraenkel-Mostowski sets*, called *nominal sets* here, which are sets equipped with a permutation actions over a set of names (which will be the set of variables or locations here).

3.3.1 Nominal Sets

Let fix a set of names $\mathbb{A}$ and consider the group of finite permutation $\mathrm{Perm}(\mathbb{A})$ of $\mathbb{A}$, i.e. the bijections π of $\mathbb{A}$ such that the set $\{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite. We write $\mathrm{id}_{\mathbb{A}}$ for the unit of this group, i.e. the identity permutation.

Definition 2. *A* ($\mathbb{A}-$)nominal set *is a set X equipped with a group action of* $\mathrm{Perm}(\mathbb{A})$*, that is an application $* : X \times \mathrm{Perm}(\mathbb{A}) \to X$ which satisfies*

$$\forall t \in X, \forall \pi_1, \pi_2 \in \mathrm{Perm}(\mathbb{A}), \pi_1 * (\pi_2 * t) = (\pi_1 \circ \pi_2) * t \qquad \textit{and} \qquad \mathrm{id}_{\mathbb{A}} * t = t.$$

We omit to indicate $\mathbb{A}$ when it is clear from the context. A subset S of $\mathbb{A}$ is said to *support* an element t of a nominal set X when

$$\forall \pi \in \mathrm{Perm}(\mathbb{A}).(\forall a \in S.\pi(a) = a) \Rightarrow \pi * t = t.$$

The smallest subset of $\mathbb{A}$ which support t is called *the support of* t, written $\nu_{\mathbb{A}}(t)$. Terms of RefML can be seen as a nominal set over both Loc and Var. Then, the support of a term is

— its set of locations if it is seen as nominal over Loc,

— its set of free variables if it is seen as nominal over Var.

Two elements t, u of a ($\mathbb{A}$-) nominal set X are said to be *nominally-equivalent*, written $t \sim_{\mathbb{A}} u$ if there exists $\pi \in \mathrm{Perm}(\mathbb{A})$ such that $t = \pi * u$ holds. A subset Y of a nominal set X is $\mathbb{A}$-*nominal closed* if for all $t \in Y$ and $t' \sim_{\mathbb{A}} t$, $t' \in Y$.

The set of names we consider, like Loc or Var, are in fact decomposed with respect to types. That is, we can write $\mathbb{A}$ as $\biguplus_\tau \mathbb{A}_\tau$. So, we restrict our attention to finite permutations which conserve types. It means that taking $\pi \in \mathrm{Perm}(\mathcal{A})$ and $a \in \mathbb{A}_\tau$, we impose that $\pi(a) \in \mathbb{A}_\tau$. We can easily check that type-preserving finite permutations still form a group. Such type-preserving finite permutations can in fact be decomposed as finite permutations over each $\mathcal{A}_\tau$, as done in the PhD thesis of Nikos Tzevelekos [Tze08].

3.3.2 Spans

We sometimes need to be explicit in the permutation when working with two nominally equivalent elements t, u of a nominal set X. However, when this is the case, it is more convenient to work with (typed) *spans* rather than (typed) permutations because they are easier to extend than permutations. Spans have indeed already been used by Stark to reason on locations, when defining logical relations for the ν-calculus [Sta98].

Definition 3. *A* typed span $\mathcal{S} : \mathbb{A} \rightleftharpoons \mathbb{A}$ *is a collection of pairs of partial finite injections* $\mathbb{A}_\tau \hookleftarrow \mathcal{S}_\tau \hookrightarrow \mathbb{A}_\tau$ *for each type* τ.

Thus, a span is a particular kind of relation on names; it is a partial finite bijection on $\mathbb{A}$. As a set, one can see it in two ways:

— as a collection of sets $\mathcal{S}_\tau \subseteq \mathbb{A}_\tau \times \mathbb{A}_\tau$ s.t.
for all $(a_1, a_2), (a'_1, a'_2) \in \mathcal{S}_\tau. ((a_1 = a'_1) \Rightarrow (a_2 = a'_2))$ and $((a_2 = a'_2) \Rightarrow (a_1 = a'_1))$

— or as a set $\mathcal{S} \subseteq \mathbb{A} \times \mathbb{A} \times \text{Types}$ s.t.
for all $(a_1, a_2, \tau), (a'_1, a'_2, \tau') \in \mathcal{S}_\tau. ((a_1 = a'_1) \Rightarrow ((a_2 = a'_2) \wedge (\tau = \tau'))$ and
$((a_2 = a'_2) \Rightarrow (a_1 = a'_1) \wedge (\tau = \tau'))$.

We often naviguates between these two representations.

We write ε for the empty span (*i.e.* the empty set). The image of a span $\mathcal{S}$ by the left (resp. right) injection is written $\mathcal{S}_1$ (resp. $\mathcal{S}_2$). Then, we write the extension of a span $\mathcal{S}$ at type τ with $(a_1, a_2) \in \mathbb{A}_\tau$, as $\mathcal{S} \cdot (a_1, a_2, \tau)$ (the type τ could in fact be omitted in this notation), when $a_1 \notin \mathcal{S}_{1,\tau}$ and $a_2 \notin \mathcal{S}_{2,\tau}$. We say that $\mathcal{S}'$ extends $\mathcal{S}$, written $\mathcal{S}' \sqsupseteq \mathcal{S}$, when $\mathcal{S}'$ is a span which includes $\mathcal{S}$ as a set.

A span $\mathcal{S}$ induces a finite permutation $\pi_\mathcal{S} : \mathbb{A} \to \mathbb{A}$, using the so-called "Homogeneity Lemma" of [Pit03] (Lemma 1.14). Then, we define a restriction of the nominal equivalence $\sim_\mathbb{A}$ with respect to a span $\mathcal{S}$, written $X \sim_\mathcal{S} Y$, if there exists a permutation π extending $\pi_\mathcal{S}$ such that $X = \pi * Y$.

Reciprocally; from a finite permutation $\pi_\mathcal{S} : \mathbb{A} \to \mathbb{A}$, one can define a span formed by the $(a, \pi(a), \tau)$ for all τ and $a \in \mathbb{A}$ s.t. $a \neq \pi(a)$.

4

Forcing in Type Theory

Forcing is a method originally designed by Cohen to prove the independence of the Continuum Hypothesis from the axiomatic set theory ZFC [CD66]. The main idea is to add new objects which can be *approximated* in the ground system, using what is called *forcing conditions*. More precisely, a ground model M is extended to a new model M[G] by adding a new *generic* element G to M. As M[G] is in general really complicated, P. Cohen has proposed to control the true propositions in M[G] by translating them into M. To do so, he has used *forcing conditions*, which have to be seen as approximations of G. Such forcing conditions live in M while it is not the case for the generic element G. Thus, from a formula φ of M[G], the idea is to build syntactically a formula $p \Vdash \hat{\varphi}$—pronounced "p *forces* $\hat{\varphi}$"—that lives in M, and such that φ will be true when there is a "correct" approximation p of G such that $p \Vdash \hat{\varphi}$ in M. One key property of the set of forcing conditions is that they are ordered. Intuitively, we have $p \leq q$ when p is a more precise approximation of G than q, i.e. contains more information, so the relation $\Vdash$ has to be monotonic for this order.

During the past few years, forcing has received increasing attention as a way to extend the Curry-Howard correspondence to various classical reasoning. Indeed, forcing was generalized by Krivine in his work on *classical realizability* [Kri09], where forcing conditions where taken as λ-terms. He has then combined this approach with usual forcing in [Kri11] to give a computational content of various weak form of the axiom of choice, like the existence of an ultrafilter over $\mathbb{N}$ or of a well-order on $\mathbb{R}$. Following this work, A. Miquel [Miq11] has studied the proof transformation induced by the forcing, in the theory of higher-order arithmetic seen as an extension of System F_ω. That is, he has investigated how a proof of a proposition P is transformed into a proof of $p \Vdash P$.

In a constructive setting, we have used forcing in collaboration with T. Coquand [CJ10, CJ12], to give a computational content to a result of uniform continuity of definable functionals in Martin-Löf Type Theory. In this work, we have extended MLTT with a

typing judgment indexed by a forcing condition. To do so, we have developed a simple forcing translation for MLTT, fitted only for the terms we wanted to add to our theory. This leaves open the question of how to perform a complete translation for all the terms of MLTT.

To adapt the "set-theoretic" ideas of forcing to a dependant type theory, we have use its categorical restatement that Lawvere and Tierney [Tie72] have pursued using *sheaves topos*. In this setting, intuitionistic forcing is defined using the notion presheaves in category theory.

The starting point of this chapter is to connect these two observations:

"Intuitionistic forcing for type theory is an internalization of the presheaf construction in type theory."

The idea is to extend an initial type theory—called the *ground system*—with new principles, getting a new type theory—called the *forcing layer*. Terms and types of the forcing layer can be translated to the ground system using the *forcing translation*. In this way, we can develop a new generation of logics, that can be defined modularly using forcing layers. The forcing translation relies on an internalization inside $\mathrm{MLTT}_{\mathcal{U}}$ of the presheaf construction on a particular type $\mathcal{P}$—representing *forcing conditions*. Then, it becomes possible to exhibit new reasoning principles inside a forcing layer by using the structure of the chosen forcing conditions. But, no matter what new logical principles have been defined, their consistency can be deduced for free:

"The consistency of a logic defined in a forcing layer ensues from the consistency of the ground logic."

Indeed, we are able to extend a type theory with new reasoning principles and new objects, without defining them as axioms. Besides consistency problems, avoiding the axiomatic approach enables us to give a computational content to these new principles: programs can be associated to them.

On a connected line of work on Kripke semantics, Appel, Melliès, Richards and Vouillon [AMRV07] have proposed to understand step-indexing—a technique to handle general recursion in programing language semantics—as (Kripke) forcing on the set of natural numbers. Those natural numbers can be used to define (or force) a particular modality in the logic, an idea already seen by Nakano in [Nak00], with an induction principle directly lifted from natural numbers. More recently, Birkedal, Møgelberg, Schwinghammer, and Støvring [BMSS11] have shown that this construction can be understood semantically as a mean to work inside the topos of trees, which provide a generic way to define general recursive types in a semantical model. Similarly, we use forcing on the set of natural numbers to provide general recursive types in $\mathrm{MLTT}_{\mathcal{U}}$, without relying on a positivity condition. This construction makes it possible to define a universal type $\mathcal{D}$ for terms of the pure λ-calculus that induces a shallow embedding of the pure λ-calculus into $\mathrm{MLTT}_{\mathcal{U}}$. The fact that we can use a conversion rule that is normalizing to describe the β-reduction

of the pure λ-calculus should not appear as a contradiction. Unfolding of the recursive type $\mathcal{D}$ is handled by a propositional equality and not by the conversion rule and so has to be explicit in the term. As another example, we rephrase Cohen's construction of a model negating the continuum hypothesis in our framework, using the finite subsets of $\mathbf{P}(\mathbf{P}(\mathbf{Nat})) \times \mathbf{Nat}$ as the set of forcing conditions.

Plan of the Chapter In Section 4.1, we explain the forcing translation in terms of the presheaf construction. Then, we define it formally in Section 4.2, dealing with various coherence problems caused by the conversion rule. In Section 4.3, we present a systematic way to introduce new reasoning principles in a forcing layer. In Section 4.4, we illustrate this translation by choosing the natural numbers as the poset of forcing conditions, providing a framework for guarded recursive types, which can be seen as a syntactic presentation of the *topos of tree* [BMSS11]. Then, in Section 4.5, we illustrate the forcing translation with another poset of forcing conditions to force the negation of the continuum hypothesis. Finally, we discuss related works and possible future works in Section 4.6.

4.1 Internalizing the Presheaf Construction in $\mathrm{MLTT}^{e}_{\mathcal{U}}$

In this section, we explain the intuitionistic forcing translation $[-]$ of the forcing layer for a poset $(\mathcal{P}, \preccurlyeq)$ of forcing conditions. The full definition is given in Section 4.2. This translation can be seen as an internalization inside $\mathrm{MLTT}_{\mathcal{U}}$ of the presheaf construction on $\mathcal{P}$. We refer the reader to [MLM92] for the definition of the presheaf construction.

Actually, we do not interpret $\mathrm{MLTT}_{\mathcal{U}}$, but rather the version with explicit coercions $\mathrm{MLTT}^{e}_{\mathcal{U}}$ + UIP (Section 3.1.7). This is due to the fact that the forcing translation does not preserve substitution on the nose but only up-to a propositional equality. The fact that we use UIP is not crucial in our translation but it simplifies it a lot, for instance enabling the use of subset types. A discussion on the points where UIP is used is given in Section 4.2.6.

4.1.1 Forcing conditions

The forcing translation is defined on a poset $\mathcal{P}$ of forcing condition of type $\mathrm{Set}_{\mathcal{U}}$ equipped with the preorder relation $\preccurlyeq_{\mathcal{P}}$.

More precisely, we first define a type $\mathbf{PreOrder}_X$ as

$$\{f : X \to X \to \mathrm{Prop}_{\mathcal{U}} \mid \mathbf{ReflBinRel}_{X,f}, \mathbf{TransBinRel}_{X,f}\}$$

where $\mathbf{ReflBinRel}_{X,f}$ is defined as $\Pi x : X.f\ x\ x$ and $\mathbf{TransBinRel}_{X,f}$ is defined as $\Pi x, y, z : X.f\ x\ y \wedge f\ y\ z \to f\ x\ z$. So we can consider $\langle \mathcal{P}, \preccurlyeq_{\mathcal{P}} \rangle$ of type $\Sigma X : \mathcal{U}_i.\mathbf{PreOrder}_X$. We refer to the proof of transitivity of $\preccurlyeq_{\mathcal{P}}$ as $(\preccurlyeq_{\mathcal{P}}).\mathbf{TransBinRel}$. When $\mathcal{P}$ is clear from the context, we write $\preccurlyeq$ for $\preccurlyeq_{\mathcal{P}}$.

The structure of preorder enables us to define the poset $\mathcal{P}_p$ of forcing conditions that are below p as

$$\mathcal{P}_p = \{q : \mathcal{P} \mid q \preccurlyeq p\}$$

Because $\mathcal{P}$ is a poset, there is an injection $\iota_{p,q}$ from $\mathcal{P}_q$ to $\mathcal{P}_p$ as soon as there is a proof $M_{p,q} : q \preccurlyeq p$. This injection is defined formally, when $\mathcal{P}_p$ and $\mathcal{P}_q$ are seen as depedent sum, as

$$\iota_{p,q} = \lambda \mathbf{r} : \mathcal{P}_q.\langle \pi_1\ \mathbf{r}, (\preccurlyeq).\mathbf{TransBinRel}\ p\ q\ r\ M_{p,q}\ (\pi_2\ \mathbf{r})\rangle.$$

This chain of preorders enables us to construct presheaves on $\mathcal{P}$ by approximation. For the sake of simplicity, these injections are used implicitly in the rest of the chapter. They can be made explicit but it complicates a lot the reading of the translation.

4.1.2 Presheaf approximations as dependent sums

In Category Theory, a presheaf P on $\mathcal{P}$ in Set is given by a family $(P_p)_{p\in\mathcal{P}}$ of sets together with restriction maps

$$P_q \xleftarrow{\theta_{p,q}} P_p$$

for all $q \preccurlyeq p$, satisfying the usual commutative diagrams ensuring the naturality of those maps. In the special setting of a preorder, the naturality corresponds to the reflexivity and transitivity of restriction maps.

This restriction maps can be formalized using a dependent sum and the naturality conditions can be imposed using a subset type rejecting ill-formed restriction maps. Thus, the type $\mathbf{PSh}(p,\mathcal{U})$ of a presheaf at level p on a universe $\mathcal{U}$ can be defined as

$$\Sigma f : \mathcal{P}_p \to s.\{\theta : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.f\ q \to f\ r \mid \mathbf{trans}_\theta(f,p), \mathbf{refl}_\theta(f,p)\}$$

where $\mathbf{trans}_\theta(f,p)$ and $\mathbf{refl}_\theta(f,p)$ are defined as

$$\begin{array}{lcl} \mathbf{refl}_\theta(f,p) & \stackrel{def}{=} & \Pi q : \mathcal{P}_p.\Pi x : (fq).(\theta\ q\ q\ x) =_{(Tq)} x \\ \mathbf{trans}_\theta(f,p) & \stackrel{def}{=} & \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.\Pi s : \mathcal{P}_r.\Pi x : (fq).(\theta\ r\ s)(\theta\ q\ r\ x) =_{(fs)} \theta\ q\ s\ x \end{array}$$

Given a (closed) type $T : \mathcal{U}$, we introduce two notations to extract the support and the restriction maps of the associated presheaf $[T]_p$:

$$\begin{array}{lcl} [\![T]\!]_p & \stackrel{def}{=} & (\pi_1[T]_p)p \\ \theta^T_{p\to q} & \stackrel{def}{=} & (\pi_2[T]_p)pq \end{array}$$

4.1.3 Presheaf approximation of variables

As we internalize the presheaf construction directly in $\mathrm{MLTT}_{\mathcal{U}}$, we have to translate variables of the calculus, which is not the case for purely semantic definitions. The problem

with variables is that they can be used for a presheaf approximation that is smaller than the presheaf approximation for which they have been defined. For instance, this situation typically amounts to derive the following judgment, for $q \preccurlyeq p$,

$$[\Gamma], x : [\![T]\!]_p \vdash x : [\![T]\!]_q$$

which differs from the usual VAR rule. This means that the translation of a variable must introduce the right restriction map to go from the presheaf approximation at level p to the presheaf approximation at level q. For that purpose, we need to parametrize the translation of a term T with a *forcing environment* σ that associates the type and level of approximation to every free variable occurring in T. More precisely, in what follow, σ denotes an ordered list of triples $(x_1, T_1, p_1), \ldots, (x_n, T_n, p_n)$ where x_i is a variable of type T_i, whose associate forcing condition is p_i. We write $\sigma_1(x)$ (resp. $\sigma_2(x)$) for the type (resp. forcing condition) assigned to x by σ. Given a context Γ, we say that σ is a adequate environment of Γ if it assigns the same variable to the same type, and all conditions appearing in σ are either fresh, are equal to the previous one. The translation of Γ will then enforce that the underlying set of forcing conditions is ordered. So taking $\Gamma = (x_1 : T_1), \ldots, (x_n : T_n)$, an adequate environment σ for Γ is defined as $(x_1, T_1, p_1), \ldots, (x_n, T_n, p_n)$ such that p_i is fresh or $p_i = p_{i-1}$. This notion of *forcing environment* is formally defined in Section 4.2.1 Given a adequate environment σ, the translation of a variable is given by

$$[x]^\sigma_p \stackrel{def}{=} \theta^{\sigma,\sigma_1(x)}_{\sigma_2(x)\to p} x$$

and the translation of rule VAR becomes

$$[\Gamma]^\sigma, x : [\![T]\!]^\sigma_p \vdash \theta^{\sigma,T}_{p\to q} x : [\![T]\!]^\sigma_q$$

which is now derivable from the rules VAR and APP. Note that, inductively, the definitions of the support and restriction maps of the presheaf approximation are also annotated with the environment σ.

4.1.4 Presheaf approximation of dependent products

The category of presheaves is cartesian closed. This suggests that dependent products can be translated as presheaf approximations. In category theory, the internal hom $[-,-]$ is described by

$$[T, U]_p \cong \mathbf{Hom}_{\mathbf{PSh}}(y(p) \times T, U)$$

where y denotes the Yoneda embedding. This means that $[T, U]_p$ is itself a presheaf that associates to any forcing condition q a morphism

$$f_q : \mathcal{P}(q, p) \times T_q \to U_q.$$

But in our case, $\mathcal{P}$ is a preorder so f_q exists only when $q \preccurlyeq p$. A dependent product will thus be translated at level p as a family of dependent product indexed by forcing

conditions that are below p. As it is the case for morphisms of presheaves, dependent functions between presheaf approximations also have to commute with restriction maps as given by the following categorical commutative diagram

$$\begin{array}{ccc} [\![T]\!]^{\sigma}_{p} & \xrightarrow{f_p} & [\![U]\!]^{\sigma}_{p} \\ {\scriptstyle \theta^{\sigma,T}_{p\to q}}\downarrow & & \downarrow{\scriptstyle \theta^{\sigma,U}_{p\to q}} \\ [\![T]\!]^{\sigma}_{q} & \xrightarrow[f_q]{} & [\![U]\!]^{\sigma}_{q} \end{array} \tag{4.1}$$

To this end, the support of the presheaf approximation at level p is given by the following subset type

$$[\![\Pi x : T.U]\!]^{\sigma}_{p} \stackrel{def}{=} \{f : \Pi q : \mathcal{P}_p.\Pi x : [\![T]\!]^{\sigma}_{q}.[\![U]\!]^{\sigma\cdot(x,T,q)}_{q} \mid \mathbf{comm}^{\sigma}_{\Pi}(T,U,p,f)\}$$

where the first component is a family of dependent product indexed by a forcing condition below p and where

$$\mathbf{comm}^{\sigma}_{\Pi}(T,U,p,f) \stackrel{def}{=} \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.\Pi x : [\![T]\!]^{\sigma}_{q}.(fr(\theta^{\sigma,T}_{q\to r}x)) = \theta^{\sigma\cdot(x,T,q),U}_{q\to r}(fqx)$$

reflects the categorical commutative diagram (4.1).

The restriction maps are simply given by identity coercions

$$\theta^{\sigma,\Pi x:T.U}_{p\to q} \stackrel{def}{=} \lambda f : [\![\Pi x : T.U]\!]^{\sigma}_{p}.\lambda r : \mathcal{P}_q.fr$$

The translation of a function is given by

$$[\lambda x : T.M]^{\sigma}_{p} \stackrel{def}{=} \lambda q : \mathcal{P}_p.\lambda x : [\![T]\!]^{\sigma}_{q}.[M]^{\sigma\cdot(x,T,q)}_{q}$$

The proof that $[\lambda x : T.M]^{\sigma}_{p}$ validates the commutation condition is deduced from the set of equalities on restriction maps.

The translation of an application is more complicated. Because substitution does not commute with the translation on the nose. That is, the following term

$$\downarrow^{\sigma,p}_{x,N} : [\![U]\!]^{\sigma\cdot(x,T,p)}_{p}\left\{[N]^{\sigma}_{p}/x\right\} \to [\![U\,\{N/x\}]\!]^{\sigma}_{p}$$

is not given by the identity function but by an explicit transport of a propositional equality. Thus, the translation of an application is obtained by applying the translated argument $[N]^{\sigma}_{p}$ to the translated function $[M]^{\sigma}_{p}$ taken at level p, and then applying $\downarrow^{\sigma,p}_{x,N}$ to get the expected type:

$$[\mathbf{App}_{(x:T)U}(M,N)]^{\sigma}_{p} \stackrel{def}{=} \downarrow^{\sigma,p}_{x,N} (\mathbf{App}_{(x:[\![T]\!]^{\sigma}_{p})[\![U]\!]^{\sigma}_{p}}([M]^{\sigma}_{p}\ p, [N]^{\sigma}_{p})).$$

Indeed, without $\downarrow^{\sigma,p}_{x,N}$ (which is defined in Section 4.2.2), the naive translation of the application as $\mathbf{App}_{(x:[\![T]\!]^{\sigma}_{p})[\![U]\!]^{\sigma}_{p}}([M]^{\sigma}_{p}\ p, [N]^{\sigma}_{p})$ would be of type $[\![U\,\{N/x\}]\!]^{\sigma}_{p}$, which would be problematic for the soundness of the forcing translation.

Note that the fact that we need an explicit rewriting in the definition of application corresponds to the fact that, in $\mathrm{MLTT}^{e}_{\mathcal{U}}$, the rule for application makes use of a coercion in the type of the application, as shown in [CGH14]. Also this coercion can be considered to be the identity in $\mathrm{MLTT}^{e}_{\mathcal{U}}$, it is no longer the case in the forcing translation.

4.1.5 Presheaf approximation of universes

MLTT$_{\mathcal{U}}$ has a hierarchy of universes. This means that the $\mathcal{U}_i$ have to be themselves translated as presheaf approximation at level p. This is done by defining a term $\mathbf{PShC}(p, \mathcal{U}_i)$ that encodes the restriction map available on $\mathbf{PSh}(p, \mathcal{U}_i)$.

$$\begin{array}{lcl} [\mathcal{U}_i]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.\mathbf{PSh}(q, \mathcal{U}_i), \mathbf{PShC}(p, \mathcal{U}_i)) \\ \mathbf{PShC}(p, \mathcal{U}_i) & \stackrel{def}{=} & \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : \mathbf{PSh}(q, \mathcal{U}_i). \\ & & \quad (\lambda s : \mathcal{P}_r.(\pi_1 f)s, \lambda s : \mathcal{P}_r.\lambda t : \mathcal{P}_s.\lambda x : (\pi_1 f)s.(\pi_2 f)stx) \end{array}$$

4.1.6 Presheaf approximation of dependent sums

The category of presheaves has products defined pointwise. This suggests that dependent sums (whose categorical interpretations is a product) can be translated as presheaf approximation pointwisely, and likely for the associated operators.

$$\begin{array}{lcl} [\![\Sigma x : T.U]\!]^\sigma_p & \stackrel{def}{=} & \Sigma x : [\![T]\!]^\sigma_p.[\![U]\!]^{\sigma\cdot(x,T,p)}_p \\ [\langle M, N\rangle_{(x:T)U}]^\sigma_p & \stackrel{def}{=} & \langle [M]^\sigma_p, \downarrow^{\sigma,p}_{x,M} \ [N]^\sigma_p \rangle_{(x:[\![T]\!]^\sigma_p)[\![U]\!]^\sigma_p} \\ [\pi_1 M]^\sigma_p & \stackrel{def}{=} & \pi_1 [M]^\sigma_p \\ [\pi_2 M]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{x,\pi_1 M} (\pi_2 [M]^\sigma_p) \end{array}$$

Here, we need to use $\uparrow^{\sigma,p}_{x,M}$, the inverse of $\downarrow^{\sigma,p}_{x,M}$, of type $[\![U\ \{M/x\}]\!]^\sigma_p \to [\![U]\!]^{\sigma\cdot(x,T,p)}_p \left\{[M]^\sigma_p/x\right\}$ in the translation of $\langle M, N\rangle_{(x:T)U}$ to ensure that the translation of the second component has the right type. This term is given by a transport of the inverse propositional equality as the one used for $\downarrow^{\sigma,p}_{x,M}$ (again, all this is defined in Section 4.2.2). Note again that the fact that we need an explicit rewriting corresponds to the fact in MLTT$^e_{\mathcal{U}}$, the rule for dependent sums makes use of coercion in [CGH14].

To translate the strong elimination of dependent sum $\mathbf{ind}^\Sigma_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, M)$, we need to be careful about the bounded terms z in V and x, y in M. Thus, the translation $[\mathbf{ind}^\Sigma_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, M)]^\sigma_p$ is defined as

$$\uparrow^{\sigma,p}_{z,M} \left(\mathbf{ind}^\Sigma_{(z:[\![\Sigma x:T.U]\!]^\sigma_p)}[V]^{\sigma\cdot(z,\Sigma x:T.U,p)}_p, {}_{(b_1)} \downarrow^{\sigma^2,p}_{z,\langle x,y\rangle} [N]^{\sigma^2}_r, [M]^\sigma_p)\right)$$

where $\sigma^1 = \sigma \cdot (x, T, p), \sigma^2 = \sigma^1 \cdot (y, U, p)$ and ${}_{(b_1)} = {}_{(x:[\![T]\!]^\sigma_p, y:[\![U]\!]^{\sigma^1}_p)}$.

4.2 The Forcing Translation

In this section, we present the formal definition of the forcing translation and prove its soundness. But before that, let us see an example of the translation following the

presentation of the previous section. Let consider id the polymorphic identity function $\lambda\alpha : \mathcal{U}.\lambda x : \alpha.x$. Then taking $p : \mathcal{P}$, we have

$$\begin{array}{rcl}
[\mathrm{id}]^{\varepsilon}_{p} & = & \lambda q : \mathcal{P}_p.\lambda\alpha : [\![\mathcal{U}]\!]^{\varepsilon}_{q}.\lambda r : \mathcal{P}_q.\lambda x : [\![\alpha]\!]^{(\alpha,\mathcal{U},q)}_{q}.[x]^{(\alpha,\mathcal{U},q)\cdot(x,\alpha,r)}_{r} \\
& = & \lambda q : \mathcal{P}_p.\lambda\alpha : [\![\mathcal{U}]\!]^{\varepsilon}_{q}.\lambda r : \mathcal{P}_q.\lambda x : [\![\alpha]\!]^{(\alpha,\mathcal{U},q)}_{q}.\theta^{\alpha,(\alpha,\mathcal{U},q)\cdot(x,\alpha,r)}_{r\to r}x \\
& = & \lambda q : \mathcal{P}_p.\lambda\alpha : [\![\mathcal{U}]\!]^{\varepsilon}_{q}.\lambda r : \mathcal{P}_q.\lambda x : ((\pi_1(\theta^{\mathcal{U},\varepsilon}_{p\to q}\alpha))r).((\pi_2(\theta^{\mathcal{U},\varepsilon}_{q\to r}\alpha))r\ r)x
\end{array}$$

Then, suppose we have a variable β of type $[\![\mathcal{U}]\!]^{\varepsilon}_{p_\beta}$, the translation of id β is:

$$\begin{array}{rcl}
[\mathrm{id}\ \beta]^{\sigma}_{p} & = & [\mathrm{id}]^{\varepsilon}_{p}\ p\ [\beta]^{\sigma}_{p} \\
& = & [\mathrm{id}]^{\varepsilon}_{p}\ p\ (\theta^{\mathcal{U},\sigma}_{p_\beta\to p}\beta) \\
& = & \lambda r : \mathcal{P}_p.\lambda x : ((\pi_1(\theta^{\mathcal{U},\varepsilon}_{p\to r}(\theta^{\mathcal{U},\sigma}_{p_\beta\to p}\beta)))r).((\pi_2(\theta^{\mathcal{U},\varepsilon}_{p\to r}(\theta^{\mathcal{U},\sigma}_{p_\beta\to p}\beta)))r\ r)x
\end{array}$$

But β is convertible to $\lambda x : \beta.x$, whose translation is :

$$\begin{array}{rcl}
[\lambda x : \beta.x]^{\sigma}_{p} & = & \lambda r : \mathcal{P}_p.\lambda x : [\![\beta]\!]^{\sigma}_{r}.[x]^{\sigma\cdot(x,\beta,r)}_{r} \\
& = & \lambda r : \mathcal{P}_p.\lambda x : ((\pi_1[\beta]^{\sigma}_{r})r).\theta^{\beta,\sigma}_{r\to r}x \\
& = & \lambda r : \mathcal{P}_p.\lambda x : ((\pi_1(\theta^{\mathcal{U},\varepsilon}_{p_\beta\to r}\beta)r).((\pi_2[\beta]^{\sigma}_{r})r\ r)x \\
& = & \lambda r : \mathcal{P}_p.\lambda x : ((\pi_1(\theta^{\mathcal{U},\varepsilon}_{p_\beta\to r}\beta)r).((\pi_2\theta^{\mathcal{U},\varepsilon}_{p_\beta\to r}\beta)r\ r)x
\end{array}$$

As we can see, the two are equal only up-to the transitivity of $\theta^{\mathcal{U},\varepsilon}$. This means that in general, from $M_1 \equiv M_2 : T$, there is no reason that $[M_1]^{\sigma}_{p} \equiv [M_2]^{\sigma}_{p} : [\![T]\!]^{\sigma}_{p}$, but rather that we can define a term $\mathbf{convEq}^{\sigma,p}_{T,U}$ of type $[M_1]^{\sigma}_{p} =_{[\![T]\!]^{\sigma}_{p}} [M_2]^{\sigma}_{p}$. Notice also that the definition of the forcing translation presented before makes use of the coercions $\uparrow^{\sigma,p}_{x,N}$ and $\downarrow^{\sigma,p}_{x,N}$ which are, as we will see in Section 4.2.2, defined on proofs of typing judgment and not only on the structure of terms. So even if the presentation of the translation could give the feeling that it is defined on the structure of terms, this is a not actually the case. It is rather defined on typing judgment derivations. All of this justifies the use of explicit conversion coercions, as presented in Section 3.1.7.

4.2.1 Definition of the Translation

A *forcing environment* σ is defined as an ordered list of triples (x, T, p) formed by a variable x, a type T and a variable forcing condition p. The empty environment is written ε, and the addition of a triple (x, T, p) at the end of σ is written $\sigma \cdot (x, T, p)$. We say that σ is *valid* if :

— $\sigma = \varepsilon$,

— or $\sigma = \sigma' \cdot (x, T, p)$ s.t. σ' is a valid environment, x does not appear in σ, the free variables of T appears in σ and either p does not appear in σ or p is equal to the last forcing condition of σ'.

Taking a context $\Gamma = (x_1 : T_1), \ldots, (x_n : T_n)$, σ is said to be *adequate* for Γ if it is valid and is equal to $(x_1, T_1, p_1) \ldots (x_n, T_n, p_n)$, where the forcing condition variables $p_1, \ldots, p_n$ can be possibly equal. When they are all distinct, we say that σ is *canonical* for Γ.

$$
\begin{array}{lcl}
[\![T]\!]^\sigma_p & \stackrel{def}{=} & (\pi_1[T]^\sigma_p)p \\
\theta^{\sigma,T}_{p\to q} & \stackrel{def}{=} & (\pi_2[T]^\sigma_p)pq \\
\\
[x]^\sigma_p & \stackrel{def}{=} & \theta^{\sigma,\sigma_1(x)}_{\sigma_2(x)\to p}x \\
[\mathbf{c}_{T,U}(M)]^\sigma_p & \stackrel{def}{=} & \mathbf{transport}^{\lambda T.T}_{\mathcal{U}}\mathbf{convEq}^{\sigma,p}_{T,U}[M]^\sigma_p \\
\\
\mathbf{refl}_\theta(T,p) & \stackrel{def}{=} & \Pi q:\mathcal{P}_p.\Pi x:(Tq).(\theta\ q\ q\ x) =_{(Tq)} x \\
\mathbf{trans}_\theta(T,p) & \stackrel{def}{=} & \Pi q:\mathcal{P}_p.\Pi r:\mathcal{P}_q.\Pi s:\mathcal{P}_r.\Pi x:(Tq).(\theta\ r\ s)(\theta\ q\ r\ x) =_{(Ts)} \theta\ q\ s\ x \\
\mathbf{PSh}(p,\mathcal{U}_i) & \stackrel{def}{=} & \Sigma f:\mathcal{P}_p\to\mathcal{U}_i.\{\theta:\Pi q:\mathcal{P}_p.\Pi r:\mathcal{P}_q.fq\to fr \mid \mathbf{trans}_\theta(f,p), \\
 & & \qquad\qquad \mathbf{refl}_\theta(f,p)\} \\
\mathbf{PShC}(p,\mathcal{U}_i) & \stackrel{def}{=} & \lambda q:\mathcal{P}_p.\lambda r:\mathcal{P}_q.\lambda f:\mathbf{PSh}(q,\mathcal{U}_i). \\
 & & \quad (\lambda s:\mathcal{P}_r.(\pi_1 f)s, \lambda s:\mathcal{P}_r.\lambda t:\mathcal{P}_s.\lambda x:(\pi_1 f)s.(\pi_2 f)stx) \\
[\mathcal{U}_i]^\sigma_p & \stackrel{def}{=} & (\lambda q:\mathcal{P}_p.\mathbf{PSh}(q,\mathcal{U}_i), \mathbf{PShC}(p,\mathcal{U}_i)) \\
\\
\mathbf{comm}^\sigma_\Pi(T,U,p,f) & \stackrel{def}{=} & \Pi q:\mathcal{P}_p.\Pi r:\mathcal{P}_q.\Pi a:[\![T]\!]^\sigma_q.(fr)(\theta^{\sigma,T}_{q\to r}a) = \theta^{\sigma\cdot(a,T,q),U\{a/x\}}_{q\to r}(fqa) \\
\\
[\Pi x:T.U]^\sigma_p & \stackrel{def}{=} & (\lambda q:\mathcal{P}_p.\{f:\Pi r:\mathcal{P}_q.\Pi x:[\![T]\!]^\sigma_r.[\![U]\!]^{\sigma\cdot(x,T,r)}_r \mid \mathbf{comm}^\sigma_\Pi(T,U,q,f)\}, \\
 & & \quad \lambda q:\mathcal{P}_p.\lambda r:\mathcal{P}_q.\lambda f:[\![\Pi x:T.U]\!]^\sigma_q.\lambda s:\mathcal{P}_r.fs) \\
[\lambda x:T.M]^\sigma_p & \stackrel{def}{=} & \lambda q:\mathcal{P}_p.\lambda x:[\![T]\!]^\sigma_q.[M]^{\sigma\cdot(x,T,q)}_q \\
[\mathbf{App}_{(x:T)U}(M,N)]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{x,N}(\mathbf{App}_{(x:[\![T]\!]^\sigma_p)[\![U]\!]^\sigma_p}([M]^\sigma_p\ p, [N]^\sigma_p)) \\
\\
[\Sigma x:T.U]^\sigma_p & \stackrel{def}{=} & (\lambda q:\mathcal{P}_p.\Sigma x:[\![T]\!]^\sigma_q.[\![U]\!]^{\sigma\cdot(x,T,q)}_q, \\
 & & \quad \lambda q:\mathcal{P}_p.\lambda r:\mathcal{P}_q.\lambda f:\Sigma x:[\![T]\!]^\sigma_q.[\![U]\!]^{\sigma\cdot(x,T,q)}_q.(\theta^{\sigma,T}_{q\to r}(\pi_1 f), \theta^{\sigma\cdot(x,T,q),U}_{q\to r}(\pi_2 f))) \\
[\langle M,N\rangle_{(x:T)U}]^\sigma_p & \stackrel{def}{=} & \langle [M]^\sigma_p, \downarrow^{\sigma,p}_{x,M}\ [N]^\sigma_p\rangle_{(x:[\![T]\!]^\sigma_p)[\![U]\!]^\sigma_p} \\
[\pi_1 M]^\sigma_p & \stackrel{def}{=} & \pi_1[M]^\sigma_p \\
[\pi_2 M]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{x,(\pi_1 M)}(\pi_2[M]^\sigma_p) \\
[\mathbf{ind}^\Sigma_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, M]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{z,M}\left(\mathbf{ind}^\Sigma_{(b_1)}[V]^{\sigma\cdot(z,\Sigma x:T.U,p)}_p, {}_{(b_2)}\downarrow^{\sigma^2,p}_{z,\langle x,y\rangle}[N]^{\sigma^2}_p, [M]^\sigma_p\right) \\
\multicolumn{3}{l}{\quad \text{where } {}_{(b_1)} = {}_{(z:\Sigma x:[\![T]\!]^\sigma_p.[\![U]\!]^{\sigma^1}_p)}, {}_{(b_2)} = {}_{(x:[\![T]\!]^\sigma_p, y:[\![U]\!]^{\sigma^1}_p)} \text{ and } \sigma^1 = \sigma\cdot(x,T,p), \sigma^2 = \sigma^1\cdot(y,U,p)}
\end{array}
$$

Figure 4.1: Definition of the Forcing Translation (I/II)

$$
\begin{array}{lcl}
[M =_T N]^\sigma_p & \stackrel{def}{=} & \Big(\lambda q : \mathcal{P}_p.[M]^\sigma_q =_{[\![T]\!]^\sigma_q} [N]^\sigma_q, \\
 & & \quad \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda u : ([M]^\sigma_q =_{[\![T]\!]^\sigma_q} [N]^\sigma_q).\mathbf{ap}_{\theta^{\sigma,T}_{q\to r}}[M]^\sigma_q\ [N]^\sigma_q\ u\Big) \\
[\mathbf{refl}_M]^\sigma_p & \stackrel{def}{=} & \mathbf{refl}_{[M]^\sigma_p} \\
[\mathbf{ind}^=_{(b)}V, {}_{(w:T)}M, N_1, N_2, u)]^\sigma_p & \stackrel{def}{=} & \\
\multicolumn{3}{c}{\uparrow^{\sigma,p}_{w,u}\uparrow^{\sigma,p}_{y,N_2}\uparrow^{\sigma,p}_{x,N_1} \mathbf{ind}^=_{(b')}[V]^{\sigma^1}_s, \downarrow^{\sigma,p}_{w,u}\downarrow^{\sigma,p}_{y,N_2}\downarrow^{\sigma,p}_{x,N_1}\ {}_{(w:[\![T]\!]^\sigma_p)}[M]^{\sigma\cdot(w,T,p)}_p, [N_1]^\sigma_p, [N_2]^\sigma_p, [u]^\sigma_p)} \\
\multicolumn{3}{c}{\text{where } {}_{(b)} = {}_{(x,y:T,z:x=_T y)},\ {}_{(b')} = {}_{(x:[\![T]\!]^\sigma_p, y:[\![T]\!]^\sigma_p z:[x]^\sigma_p =_{[\![T]\!]^\sigma_s}[y]^\sigma_p)}}
\end{array}
$$

$$
\begin{array}{lcl}
[\mathbf{0}]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.\mathbf{0}, \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda u : \mathbf{0}.u) \\
[\mathbf{ind^0}({}_{(z:\mathbf{0})}V, N)]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{z,N} \Big(\mathbf{ind^0}({}_{(z:\mathbf{0})}[V]^{\sigma\cdot(z,\mathbf{0},p)}_p, [N]^\sigma_p)\Big)
\end{array}
$$

$$
\begin{array}{lcl}
[\mathbf{1}]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.\mathbf{1}, \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda u : \mathbf{1}.u) \\
[\star]^\sigma_p & \stackrel{def}{=} & \star \\
[\mathbf{ind^1}({}_{(z:\mathbf{0})}V, M, N)]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{z,N} \Big(\mathbf{ind^1}({}_{(z:\mathbf{1})}[V]^{\sigma\cdot(z,\mathbf{1},p)}_p, [N]^\sigma_p, [M]^\sigma_p)\Big)
\end{array}
$$

$$
\begin{array}{lcl}
[T+U]^\sigma_p & \stackrel{def}{=} & (\lambda q : \mathcal{P}_p.[T]^\sigma_q + [U]^\sigma_q, \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\mathbf{lift}^+_{([\![T]\!]^\sigma_q, [\![U]\!]^\sigma_q)}(\theta^{\sigma,T}_{q\to r}, \theta^{\sigma,U}_{q\to r})) \\
[\mathbf{inl}(M)]^\sigma_p & \stackrel{def}{=} & \mathbf{inl}([M]^\sigma_p) \\
[\mathbf{inr}(M)]^\sigma_p & \stackrel{def}{=} & \mathbf{inr}([M]^\sigma_p) \\
[\mathbf{ind}^+({}_{(z:T+U)}V, {}_{(x:T)}M_l, {}_{(y:U)}M_r, N)]^\sigma_p & \stackrel{def}{=} & \uparrow^{\sigma,p}_{z,N} \mathbf{ind}^+({}_{(z:[\![T]\!]^\sigma_p + [\![U]\!]^\sigma_p)}[V]^{\sigma^1}_p, M'_l, M'_r, [N]^\sigma_p) \\
\multicolumn{3}{c}{\text{where } M'_l = {}_{(x:[\![T]\!]^\sigma_p)} \downarrow^{\sigma^l,p}_{z,\mathbf{inl}(x)} [M_l]^{\sigma^l}_p \text{ and } \sigma^l = \sigma\cdot(x,T,p)} \\
\multicolumn{3}{c}{M'_r = {}_{(y:[\![U]\!]^\sigma_p)} \downarrow^{\sigma^r,p}_{z,\mathbf{inl}(y)} [M_r]^{\sigma^r}_p \text{ and } \sigma^r = \sigma\cdot(y,U,p)}
\end{array}
$$

Figure 4.2: Definition of the Forcing Translation (II/II)

Given an adequate environment σ for Γ, we define the translation of Γ at σ, written $[\Gamma]^\sigma$, as the context $p : \mathcal{P}, \Gamma[\sigma]^p$, where $\Gamma[\sigma]^p$ is defined as

— $\diamond[\sigma]^p \overset{def}{=} \diamond$

— $(x : T), \Gamma[\sigma]^p \overset{def}{=} (q : \mathcal{P}_p), (x : [\![T]\!]^\sigma_q), [\Gamma]^\sigma q$ if (x, T, q) is in σ and $p \neq q$,

— $(x : T), \Gamma[\sigma]^p \overset{def}{=} (x : [\![T]\!]^\sigma_p), [\Gamma]^\sigma p$ if (x, T, p) is in σ.

Note that when (x, T, p) is in σ, we do not want to reassign p in the context, since it has already been declared.

Figure 4.1 presents the forcing translation of $\mathrm{MLTT}^e_{\mathcal{U}}$. This translation has largely been explained in Section 4.1. The only remaining point concerns the special translation of explicit coercions (see Section 4.2.2) and the management of rewriting for substitutions (see Section 4.2.2).

Figure 4.2 presents the translation for the Identity Type, the Empty type, the Unit type and coproducts. The translation is simply given by considering them as constant presheaves. One only has to be careful in the use of explicit coercions in the translation of elimination rules.

4.2.2 Substitution as a propositional equality

As we have seen in Section 4.1, we need some coercions $\uparrow^{\sigma,p}_{x,N}$ and $\downarrow^{\sigma,p}_{x,N}$ to make the translation commute with substitutions. Those coercions are syntactic sugar for the following terms $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}$ and $\overline{\mathbf{subst}}^{\sigma,p}_{(x:T)U,N}$. So for every terms N and types T, U s.t. $\Gamma, x : T \vdash_e U : \mathcal{U}_i$ and $\Gamma \vdash_e N : T$, we have to define the two following terms :

— $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}$ of type $[\![U\,\{N/x\}]\!]^\sigma_p \to [\![U]\!]^{\sigma\cdot(x,T,p)}_p \left\{[N]^\sigma_p/x\right\}$

— $\overline{\mathbf{subst}}^{\sigma,p}_{(x:T)U,N}$ of type $[\![U]\!]^{\sigma\cdot(x,T,p)}_p \left\{[N]^\sigma_p/x\right\} \to [\![U\,\{N/x\}]\!]^\sigma_p$

Their definitions follows from the existence of a general term $\mathbf{substEq}^{\sigma,p}_{(x:T)U,N,\mathcal{U}_i}$ of type

$$\Pi q : \mathcal{P}_p.[U\,\{N/x\}]^\sigma_q =_{[\![\mathcal{U}_i]\!]^\sigma_p} [U]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

We can then define them as rewriting with this equality:

— $\mathbf{subst}^{\sigma,p}_{(x:T)U,N} \overset{def}{=} \mathbf{transport}^{\lambda T.T}_{\mathcal{U}}(\mathbf{ap}_{\lambda T.(\pi_1 T)p}\ \mathbf{substEq}^{\sigma,p}_{(x:T)U,N,\mathcal{U}_i})$,

— $\mathbf{subst}^{\sigma,p}_{(x:T)U,N} \overset{def}{=} \mathbf{transport}^{\lambda T.T}_{\mathcal{U}}(\mathbf{ap}_{\lambda T.(\pi_1 T)p}\ (\mathbf{substEq}^{\sigma,p}_{(x:T)U,N,\mathcal{U}_i})^{-1})$,

Before defining them, we can first state the following lemma, which allows us to rewrite $\uparrow^{\sigma,p}_{x,N}\downarrow^{\sigma,p}_{x,N} M$ into M.

Lemma 1: *The identity type* $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}(\overline{\mathbf{subst}}^{\sigma,p}_{(x:T)U,N}[M]^\sigma_p) =_{[\![U\{N/x\}]\!]^\sigma_p} [M]^\sigma_p$ *is inhabited.*

The term $\mathbf{substEq}^{\sigma,p}_{(x:T)U,N,\mathcal{U}_i}$ is defined by induction on the two typing derivations $\Gamma, x : T \vdash_e U : \mathcal{U}_i$ and $\Gamma \vdash_e N : T$. One of the most interesting case is when $U = x$, Then the equality is $[N]^\sigma_q = \theta^{\sigma,T}_{p\to q}[N]^\sigma_p$, i.e. the monotonicity of the translation. As we have said

before, this monotonicity comes from the transitivity of $\theta^{\sigma,T}_{p\to q}$ when N is a variable, and from the commutation property when N is a function.

Another, more problematic, case is when $U = U'M$ is an application such that $\Gamma, x : T \vdash_e U' : T' \to \mathcal{U}_i$ and $\Gamma, x : T \vdash_e M : T'$. Then, we want to use the induction hypothesis on $\Gamma, x : T \vdash_e M : T'$, however, a priori $T' \neq \mathcal{U}$. This means that we have to generalize the construction of **subst** to terms M whose type is not necessarily equal to an universe $\mathcal{U}$. A first try would be the following term $\mathbf{substEq}^{\sigma,p}_{(x:T)M,N,U}$ of type

$$\Pi q : \mathcal{P}_p.[M\,\{N/x\}]^\sigma_q =_{[\![U]\!]^\sigma_p} [M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

with $\Gamma, x : T \vdash_e M : U, x \notin \mathrm{FV}(U)$ and $\Gamma \vdash_e N : T$. The hypothesis $x \notin \mathrm{FV}(U)$ is necessary so that $[M\,\{N/x\}]^\sigma_q$ and $[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$ have the same type $[\![U]\!]^\sigma_p$. However, here again, when trying to build these terms by induction on the proof of $\Gamma, x : T \vdash_e M : U$, when M is an application, we have again to generalize the construction to cases where $x \in \mathrm{FV}(U)$.

To do so, the type of $\mathbf{substEq}^{\sigma,p}_{(x:T)M,N,U}$ uses $\mathbf{substEq}^{\sigma,p}_{(x:T)U,N,\mathcal{U}_i}$ via $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}$, i.e. it is defined as

$$\Pi q : \mathcal{P}_p.[M\,\{N/x\}]^\sigma_q =_{[\![U\{N/x\}]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(x:T)U,N}[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

when $\Gamma, x : T \vdash_e U : \mathcal{U}_i$ and $\Gamma \vdash_e N : T$.

The definition of $\mathbf{substEq}^{\sigma,p}_{(x:T)M,N,U}$ is even more complicated as it involves properties on the restriction maps that also need to be defined inductively on the typing derivation. To formalize those mutually recursive definitions, we will proceed as in the Coq proof assistant by first stating the type of the defined terms and the given the meaning of the term by a proof, by induction on typing derivations.

Definition 4. *Suppose that*
- $\Gamma, x : T \vdash_e U : \mathcal{U}_i$,
- $\Gamma, x : T \vdash_e M : U$,
- $\Gamma \vdash_e N : T$,
- *and σ is a adequate environment for Γ.*

Then, we can define the following terms :
- $\mathbf{monoTrad}^{\sigma,p}_{N,T}$ *of type*

$$\Pi q : \mathcal{P}_p.\theta^{\sigma,T}_{p\to q}[N]^\sigma_p =_{[\![T]\!]^\sigma_q} [N]^\sigma_q$$

- $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_i,U,N}$ *of type*

$$\Pi q : \mathcal{P}_p.[U\,\{N/x\}]^\sigma_q =_{[\![\mathcal{U}_i]\!]^\sigma_p} [U]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

— $\mathbf{substEq}^{\sigma,p}_{(x:T)U,M,N}$ *of type*

$$\Pi q : \mathcal{P}_p.[M\,\{N/x\}]^\sigma_q =_{[\![U\{N/x\}]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(x:T)U,N}[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

— $\mathbf{commute}^{\sigma,p}_{(x:T)U,M,N}$ *of type*

$$\begin{aligned}\Pi q : \mathcal{P}_p.\quad & \theta^{\sigma,T}_{p\to q}(\mathbf{subst}^{\sigma,p}_{(x:T)U,N}([M]^{\sigma\cdot(x,T,p)}_p \left\{[N]^\sigma_p/x\right\})) =_{[\![U\{N/x\}]\!]^\sigma_q} \\ & \mathbf{subst}^{\sigma,q}_{(x:T)U,N}([M]^{\sigma\cdot(x,T,q)}_q \left\{\theta^{\sigma,T}_{p\to q}[N]^\sigma_p/x\right\})\end{aligned}$$

Proof We reason by induction on the proofs of $\Gamma, x : T \vdash_e U : \mathcal{U}_i$, $\Gamma, x : T \vdash_e M : U$. and $\Gamma \vdash_e N : T$.

First, we build $\mathbf{substEq}^{\sigma,p}_{(x:T)U,M,N}$ and $\mathbf{commute}^{\sigma,p}_{(x:T)U,M,N}$.

Univ: If $M = \mathcal{U}_k$, then $U = \mathcal{U}_j$ and $k < j < i$.

Moreover, $x \notin \mathrm{FV}(U)$ so $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_i,U,N} = \lambda q : \mathcal{P}_P.\mathbf{refl}_{[U]^\sigma_p}$. This means that:

— $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$ is convertible to $[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$

— $\mathbf{subst}^{\sigma,q}_{(x:T)U,N}([M]^{\sigma\cdot(x,T,p)}_q \left\{\theta^{\sigma,T}_{p\to q}[N]^\sigma_p/x\right\})$ is convertible to $[M]^{\sigma\cdot(x,T,p)}_q \left\{\theta^{\sigma,T}_{p\to q}[N]^\sigma_p/x\right\}$

and since $x \notin \mathrm{FV}(M)$, they are both convertible to $[M]^\sigma_q$. Then, we can define

— $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_j,\mathcal{U}_k,N} \stackrel{def}{=} \lambda q : \mathcal{P}_P.\mathbf{refl}_{[U]^\sigma_p}$.

— $\mathbf{commute}^{\sigma,p}_{(x:T)\mathcal{U}_j,\mathcal{U}_k,N} \stackrel{def}{=} \mathbf{monoTrad}^{\sigma,p}_{\mathcal{U}_k,\mathcal{U}_j}$

0-Univ, 1-Univ, Nat-Univ: The same reasoning applies when M is equal to the types $\mathbf{0}, \mathbf{1}$ or $\mathbf{Nat}$.

Π-Univ: If $M = \Pi y : T_1.T_2$, then $U = \mathcal{U}_j$ with $j < i$. Then again $x \notin \mathrm{FV}(U)$ so $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_i,U,N} = \lambda q : \mathcal{P}_P.\mathbf{refl}_{[U]^\sigma_p}$. so

— $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$ is convertible to $[M]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$

— $\mathbf{subst}^{\sigma,q}_{(x:T)U,N}([M]^{\sigma\cdot(x,T,p)}_q \left\{\theta^{\sigma,T}_{p\to q}[N]^\sigma_p/x\right\})$ is convertible to $[M]^{\sigma\cdot(x,T,p)}_q \left\{\theta^{\sigma,T}_{p\to q}[N]^\sigma_p/x\right\}$

but now x is possibly free in M.

However, we have $\Gamma, x : T \vdash_e T_1 : \mathcal{U}_{j_1}$ and $\Gamma, x : T, y : T_1 \vdash_e T_2 : \mathcal{U}_{j_2}$ with $j = max(j_1, j_2)$. Thus, by induction hypothesis, we get the existence of $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_{j_1},T_1,N}$ and $\mathbf{substEq}^{\sigma\cdot(y,T_1,p),p}_{(x:T)\mathcal{U}_{j_2},T_2,N}$, such that, transporting these equalities, we can build $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_j,\Pi y:T_1.T_2,N}$. The same reasoning applies for $\mathbf{commute}^{\sigma,p}_{(x:T)\mathcal{U}_j,\Pi y:T_1.T_2,N}$.

Id-Univ,Σ-Univ, +-Univ: The same reasoning applies when M is equal to the types $M_1 =_T M_2$, $\Sigma y : T_1.T_2$ or $T_1{+}T_2$.

Π-Intro: If M is an abstraction $\lambda y : T_1.M_1$ and $U = \Pi y : T_1.T_2$, then we first build $\mathbf{substEq}^{\sigma,p}_{(x:T)\Pi y:T_1.T_2,(\lambda y:T_1.M_1),N}$ of type

$$\begin{array}{l}\Pi q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda y : (\llbracket T_1\,\{N/x\}\rrbracket^\sigma_r).[M_1\,\{N/x\}]^\sigma_r =_{\llbracket U\rrbracket^\sigma_q} \\ \quad \mathbf{subst}^{\sigma,q}_{(x:T)\Pi y:T_1.T_2,N}\lambda r : \mathcal{P}_q.\lambda y : (\llbracket T_1\rrbracket^\sigma_r\,\{[N]^\sigma_r/x\}).[M_1]^{\sigma'}_r\,\{[N]^\sigma_r/x\}\end{array}$$

Then, we have $\Gamma, x : T \vdash_e T_1 : \mathcal{U}_j$ and $\Gamma, x : T, y : T_1 \vdash_e M_1 : T_2$ so by induction hypothesis we get

— $\mathbf{substEq}^{\sigma,p}_{(x:T)\mathcal{U}_j,T_1,N}$ of type

$$\Pi q : \mathcal{P}_p.[T_1\,\{N/x\}]^\sigma_q =_{\llbracket\mathcal{U}_j\rrbracket^\sigma_q} [T_1]^{\sigma\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

— $\mathbf{substEq}^{\sigma\cdot(y,T_1,p),p}_{(x:T)T_2,M_1,N}$ of type

$$\Pi q : \mathcal{P}_p.[M_1\,\{N/x\}]^\sigma_q =_{\llbracket T_2\{N/x\}\rrbracket^\sigma_q} \mathbf{subst}^{\sigma,q}_{(x:T)T_2,N}[M_1]^{\sigma'\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$$

Π-Elim: If M is an application $\mathbf{App}_{(y:T_1)T_2}(M_1, M_2)$ with $U = T_2\,\{M_2/y\}$ then we first build $\mathbf{substEq}^{\sigma,p}_{(x:T)U,\mathbf{App}_{(y:T_1)T_2}(M_1,M_2),N}$ of type

$$\begin{array}{ll}\Pi q : \mathcal{P}_p. & [(\mathbf{App}_{(y:T_1)T_2}(M_1, M_2))\,\{N/x\}]^\sigma_q =_{\llbracket U\{N/x\}\rrbracket^\sigma_q} \\ & \mathbf{subst}^{\sigma,q}_{(x:T)U,N}[\mathbf{App}_{(y:T_1)T_2}(M_1, M_2)]^{\sigma'\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}\end{array}$$

We have $[(\mathbf{App}_{(y:T_1)T_2}(M_1, M_2))\,\{N/x\}]^\sigma_q$ equal to

$$\mathbf{subst}^{\sigma,q}_{(y:(T_1\{N/x\}))(T_2\{N/x\}),(M_2\{N/x\})}[(M_1\,\{N/x\})]^\sigma_q\ q\ [M_2\,\{N/x\}]^\sigma_q$$

and $[\mathbf{App}_{(y:T_1)T_2}(M_1, M_2)]^{\sigma'\cdot(x,T,p)}_q \left\{[N]^\sigma_p/x\right\}$ equal to

$$(\mathbf{subst}^{\sigma\cdot(x,T,p),q}_{(y:T_1)T_2,M_2}) \left\{[N]^\sigma_q/x\right\} ([M_1]^{\sigma\cdot(y,T_1,q)}_q) \left\{[N]^\sigma_q/x\right\}\ q\ ([M_2]^{\sigma\cdot(y,T_1,q)}_q) \left\{[N]^\sigma_q/x\right\}$$

Using the induction hypothesis we get a proof of equality between:
— $[(M_1\,\{N/x\})]^\sigma_q$ and $[M_1]^{\sigma\cdot(y,T_1,q)}_q) \left\{[N]^\sigma_q/x\right\}$,
— $[M_2\,\{N/x\}]^\sigma_q$ and $[M_2]^{\sigma\cdot(y,T_1,q)}_q) \left\{[N]^\sigma_q/x\right\}$.
And using UIP, we have a proof of equality between

$$\mathbf{subst}^{\sigma,q}_{(y:(T_1\{N/x\}))(T_2\{N/x\}),(M_2\{N/x\})} \text{ and } \left(\mathbf{subst}^{\sigma\cdot(x,T,p),q}_{(y:T_1)T_2,M_2}\right) \left\{[N]^\sigma_q/x\right\}.$$

Var: If M is a variable y, then
— either $y \neq x$ in which case $\mathbf{substEq}^{\sigma,p}_{(x:T)U,y,N}$ and $\mathbf{commute}^{\sigma,p}_{(x:T)U,y,N}$ are simply defined as $\lambda q : \mathcal{P}_P.\mathbf{refl}_{[y]^\sigma_q}$,
— or $y = x$ and $T = U$, such that $x \notin \mathrm{FV}(U)$. Then, $\mathbf{subst}^{\sigma,p}_{(x:T)U,N}$ disappears, so $\mathbf{substEq}^{\sigma,p}_{(x:T)U,x,N}$ is of type

$$\Pi q : \mathcal{P}_p.[N]^\sigma_q =_{[\![U]\!]^\sigma_q} \theta^{\sigma,T}_{p\to q}[N]^\sigma_p$$

which is exactly $\mathbf{monoTrad}^{\sigma,p}_{N,T}$. Moreover, $\mathbf{commute}^{\sigma,p}_{(x:T)U,x,N}$ is of type

$$\Pi q : \mathcal{P}_p.\theta^{\sigma,T}_{p\to q}(\theta^{\sigma,T}_{p\to p}[N]^\sigma_p) =_{[\![U]\!]^\sigma_q} \theta^{\sigma,T}_{q\to q}(\theta^{\sigma,T}_{p\to q}[N]^\sigma_p)$$

so we build it using reflexivity of $\theta^{\sigma,T}_{q\to q}$ and $\theta^{\sigma,T}_{p\to p}$.

Second, we build $\mathbf{monoTrad}^{\sigma,p}_{N,T}$ by induction on $\Gamma \vdash_e N : T$.

Var: If N is a variable x such that $\sigma_2(x) = p_x$, then we have to prove

$$\Pi q : \mathcal{P}_p.\theta^{\sigma,T}_{p\to q}(\theta^{\sigma,T}_{p_x\to p}x) =_{[\![T]\!]^\sigma_q} \theta^{\sigma,T}_{p_x\to q}x$$

which directly comes from transitivity of $\theta^{\sigma,T}$.

Π-Intro: If N is a function $\lambda x : T'.M$ and $T = \Pi x : T'.U$, we have to prove

$$\Pi q : \mathcal{P}_p.\theta^{\sigma,\Pi x:T'.U}_{p\to q}(\lambda r : \mathcal{P}_p.\lambda x : [\![T']\!]^\sigma_r.M) =_{[\![\Pi x:T'.U]\!]^\sigma_q} (\lambda s : \mathcal{P}_q.\lambda x : [\![T']\!]^\sigma_s.M).$$

But $\theta^{\sigma,\Pi x:T'.U}_{p\to q}$ is equal to $\lambda f : (\Pi r : \mathcal{P}_p.\Pi x : [\![T]\!]^\sigma_r.[\![U]\!]^{\sigma\cdot(x,T,r)}_r).\lambda s : \mathcal{P}_q.fs$ so these terms are directly convertible.

Π-Elim: If N is an application $\mathbf{App}_{(y:T_1)T_2}(M_1, M_2)$ then we have $\Gamma, x : T \vdash_e M_1 : \Pi y : T_1.T_2$ and $\Gamma, x : T \vdash_e M_1 : T_1$ s.t. $T = T_2\{M_2/y\}$. We have to prove

$$\Pi q : \mathcal{P}_p.\theta^{\sigma,T}_{p\to q}(\mathbf{subst}^{\sigma,p}_{(y:T_1)T_2,M_2}([M_1]^\sigma_p\ p[M_2]^\sigma_p)) =_{[\![T]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(y:T_1)T_2,M_2}([M_1]^\sigma_q\ q[M_2]^\sigma_q).$$

Taking $q : \mathcal{P}_q$, by induction, we have $\mathbf{monoTrad}^{\sigma,p}_{M_2,T_1}$ of type $\Pi q : \mathcal{P}_p : \theta^{\sigma,T_1}_{p\to q}[M_2]^\sigma_p =_{[\![T_1]\!]^\sigma_q} [M_2]^\sigma_q$. so transporting this equality to the one we want to prove, we still have to show that

$$\theta^{\sigma,T}_{p\to q}(\mathbf{subst}^{\sigma,p}_{(y:T_1)T_2,M_2}([M_1]^\sigma_p\ p\ [M_2]^\sigma_p)) =_{[\![T]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(y:T_1)T_2,M_2}([M_1]^\sigma_q\ q\ \theta^{\sigma,T_1}_{p\to q}[M_2]^\sigma_p).$$

Then, we reason by induction on $\Gamma, x : T \vdash_e M_1 : \Pi y : T_1.T_2$.
— If $M_1 = \lambda y : T_1.M_1'$, then we have to prove

$$\theta^{\sigma,T}_{p\to q}(\mathbf{subst}^{\sigma,p}_{(y:T_1)T_2,M_2}([M_1']^\sigma_p \left\{[M_2]^\sigma_p/y\right\})) =_{[\![T]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(y:T_1)T_2,M_2}([M_1']^\sigma_q \left\{\theta^{\sigma,T_1}_{p\to q}[M_2]^\sigma_p/y\right\}).$$

which is exactly $\mathbf{commute}^{\sigma,p}_{(x:T)U,M_1',N}$.

— If $M_1 = z$ such that $(z, \Pi y : T_1.T_2, p_z)$ is in σ, we have to prove

$$\theta^{\sigma,T}_{p\to q}(\mathbf{subst}^{\sigma,p}_{(y:T_1)T_2,M_2}((\theta^{\sigma,\Pi y:T_1.T_2}_{p_z\to p} z)\ p\ [M_2]^\sigma_p)) =_{[\![T]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(y:T_1)T_2,M_2}((\theta^{\sigma,\Pi y:T_1.T_2}_{p_z\to q} z)\ q\ \theta^{\sigma,T_1}_{p\to q}[M_2]^\sigma_p).$$

Then, due to the definition of $\theta^{\sigma,\Pi y:T_1.T_2}$, we just have to prove

$$\theta^{\sigma,T}_{p\to q}(\mathbf{subst}^{\sigma,p}_{(y:T_1)T_2,M_2}(z\ p\ [M_2]^\sigma_p)) =_{[\![T]\!]^\sigma_q} \mathbf{subst}^{\sigma,q}_{(y:T_1)T_2,M_2}(z\ q\ \theta^{\sigma,T_1}_{p\to q}[M_2]^\sigma_p).$$

But this come from the fact that z is of type $[\![\Pi y : T_1.T_2]\!]^\sigma_{p_z}$ so that $\mathbf{comm}^\sigma_\Pi(T_1, T_2, p_z, z)$ is inhabited. ∎

4.2.3 Translation of explicit coercions

Let M, N two terms such that $\Gamma \vdash_e M \equiv N : T$, and σ the adequate environment for Γ. Then in general, as we said before we cannot prove that $[\Gamma]^\sigma \vdash [M]^\sigma_p \equiv [N]^\sigma_p : [\![T]\!]^\sigma_p$, that is that the translated term are judgmentally equal. However, we can build a term $\mathbf{convEq}^{\sigma,p}_{M,N}$ that corresponds to the proof that they are propositionally equal. Using it, we define $[\mathbf{c}_{T,U}(M)]^\sigma_p$ as $\mathbf{transport}^{\lambda T.T}_{\mathcal{U}}\mathbf{convEq}^{\sigma,p}_{T,U}[M]^\sigma_p$ (as defined in Figure 4.1).

As the definition is done by induction on the proof of conversion, we proceed as in Section 4.2.2 and provide the explicit term as a proof.

Definition 5. *Given M, N two terms such that $\Gamma \vdash_e M \equiv N : T$, and σ the adequate environment for Γ, we can define a term $\mathbf{convEq}^{\sigma,p}_{M,N}$ of type*

$$[\Gamma]^\sigma \vdash \mathbf{convEq}^{\sigma,p}_{M,N} : [M]^\sigma_p =_{[\![T]\!]^\sigma_p} [N]^\sigma_p.$$

Proof It is build by induction on the proof of $\Gamma \vdash_e M \equiv N : T$.

Π-Conv

Suppose that $M = (\lambda x : T.M_1)M_2$ and $N = M_1\{M_2/x\}$. Then, $[(\lambda x : T.M_1)M_2]^\sigma_p$ is defined as

$$\uparrow^{p,\sigma}_{x,M_2} ((\lambda q : \mathcal{P}_p.\lambda x : [\![T]\!]^\sigma_p.[M_1]^{\sigma\cdot(x,T,p)}_p)p\ [M_2]^\sigma_p)$$

which is convertible to $\uparrow^{p,\sigma}_{x,M_2} ([M_1]^{\sigma\cdot(x,T,p)}_p)\left\{[M_2]^\sigma_p/x\right\}$

Then using $\mathbf{substEq}^{p,\sigma}_{(x:T)U,M_1,M_2}$ one can prove that it is indeed equal to $[M_1\{M_2/x\}]^\sigma_p$. So $\mathbf{convEq}^{\sigma,p}_{(\lambda x:T.M_1)N_2,M_1\{M_2/x\}}$ is defined as $\mathbf{substEq}^{p,\sigma}_{(x:T)U,M_1,M_2}\ p$.

Conv-Proj1

Suppose that $M = \pi_1\langle M_1, M_2\rangle$ and $N = M_1$. Then $[\pi_1\langle M_1, M_2\rangle]^\sigma_p$ is equal to

$$\pi_1\langle [M_1]^\sigma_p, \downarrow^{\sigma,p}_{x,M_1} [M_2]^\sigma_p\rangle$$

which is indeed convertible to $[M_1]^\sigma_p$. So $\mathbf{convEq}^{\sigma,p}_{\pi_1\langle M_1,M_2\rangle,M_1}$ is defined as $\mathbf{refl}_{M_1}$.

Conv-Proj2

Suppose that $M = \pi_2\langle M_1, M_2\rangle$ and $N = M_2$ Then $[\pi_2\langle M_1, M_2\rangle]^\sigma_p$ is equal to

$$\uparrow^{\sigma,p}_{x,(\pi_1\langle M_1,M_2\rangle)} (\pi_2\langle [M_1]^\sigma_p, \downarrow^{\sigma,p}_{x,M_1} [M_2]^\sigma_p\rangle)$$

which is convertible to $\uparrow^{\sigma,p}_{x,(\pi_1\langle M_1,M_2\rangle)} (\downarrow^{\sigma,p}_{x,M_1} [M_2]^\sigma_p)$

Then using UIP, we can prove that it is equal to $\uparrow^{\sigma,p}_{x,M_1} (\downarrow^{\sigma,p}_{x,M_1} [M_2]^\sigma_p)$. And we conclude the construction using lemma 1.

Conv-ind$^\Sigma$

$[\mathbf{ind}^\Sigma{}_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, \langle M_1, M_2\rangle)]^\sigma_p$ is defined as

$$\uparrow^{\sigma,p}_{z,\langle M_1,M_2\rangle} \left(\mathbf{ind}^\Sigma{}_{(b_1)}[V]^{\sigma\cdot(z,\Sigma x:T.U,p)}_p, {}_{(b_2)} \downarrow^{\sigma^2,p}_{z,\langle x,y\rangle} [N]^{\sigma^2}_p, \langle [M_1]^\sigma_p, \downarrow^{\sigma,p}_{x,M_1} [M_2]^\sigma_p\rangle)\right) \quad \blacksquare$$

where $(b_1) = (z:\Sigma x:[\![T]\!]^\sigma_p.[\![U]\!]^{\sigma^1}_p)$, $(b_2) = (x:[\![T]\!]^\sigma_p, y:[\![U]\!]^{\sigma^1}_p)$ and $\sigma^1 = \sigma\cdot(x,T,p), \sigma^2 = \sigma^1\cdot(y,U,p)$.
It converts to $\uparrow^{\sigma,p}_{z,\langle M_1,M_2\rangle} \left((\downarrow^{\sigma^2,p}_{z,\langle x,y\rangle} [N]^{\sigma^2}_p) \left\{[M_1]^\sigma_p/x\right\} \left\{(\downarrow^{\sigma,p}_{x,M_1} [M_2]^\sigma_p)/y\right\}\right)$
It has to be equal to $[N\{M_1/x\}\{M_2/y\}]^\sigma_p$
If it is equal to $\uparrow^{\sigma,p}_{z,\langle M_1,M_2\rangle} (\downarrow^{\sigma,p}_{z,\langle M_1,M_2\rangle} [N\{M_1/x\}\{M_2/y\}]^{\sigma^2}_p)\big)$.

4.2.4 Soundness of the Translation

Before stating the soundness of the translation, we need a weakening lemma for adequate environments.

Lemma 2: *If $\Gamma \vdash T : \mathcal{U}$ and σ is a canonical environment of Γ, then for every forcing condition p and every interpretation σ' disjoint of σ, $[T]^{\sigma\cdot\sigma'}_p = [T]^\sigma_p$.*

Proof The proof is done by induction on T. The only interesting case is when T is a variable x. Then from $\Gamma \vdash T : \mathcal{U}$ we know that $(x,\mathcal{U}) \in \Gamma$, and since σ is canonical for Γ, there exists a forcing condition q such that $(x,T,q) \in \sigma$. So we just have to prove that $\theta^{\sigma\cdot\sigma',\mathcal{U}}_{p\to q} = \theta^{\sigma,\mathcal{U}}_{p\to q}$, which follows from the definition of $\mathbf{PSh}(p,\mathcal{U})$. $\blacksquare$

Lemma 3: *Let Γ a context and σ a canonical environment of Γ. Suppose that*

$$[\Gamma]^\sigma, q : \mathcal{P}_p, x_1 : [\![T_1]\!]_q^{\sigma_1}, \ldots, x_n : [\![T_n]\!]_q^{\sigma_n} \vdash [M]_q^{\sigma_n} : [T]_q^{\sigma_n}$$

with $\sigma_1 = \sigma$, $\sigma_{i+1} = \sigma_i \cdot (x_i, T_i, q)$. Then

$$[\Gamma]^\sigma, x_1 : [\![T_1]\!]_p^{\sigma'_1}, \ldots, x_n : [\![T_n]\!]_p^{\sigma'_n} \vdash [M]_p^{\sigma'_n} : [\![T]\!]_p^{\sigma'_n}$$

where $\sigma'_i = \sigma_i \{p/q\}$.

Theorem 4 (Typing Soundness): *If $\Gamma \vdash_e M : T$ and σ is a canonical environment of Γ then*

$$[\Gamma]^\sigma \vdash_e [M]_p^\sigma : [\![T]\!]_p^\sigma$$

where p is the last forcing condition occurring in σ.

Proof The proof is done by induction on the proof of $\Gamma \vdash_e M : T$.

Univ : To prove $[\Gamma]^\sigma \vdash (\lambda q : \mathcal{P}_p.\mathbf{PSh}(q,\mathcal{U}_i), \mathbf{PShC}(p,\mathcal{U}_i)) : \mathbf{PSh}(p,\mathcal{U}_{i+1})$ we use the rule Σ-Intro with the two following proof trees :

$$\dfrac{\dfrac{\dfrac{\dfrac{[\Gamma]^\sigma, q : \mathcal{P}_p \vdash \mathcal{P}_q : \mathcal{U}_0 \qquad \overline{[\Gamma]^\sigma, q : \mathcal{P}_p \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\ \text{Univ}}{[\Gamma]^\sigma, q : \mathcal{P}_p \vdash \mathcal{P}_q \to \mathcal{U}_i : \mathcal{U}_{i+1}}\ \Pi\text{-Univ}}{[\Gamma]^\sigma, q : \mathcal{P}_p \vdash \Sigma f : \mathcal{P}_q \to \mathcal{U}_i.\{\theta : \Pi r : \mathcal{P}_q.\Pi s : \mathcal{P}_r.fr \to fs \mid \mathbf{trans}_\theta(f,q) \wedge \mathbf{refl}_\theta(f,q)\} : \mathcal{U}_{i+1}}\ \Sigma\text{-Univ}}{[\Gamma]^\sigma \vdash \underbrace{\lambda q : \mathcal{P}_p.\mathbf{PSh}(q,\mathcal{U}_i)}_{M} : \mathcal{P}_p \to \mathcal{U}_{i+1}}\ \Pi\text{-Intro}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\Gamma' \vdash (\lambda s : \mathcal{P}_r.(\pi_1 f)s, \lambda s : \mathcal{P}_r.\lambda t : \mathcal{P}_s.\lambda x : (\pi_1 f)s.(\pi_2 f)stx) : \mathbf{PSh}(r,\mathcal{U}_i)}\ \Sigma\text{-Intro}}{[\Gamma]^\sigma \vdash \mathbf{PShC}(p,\mathcal{U}_i) : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.\mathbf{PSh}(q,\mathcal{U}_i) \to \mathbf{PSh}(r,\mathcal{U}_i)}\ \Pi\text{-Intro}}{[\Gamma]^\sigma \vdash \mathbf{PShC}(p,\mathcal{U}_i) : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.Mq \to Mr.}\ \text{Conv}}{[\Gamma]^\sigma \vdash \mathbf{PShC}(p,\mathcal{U}_i) : \{\theta : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.Mq \to Mr \mid \mathbf{trans}_\theta(M,p) \wedge \mathbf{refl}_\theta(M,p)\}}\ \Sigma\text{-Intro}$$

where $\Gamma' = [\Gamma]^\sigma, q : \mathcal{P}_p, r : \mathcal{P}_q, f : \mathbf{PSh}(q,\mathcal{U}_i)$
with proof obligations :
— $\mathbf{trans}_{\mathbf{PShC}(p,\mathcal{U}_i)}(M,p)$
— $\mathbf{refl}_{\mathbf{PShC}(p,\mathcal{U}_i)}(M,p)$

Conv : We want to show that the following rule is derivable:

$$\dfrac{[\Gamma]^\sigma \vdash [M]_p^\sigma : [\![T]\!]_p^\sigma \qquad T \vdash U \equiv \mathcal{U} :}{[\Gamma]^\sigma \vdash [\mathbf{c}_{T,U}(M)]_p^\sigma : [\![U]\!]_p^\sigma}$$

which simply comes from

$$\Pi\text{-Elim}\ \dfrac{[\Gamma]^\sigma \vdash [M]_p^\sigma : [\![T]\!]_p^\sigma}{[\Gamma]^\sigma \vdash \mathbf{transport}_{\mathcal{U}}^{\lambda T.T}\mathbf{convEq}_{T,U}^{\sigma,p}[M]_p^\sigma : [\![U]\!]_p^\sigma}$$

Var :

$$\textsc{Var}\ \frac{\vdash \mathbf{wf}(x : [\![A]\!]^{\sigma}_{\sigma_2(x)}, [\Gamma]^{\sigma})}{\dfrac{x : [\![A]\!]^{\sigma}_{\sigma_2(x)}, [\Gamma]^{\sigma} \vdash_e x : [\![A]\!]^{\sigma}_{\sigma_2(x)}}{x : [\![A]\!]^{\sigma}_{\sigma_2(x)}, [\Gamma]^{\sigma} \vdash_e \theta^{\sigma,A}_{\sigma_2(x)\to p} x : [\![A]\!]^{\sigma}_{p}}}$$

Π-Univ :

To prove $[\Gamma]^{\sigma} \vdash_e [\Pi x : T.U]^{\sigma}_p : \mathbf{PSh}(p, \mathcal{U}_{max(i,j)})$ we use the rule Σ-Intro :

$$\textsc{Σ-Univ}\ \frac{\textsc{Π-Intro}\ \dfrac{\textsc{Π-Univ}\ \dfrac{\textsc{Proj-1}\ \dfrac{q : \mathcal{P}_p, r : \mathcal{P}_q, [\![\Gamma, x : T]\!]^{\sigma\cdot(x,T,r)} \vdash_e [U]^{\sigma\cdot(x,T,r)}_r : \mathbf{PSh}(r, \mathcal{U}_i)}{[\Gamma]^{\sigma}, q : \mathcal{P}_p, r : \mathcal{P}_q, x : [\![T]\!]^{\sigma}_r \vdash_e [\![U]\!]^{\sigma\cdot(x,T,r)}_r : \mathcal{U}_j}}{[\Gamma]^{\sigma}, q : \mathcal{P}_p \vdash_e \Pi r : \mathcal{P}_q.\Pi x : [\![T]\!]^{\sigma}_r.[\![U]\!]^{\sigma\cdot(x,T,r)}_r \mathcal{U}_i :}}{[\Gamma]^{\sigma}, q : \mathcal{P}_p \vdash_e \{f : \Pi r : \mathcal{P}_q.\Pi x : [\![T]\!]^{\sigma}_r.[\![U]\!]^{\sigma\cdot(x,T,r)}_r \mid \mathbf{comm}^{f}_{\Pi}(T, U, q,)\} : \mathcal{U}_i}}{[\Gamma]^{\sigma} \vdash_e \lambda q : \mathcal{P}_p.\underbrace{\{f : \Pi r : \mathcal{P}_q.\Pi x : [\![T]\!]^{\sigma}_r.[\![U]\!]^{\sigma\cdot(x,T,r)}_r \mid \mathbf{comm}^{f}_{\Pi}(T, U, q,)\}}_{M} : \mathcal{P}_p \to \mathcal{U}_i}$$

Then, we have to prove that

$$[\Gamma]^{\sigma} \vdash_e \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Pi x : T.U]\!]^{\sigma}_q.\lambda s : \mathcal{P}_r.f\ s : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.M\ q \to M\ r$$

which is straightforward from the fact that $M\ r$ is of type $[\![\Pi x : T.U]\!]^{\sigma}_r$:

$$\textsc{Π-Intro}\ \frac{\textsc{Π-Elim}\ \dfrac{\Gamma' \vdash_e f : \Pi s : \mathcal{P}_r.[\![\Pi x : T.U]\!]^{\sigma}_q \qquad \Gamma' \vdash_e s : \mathcal{P}_r}{\Gamma' \vdash_e f\ s : \Pi x : [\![T]\!]^{\sigma}_r.[\![U]\!]^{\sigma\cdot(x,T,r)}_r}}{[\Gamma]^{\sigma} \vdash_e \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Pi x : T.U]\!]^{\sigma}_q.\lambda s : \mathcal{P}_r.fs : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.M\ q \to M\ r}$$

where $\Gamma' = [\Gamma]^{\sigma}, q : \mathcal{P}_p, r : \mathcal{P}_q, f : [\![\Pi x : T.U]\!]^{\sigma}_q, s : \mathcal{P}_r$ with the proof obligations

$$\mathbf{trans}_{\theta}(\lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Pi x : T.U]\!]^{\sigma}_q.\lambda s : \mathcal{P}_r.f\ s, p)$$

and

$$\mathbf{refl}_{\theta}(\lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Pi x : T.U]\!]^{\sigma}_q.\lambda s : \mathcal{P}_r.f\ s, p)$$

Π-Intro :

$$\textsc{Σ-Intro}\ \frac{\textsc{Π-Intro}\ \dfrac{\textsc{Hyp. Ind.}\ \dfrac{-}{[\Gamma, x : A]^{\sigma\cdot(x,T,q)} \vdash_e [M]^{\sigma\cdot(x,T,q)}_q : [\![U]\!]^{\sigma\cdot(x,T,q)}_q}}{[\Gamma]^{\sigma} \vdash_e \lambda q : \mathcal{P}_p.\lambda x : [\![T]\!]^{\sigma}_q.[M]^{\sigma\cdot(x,T,q)}_q : \Pi q : \mathcal{P}_p.\Pi x : [\![T]\!]^{\sigma}_q.[\![U]\!]^{\sigma\cdot(x,T,q)}_q}}{[\Gamma]^{\sigma} \vdash_e \lambda q : \mathcal{P}_p.\lambda x : [\![T]\!]^{\sigma}_q.[M]^{\sigma\cdot(x,T,q)}_q : \{f : \Pi q : \mathcal{P}_p.\Pi x : [\![T]\!]^{\sigma}_q.[\![U]\!]^{\sigma\cdot(x,T,q)}_q \mid \mathbf{comm}^{f}_{\Pi}(T, U, q, \})}$$

with the proof obligation $\mathbf{comm}^{\lambda q:\mathcal{P}_p.\lambda x:[\![T]\!]^{\sigma}_q.[M]^{\sigma\cdot(x,T,q)}_q}_{\Pi}(T, U, q,)$ which is proved using $\mathbf{commute}^{\sigma,p}_{(x:T)U,M,N}$.

Π-Elim :

$$\Pi\text{-Elim}\ \dfrac{\Pi\text{-Elim}\ \dfrac{[\Gamma]^\sigma \vdash [M]^\sigma_p p : \Pi x : [\![T]\!]^\sigma_p.[\![U]\!]^{\sigma\cdot(x,T,p)}_p \qquad [\Gamma]^\sigma \vdash [N]^\sigma_p : [\![T]\!]^\sigma_p}{[\Gamma]^\sigma \vdash \mathbf{App}_{(x:[\![T]\!]^\sigma_p)[\![U]\!]^{\sigma\cdot(x,T,p)}_p}([M]^\sigma_p p, [N]^\sigma_p) : [\![U]\!]^{\sigma\cdot(x,T,p)}_p \left\{[N]^\sigma_p/x\right\}}}{[\Gamma]^\sigma \vdash \mathbf{subst}^{\sigma,p}_{(x:T)U,N}\Big(\mathbf{App}_{(x:[\![T]\!]^\sigma_p)[\![U]\!]^{\sigma\cdot(x,T,p)}_p}([M]^\sigma_p p, [N]^\sigma_p)\Big) : [\![U\,\{N/x\}]\!]^\sigma_p}$$

Σ-Univ : We want to prove that the following rule is derivable :

$$\dfrac{[\Gamma]^\sigma \vdash_e [T]^\sigma_p : [\![\mathcal{U}_i]\!]^\sigma_p \qquad [\Gamma, x : T]^{\sigma'} \vdash_e [U]^{\sigma'}_q : [\![\mathcal{U}_i]\!]^{\sigma'}_q}{[\Gamma]^\vdash_e [\Sigma x : T.U]^\sigma_p : [\![\mathcal{U}_i]\!]^\sigma_p}$$

where $\sigma' = \sigma\cdot(x, T, q)$. Since $[\![\mathcal{U}_i]\!]^\sigma_p = \mathbf{PSh}(p, \mathcal{U})$ which is a dependent sum, we decompose the proof in two.

$$\begin{array}{r}\textsc{Weak}\ \dfrac{[\Gamma]^\sigma \vdash_e [\![T]\!]^\sigma_p : \mathcal{U}_i}{[\Gamma]^\sigma, q : \mathcal{P}_p \vdash_e [\![T]\!]^\sigma_q : \mathcal{U}_i} \qquad [\Gamma]^\sigma, q : \mathcal{P}_p, x : [\![T]\!]^{\sigma'}_q \vdash_e [\![U]\!]^{\sigma'}_q : \mathcal{U}_i \\ \Sigma\text{-Univ}\ \dfrac{}{[\Gamma]^\sigma, q : \mathcal{P}_p \vdash_e \Sigma x : [\![T]\!]^\sigma_q.[\![U]\!]^{\sigma\cdot(x,T,q)}_q : \mathcal{U}_i} \\ \Pi\text{-Intro}\ \dfrac{}{[\Gamma]^\sigma \vdash_e \underbrace{\lambda q : \mathcal{P}_p.\Sigma x : [\![T]\!]^\sigma_q.[\![U]\!]^{\sigma\cdot(x,T,q)}_q}_{M} : \mathcal{P}_p \to \mathcal{U}_i}\end{array}$$

Then, we prove that

$$[\Gamma]^\sigma \vdash_e \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Sigma x : T.U]\!]^\sigma_q.(\theta^{\sigma,T}_{q\to r}(\pi_1 f), \theta^{\sigma',U}_{q\to r}(\pi_2 f)) : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.M\ q \to M\ r$$

the proof is similar to the one done in Π-Univ. Finally, we deal with the proof obligations

$$\mathbf{trans}_\theta(\lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Sigma x : T.U]\!]^\sigma_q.(\theta^{\sigma,T}_{q\to r}(\pi_1 f), \theta^{\sigma',U}_{q\to r}(\pi_2 f)), p)$$

and

$$\mathbf{refl}_\theta(\lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.\lambda f : [\![\Sigma x : T.U]\!]^\sigma_q.(\theta^{\sigma,T}_{q\to r}(\pi_1 f), \theta^{\sigma',U}_{q\to r}(\pi_2 f)), p)$$

Σ-Intro : We want to prove that the following rule is derivable :

$$\dfrac{[\Gamma]^\sigma \vdash_e [M]^\sigma_p : [\![T]\!]^\sigma_p \qquad [\Gamma]^\sigma \vdash_e [N]^\sigma_p : [\![U\,\{M/x\}]\!]^\sigma_p}{[\Gamma]^\sigma \vdash_e [\langle M, N\rangle_{(x:T)U}]^\sigma_p : [\![\Sigma x : T.U]\!]^\sigma_p}$$

$$\Sigma\text{-Intro}\ \dfrac{[\Gamma]^\sigma \vdash_e [M]^\sigma_p : [\![T]\!]^\sigma_p \qquad \dfrac{[\Gamma]^\sigma \vdash_e [N]^\sigma_p : [\![U\,\{M/x\}]\!]^\sigma_p}{[\Gamma]^\sigma \vdash_e \overline{\mathbf{subst}}^{\sigma,p}_{(x:T)U,M}[N]^\sigma_p : [\![U]\!]^\sigma_p \left\{[M]^\sigma_p/x\right\}}\ \Pi\text{-Elim}}{[\Gamma]^\sigma \vdash_e \langle [M]^\sigma_p, \overline{\mathbf{subst}}^{\sigma,p}_{(x:T)U,M}[N]^\sigma_p\rangle_{(x:[\![T]\!]^\sigma_p)[\![U]\!]^{\sigma'}_{p'}} : \Sigma x : [\![T]\!]^\sigma_p.[\![U]\!]^{\sigma'}_p}$$

StrongElim-Σ : We want to prove that the following rule is derivable :

$$\frac{[\Gamma, z : \Sigma x : T.U]^{\sigma^1} \vdash [V]_q^{\sigma^1} : [\![\mathcal{U}]\!]_q^{\sigma^1} \quad [\Gamma, x : T, y : U]^{\sigma^2} \vdash [N]_r^{\sigma^2} : [\![V\,\{\langle x, y\rangle/z\}]\!]_r^{\sigma^2} \quad [\Gamma]^\sigma \vdash [M]_p^\sigma : [\![\Sigma x : T.U]\!]_p^\sigma}{[\Gamma]^\sigma \vdash [\mathbf{ind}^\Sigma_{(z:\Sigma x:T.U)}V, {}_{(x:T,y:U)}N, M)]_p^\sigma : [\![V\,\{M/z\}]\!]_p^\sigma}$$

where $\sigma^1 \overset{def}{=} \sigma \cdot (z, \Sigma x : T.U, q)$ and $\sigma^2 \overset{def}{=} \sigma \cdot (x, T, q) \cdot (y, U, r)$.
Then, using Lemma 3 we get the following hypothesis
— $[\Gamma]^\sigma, z : \Sigma x : [\![T]\!]_p^\sigma.[\![U]\!]_p^{\sigma\cdot(x,T,p)}[V]_p^{\sigma^1}[\![\mathcal{U}]\!]_p^{\sigma^1}$
— $[\Gamma]^\sigma, x : [\![T]\!]_p^\sigma, y : [\![U]\!]_p^{\sigma\cdot(x,T,p} \vdash [N]_p^{\sigma\cdot(x,T,p)\cdot(y,U,p)} : [\![V\,\{\langle x, y\rangle/z\}]\!]_p^\sigma$
Then, we get

$$[\Gamma]^\sigma, x : [\![T]\!]_p^\sigma, y : [\![U]\!]_p^{\sigma\cdot(x,T,p} \vdash \underbrace{\mathbf{subst}^{\sigma^2,r}_{(z:\Sigma x:U.)V,\langle x,y\rangle}[N]_p^{\sigma\cdot(x,T,p)\cdot(y,U,p)}}_{N'} : [\![V]\!]_q^{\sigma^1}\,\{[\langle x, y\rangle]_r^{\sigma_2}/z\}$$

So, we can StrongElim-Σ to prove

$$[\Gamma]^\sigma \vdash \mathbf{ind}^\Sigma_{(z:[\![\Sigma x:T.U]\!]_p^\sigma)}[V]_p^{\sigma^1}, {}_{(x:[\![T]\!]_p^\sigma, y:[\![U]\!]_p^{\sigma\cdot(x,T,p)})}N', [M]_p^\sigma) : [\![V]\!]_p^\sigma\left\{[M]_p^\sigma/z\right\}$$ ∎

Let consider P a mere proposition. Then, the translation $[\![P]\!]_p^\sigma$ can be seen as what is usually written $p \Vdash P$ in forcing. Then, using the previous theorem, a proof M of P (*i.e.*. M is of type P) is translated into $[M]_p^\sigma$. This explain why our work can be seen as a study of the forcing translation of proofs, *i.e.* as an extension of the work of Miquel [Miq11].

4.2.5 Extending the internalization to $\mathrm{MLTT}_\mathcal{U}$

Up-to now, we have defined the translation in $\mathrm{MLTT}^e_\mathcal{U}$ because we need an explicit management of coercions. This section discusses what need to be done to extend it to $\mathrm{MLTT}_\mathcal{U}$.

As we have explained in Section 3.1.7, we assume that a judgment $\Gamma \vdash M : T$ in $\mathrm{MLTT}_\mathcal{U}$ can be annotated with explicit coercions Γ_e, M_e, T_e as given by Theorem 3. Thus, the translation of a term of M for an environment $|\sigma_e|$, where σ_e is a canonical environment for Γ_e and p is its last forcing condition, could be defined as

$$[M]_p^{|\sigma_e|} \overset{def}{=} |[M_e]_p^{\sigma_e}|.$$

But as our translation—and in particular the terms $\mathbf{substEq}^{\sigma,p}_{(x:T)U,M,N}$—is built by induction on the proof of typing judgments, the definition may depend on the chosen Γ_e, M_e, T_e. So to get a correct definition of the translation of derivation of $\mathrm{MLTT}_\mathcal{U}$, we need to prove its independence with respect to the chosen corresponding derivation in $\mathrm{MLTT}^e_\mathcal{U}$. That is, given $\Gamma_1 \vdash_e M_1 : T_1$ and $\Gamma_2 \vdash_e M_2 : T_2$ such that $|\Gamma_1| = |\Gamma_2|$,

$|M_1| = |M_2|$, $|T_1| = |T_2|$, and given σ_1 (resp. σ_2) a canonical environment for Γ_1 (resp. Γ_2) s.t. $|\sigma_1| = |\sigma_2|$, we need to show that

$$[M_1]_p^{|\sigma_1|} = [M_2]_p^{|\sigma_2|}.$$

This property is related to the so-called *coherence* problems of categorical models of dependent type theory [Cur93, Hof95b, CGH14], *i.e.* the fact that substitution, which is interpreted as pullbacks in these models, is only "pseudo-functorial", i.e. up-to isomorphism.

First, notice that when working in the *extensional* $\mathrm{MLTT}_{\mathcal{U}}$, *i.e.* with the reflection rule which identify judgmental and propositional equality, we get this uniqueness of the translation. Indeed, in such case, we do not have to introduce the various coercions $\uparrow_{x,N}^{\sigma,p}, \downarrow_{x,N}^{\sigma,p}$ since we can directly use the proofs of $\mathbf{substEq}_{(x:T)U,M,N}^{\sigma,p}$ to convert them.

In the intentional case, it is a lot more difficult. First, notice that the fact that we our working with UIP does not really help here, because we get the uniqueness of terms $\mathbf{substEq}_{(x:T)U,M,N}^{\sigma,p}$ up to propositional equality, but one cannot conclude from that fact, since the conversion rule can be used at different stage of the proof of the typing judgment.

To achieve the proof of uniqueness of the translation of this setting, it should be possible to adapt the reasoning done in [Cur93, CGH14], by going to a setting with explicit substitution.

4.2.6 About the use of UIP and the poset restriction

The use of UIP is made at two crucial points:

1. in the use of subset types in the definitions of $\mathbf{PSh}(p, \mathcal{U}_i)$ and $[\Pi x : T.U]_p^{\sigma}$,
2. to simply the definition of $\mathbf{substEq}_{(x:T)U,N,\mathcal{U}_i}^{\sigma,p}$ in Section 4.2.2 and $\mathbf{convEq}_{T,U}^{\sigma,p}$ in Section 4.2.3.

While we think that the second use could be avoided by a more careful management of equalities between equalities (as done in Homotopy Type Theory [Uni13]), the first use seems more difficult to overcome. Indeed, if we remove the UIP axiom, then the computational content of the proofs of transitivity and reflexivity of restriction maps of a presheaf type can not be ignored. It is the same for the functoriality condition in the translation of a dependent product. The consequence of that is that the terms $\mathbf{substEq}_{(x:T)U,N,\mathcal{U}_i}^{\sigma,p}$ in Section 4.2.2 and $\mathbf{convEq}_{T,U}^{\sigma,p}$ will be more involved. We do not see any theoretical reason why it should not be possible but we have left this investigation for future work.

Regarding the poset restriction, we could have also consider categories instead of just pre-orders. However, this implies that the set of forcing conditions below a forcing condition p becomes couples (q, f) of type

$$\mathcal{P}_p = \Sigma q : \mathcal{P}.q \to p$$

The explicit management of the morphism f hardens the definition, although, from categorical considerations, it should be feasible.

4.2.7 Implemeting the translation in Coq

So far, we have presented the translation for set elements of $\mathrm{MLTT}^{e}_{\mathcal{U}}$. This translation can also be implemented for CIC, the type system behind Coq, with *proof irrelevance* and explicit coercions. To do so, we first have to introduce the translation $\mathbf{PSh}(p, \mathrm{Prop})$ of the universe Prop. It is defined in the same way than $\mathcal{U}$, with the simplication that, because of Proof-Irrelevance, the restriction maps are in that case proofs of monotonicity and thus, transitivity and reflexivity of restriction maps are automatic :

$$\mathbf{PSh}(p, \mathrm{Prop}) \stackrel{def}{=} \{f : \mathcal{P}_p \to \mathrm{Prop} \mid \theta : \Pi q : \mathcal{P}_p.\Pi r : \mathcal{P}_q.f\ q \to f\ r\}$$

We also have to translate *Inductive Types*, following the same idea as the translation of **Nat** we have presented. Their constructor is simply translated as constant presheaves, while their destructors, i.e. their induction principles, have first to be η-expanded before being translated also as constant presheaves.

A prototype implementation of the translation as been implemented by Matthieu Sozeau, available at `https://github.com/mattam82/Forcing`. It is built on top of (an old version of) Coq, with a patch that introduces proof irrelevance in the conversion, translating terms of a forcing layer so that they can be verified by the typechecker of Coq. The fact that we need proof-irrelevance in the conversion and not only propositionally is because we have not considered the coercions $\iota_{p,q}$ from $\mathcal{P}_q$ to $\mathcal{P}_p$ when $q \leq p$ in the translation. But using proof-irrelevance in the conversion makes all those coercions judgmentally equal to the identity. Using it, the Forcing Layer presented in Section 4.4 has been defined in Coq allowing to use *guarded recursive types* in this proof assistant.

4.3 Reasoning in the Forcing Layer

The goal of this section is to show how to extend safely our type theory with new elements, so that well-formed terms still typecheck and the layer stays consistent (i.e. the empty type is not inhabited). To do so, we define the notion of *Forcing Layers*. To simplify the presentation, we suppose that forcing has been defined not only on $\mathrm{MLTT}^{e}_{\mathcal{U}}$ but also on $\mathrm{MLTT}_{\mathcal{U}}$.

4.3.1 Defining new constructors in the Forcing Layer

To extend the logical power of the type theory of a forcing layer, we follow a general mechanism. We first add new symbols corresponding to the objects we want to reason on. For example, in the next section we will introduce a new fixpoint combinator. We then define their translation, which is a way to give them a meaning in the original layer.

Finally, we add properties about these new elements whose translations are proven in the original layer.

More precisely, after adding a new symbol f of type T to the forcing layer, we define its translation $[f]^\sigma_p$. Then, we can add a proof-term symbol π of a lemma P about f as soon as we are able to define a proof $[\pi]^\sigma_p$ of $[\![P]\!]^\sigma_p$ in the ground logic.

This has to be compared to the axiomatic approach in Coq, which introduce a new symbol using `Parameter f:T` and then properties about it using `Axiom` π`:P`. In this approach, `f` and π have no computational content.

So, we define a forcing layer $\mathcal{F}$ as a list of new symbols $(C_1, \dots, C_n)$. The terms of $\mathcal{F}$ are generated by the grammar of $\mathrm{MLTT}_\mathcal{U}$ extended by the new symbols $C_1, \dots, C_n$. We suppose that each symbol C_i is typable in the empty context with the type T_i, which is a term of $\mathcal{F}$ such that T_i does not contains any C_j for $j < i$. We define a new typing judgment $\Gamma \vdash^\mathcal{F}_e M : T$ for contexts, terms and types of $\mathcal{F}$, defined with the same typing rules of $\mathrm{MLTT}_\mathcal{U}$ plus the axioms $\vdash^\mathcal{F}_e C_i : T_i$. This forcing layer comes with a translation $\mathcal{F}[C_i]^\varepsilon_p$ of each element C_i, which is defined in $\mathrm{MLTT}_\mathcal{U}$. We can now state when a forcing layer is well-defined.

Definition 6. *A forcing layer is* well defined *when*

$$p : \mathcal{P}_p \vdash \mathcal{F}[C_i]^\varepsilon_p : \mathcal{F}[\![T_i]\!]^\varepsilon_p$$

in the original layer and when T_i is a dependent product for every i.

Given a well defined forcing layer, it is possible to extend the definition of $\mathcal{F}[M]^\sigma_p$ for terms of $\mathcal{F}$ by induction on the structure of M, using the standard translation $[\_]$ for the constructions of $\mathrm{MLTT}_\mathcal{U}$.

The only subtle point is for the extension of the definition of terms dealing with substitution, in particular the monotonicity of the translation, captured by the term $\mathbf{monoTrad}^{\sigma,p}_{N,T}$. Here, we use the fact that every new symbol C_i inhabits dependent product T_i, which imposes by construction that the interpretation is monotone (because of the presence of $\Pi q : \mathcal{P}_p. \dots$ at the beginning of the translation of a dependent product). Note that it would not be the case on other types, for instance on **Bool**.

4.3.2 Soundness and Consistency of the Forcing Layer

We now want to extend the soundness of the translation to the forcing layer.

Theorem 5: *Given a well-defined forcing layer $\mathcal{F}$, if $\Gamma \vdash^\mathcal{F}_e M : T$, then*

$$\mathcal{F}[\Gamma]^\sigma \vdash \mathcal{F}[M]^\sigma_p : \mathcal{F}[\![T]\!]^\sigma_p.$$

PROOF This is a straightforward consequence of the soundness for $\mathrm{MLTT}_{\mathcal{U}}$, since we just have extended it with new axioms which are sound by definition of the forcing layer.

We can also prove the consistency of the forcing layer.

Theorem 6: *There is no term M such that $\diamond \vdash^{\mathcal{F}}_{e} M : \mathbf{0}$*

PROOF Suppose there exists such a term. Then, $p : \mathcal{P}_p \vdash_e \mathcal{F}[M]^{\varepsilon}_p : \mathcal{F}[\![\mathbf{0}]\!]^{\varepsilon}_p$. But $\mathcal{F}[\![\mathbf{0}]\!]^{\varepsilon}_p = [\![\mathbf{0}]\!]^{\varepsilon}_p$ which is equal to $\mathbf{0}$, so $\mathcal{F}[M]^{\varepsilon}_p$, which is defined in $\mathrm{MLTT}_{\mathcal{U}}$, is an inhabitant of $\mathbf{0}$, which is not possible by consistency of $\mathrm{MLTT}_{\mathcal{U}}$. ∎

4.3.3 Iterating the forcing translation

For now, we have defined a forcing layer on top of $\mathrm{MLTT}_{\mathcal{U}}$, but it is in fact possible to define a forcing layer on top of another forcing layer:

$$\mathrm{MLTT}_{\mathcal{U}} \xleftarrow{[.]^{\mathcal{F}_1}} \mathcal{F}_1 \xleftarrow{[.]^{\mathcal{F}_2}} \mathcal{F}_2$$

As $\mathcal{F}_1$ contains $\mathrm{MLTT}_{\mathcal{U}}$, the translation $[.]^{\mathcal{F}_2}$ is well-defined. Then, the poset of forcing conditions $\mathcal{P}_2$ used by $\mathcal{F}_2$ has to be defined in $\mathcal{F}_1$. This means in particular that we can use terms of $\mathrm{MLTT}_{\mathcal{U}}$ but also new logical principles provided by $\mathcal{F}_1$.

This construction is reminiscent of the iterated forcing construction as presented for example in [Jec03]. In terms of topos, this means to build the category of presheaves valuated in another category of presheaves.

4.3.4 Sheaf construction and excluded middle

We have seen how to import a theorem proved in $\mathrm{MLTT}_{\mathcal{U}}$ in a forcing layer, simply by using its proof-term. However, this method does not work with axioms, for good reasons: there is no reason it stays true in the forcing layer, since we do not have a proof of it.

One simple example is the excluded-middle

$$\mathbf{EM} \overset{def}{=} \Pi Q : \mathrm{Prop}.Q + (Q \to \mathbf{0})$$

Its translation $[\mathbf{EM}]^{\sigma}_p$ is equal to

$$\Pi q : \mathcal{P}_p.\Pi Q : (\mathcal{P}_q \to \mathrm{Prop}).Qq + (\Pi r : \mathcal{P}_q.Qr \to \mathbf{0}).$$

It is thus possible to build a set of forcing conditions which negate this formula, simply adapting the construction of the usual Kripke model which negates **EM**.

If we want to keep this axiom true, a standard way is to add a negative translation to the forcing translation. This is done in Topos theory by the sheafification process on the dense topology (usually noted $\neg\neg$). Indeed, it is well known [MLM92] that the topos $Sh(\mathcal{P}_{\neg\neg}, \mathcal{E})$ is a Boolean topos while this is not true for $PSh(\mathcal{P}, \mathcal{E})$.

Adapting the same technique to Type Theory is the subject of ongoing research. The sheafification process is understood as the notion of *left exact modalities* recently defined in homotopy type theory ([Uni13], §7.7).

4.4 The Step-Indexed Layer

As a first illustration of the power of the forcing translation, we study the forcing layer $\mathcal{SI}$ obtained when $\mathcal{P}$ is $(\mathbf{Nat}, \leq)$, the set of natural numbers ordered as usual. This layer (without universes) has already been semantically studied in [BMSS11] as it corresponds to the topos of trees. We have then developed in [JTS12] a syntactic presentation of it using our forcing translation. The main novelty of our approach is to define a *later* modality $\triangleright_{\mathcal{U}}$ as an endofunction on the universe $\mathcal{U}$, inside the forcing layer. This modality is used to interpret guarded recursive types as guarded recursive functions on a universe. Interestingly, when restricted to the universe of mere proposition $\mathbf{Prop}_{\mathcal{U}}$, we get the Löb rule that provides a general inductive principle in the logic. As the Löb rule is interpreted directly as a special instance of the fixpoint construction, our construction gives a computational meaning to the Löb rule inside $\mathrm{MLTT}_{\mathcal{U}}$. The semantic work of [BMSS11] has later been extended to universes [BM13] using the same techniques.

It could seem surprising to allow such general recursive definitions, without endanger the consistency of the logic. Indeed, the usual guard condition of fixpoints in type theory, namely the strict positivity, is here to avoid to build terms which could inhabit the empty type $\mathbf{0}$. However, such strict positivity condition is needed because fixpoints can be unwound with the conversion, the so-called ι-rule. Here, our unwinding of fixpoints are only defined propositionally rather than judgmentally. This is why the consistency result of any forcing layer proved in Section 4.3.2 applied to $\mathcal{SI}$ is not contradictory with the usual way inductive types are defined in $\mathrm{MLTT}_{\mathcal{U}}$. Moreover, we keep the decidability of the type-checking for the extended type theory of $\mathcal{SI}$, while we are able to formalize for example the pure λ-calculus in it.

4.4.1 Definition of $\mathcal{SI}$

As explained in Section 4.3.1, a forcing layer is defined as a poset of forcing conditions, a list of new symbols together with their types and their translation. Here, $\mathcal{SI}$ is defined as the forcing layer whose forcing conditions is $(\mathbf{Nat}, \leq)$ as defined in Section 3.1.6, with the following new symbols :

- $\triangleright_{\mathcal{U}} : \mathcal{U} \to \mathcal{U}$, the later modality which is used as a guard on types,
- $\mathbf{fix}_T : (\triangleright_{\mathcal{U}} T \to T) \to T$, the fixpoint combinator on guarded types,
- $\mathbf{next}_T : (T \to \triangleright_{\mathcal{U}} T)$
- $\mathbf{switch}_{\mathcal{U}} : \triangleright_{\mathcal{U}} \mathcal{U} \to \mathcal{U}$ together with a proof of $[\mathbf{switch}_{\mathcal{U}}(\mathbf{next}_{\mathcal{U}} T)]^{\sigma}_{p} = [\triangleright_{\mathcal{U}} T]^{\sigma}_{p}$ which allows us to transfer the modality $\triangleright$ from a universe to a type,
- $\mathbf{comlater}^{\mathbf{T},\mathbf{U}}_{\mathbf{\Pi}} : \triangleright_{\mathcal{U}_j}(\Pi x : T.U) \to (\Pi x : \triangleright_{\mathcal{U}_i} T.\, \triangleright_{\mathcal{U}_j} U)$ and $\mathbf{comlater}^{\mathbf{T},\mathbf{U}}_{\mathbf{\Sigma}} : \triangleright_{\mathcal{U}_j}(\Sigma x : T.U) \to \Sigma x : \triangleright_{\mathcal{U}_i} T.\, \triangleright_{\mathcal{U}_j} U$ which allows us to propagate the guard $\triangleright$ into dependent products and sums.
- $\mathrm{rec}_{\mu_{\mathcal{U}}} : \Pi f : \mathcal{U} \to \mathcal{U}.\mu_{\mathcal{U}} f =_{\mathcal{U}} f(\triangleright_{\mathcal{U}} \mu_{\mathcal{U}} f)$ a proof of equality between a term and its unwinding, where $\mu_{\mathcal{U}} f$ is a notation for $\mathbf{fix}_{\mathcal{U}} \lambda x.f(\mathbf{switch}_{\mathcal{U}} x)$.

We now present the translation $\mathcal{SI}[\_]$ of these new symbols, defined as terms of $\mathrm{MLTT}_{\mathcal{U}}$.

The later modality We first introduce in the $\mathcal{SI}$ layer a later modality

$$\triangleright_{\mathcal{U}} : \mathcal{U} \to \mathcal{U}$$

for any universe $\mathcal{U}$. This modality amounts to shift a presheaf on $\mathcal{U}$ one step on the right. Its translation $\mathcal{SI}[\triangleright_{\mathcal{U}}]^{\sigma}_{p}$ is defined by

$$\begin{array}{l}
\lambda q : \mathbf{Nat}_p.\lambda T : [\![\mathcal{U}]\!]^{\sigma}_{q}. \\
\quad (\lambda r : \mathbf{Nat}_q.\texttt{match } r \texttt{ with} \\
\qquad \mid 0 \texttt{ => } \mathbf{1} \\
\qquad \mid \mathbf{S}r' \texttt{ => } (\pi_1 T) r' \\
\quad , \lambda r : \mathbf{Nat}_q.\lambda t : \mathbf{Nat}_r.\lambda x : U_r.\texttt{match } t \texttt{ with} \\
\qquad \mid 0 \texttt{ => } \star \\
\qquad \mid \mathbf{S}t' \texttt{ => } (\pi_2 T)\,(\mathrm{Pred}\ r)\,t'\,x)
\end{array}$$

with $U_r \overset{def}{=} \texttt{match } r \texttt{ with} \mid 0 \texttt{ => } \mathbf{1} \mid \mathbf{S}r' \texttt{ => } (\pi_1 T)r'$. Recall the $\mathbf{1}$ is the unit type presented in Section 3.1.5, and $\star$ is its unique element. Here, $\mathbf{Nat}_p$ stands for $\{q : \mathbf{Nat} \mid q \leq p\}$. In the following, we write $\triangleright$ for $\triangleright_{\mathcal{U}}$ when the universe can be inferred easily.

Unwinding the definition of $\triangleright_{\mathcal{U}}$ applied to a type $T : \mathcal{U}$ of $\mathrm{MLTT}_{\mathcal{U}}$, we have

— $\mathcal{SI}[\![\triangleright T]\!]^{\sigma}_{0} = \mathbf{1}$ and $\mathcal{SI}[\![\triangleright T]\!]^{\sigma}_{\mathbf{S}p} = [\![T]\!]^{\sigma}_{p}$,

— $\theta^{\sigma,\triangleright T}_{0\to 0} = \lambda x : \mathbf{1}.\star$, and $\theta^{\sigma,\triangleright T}_{(\mathbf{S}p)\to 0} = \lambda x : [\![T]\!]^{\sigma}_{p}.\star$,

— $\theta^{\sigma,\triangleright T}_{(\mathbf{S}p)\to(\mathbf{S}q)} = \theta^{\sigma,T}_{p\to q}$.

So for example $\mathcal{SI}[\![\triangleright\mathbf{Bool}]\!]^{\sigma}_{0} = \mathbf{1}$ and $\mathcal{SI}[\![\triangleright\mathbf{Bool}]\!]^{\sigma}_{\mathbf{S}p} = \mathbf{Bool}$.

Notice that there are no monotonicity problems with $\theta^{\sigma,\triangleright T}_{(\mathbf{S}p)\to 0}$ since so far we have not define any term M such that $\diamond \vdash^{\mathcal{SI}}_{e} M : \triangleright T$

The fixpoint operator Then, we introduce a fixpoint operator

$$\mathbf{fix}_T : (\triangleright_{\mathcal{U}} T \to T) \to T$$

for every term T of type $\mathcal{U}$. The meaning of $\mathbf{fix}_T$ is given by its approximation at level p, computed by induction. At level 0, the fixpoint of $f : \triangleright_{\mathcal{U}} T \to T$ is given by the value of f on the unique inhabitant of $\triangleright_{\mathcal{U}} T$ at level 0. And at level $p+1$, the value is given by f applied to the approximation at level p. Formally, the translation $\mathcal{SI}[\mathbf{fix}_T]^{\sigma}_{p}$ is defined by

$$\begin{array}{l}
\lambda q : \mathbf{Nat}_p.\lambda f : [\![\triangleright_{\mathcal{U}} T \to T]\!]^{\sigma}_{q}.\mathbf{nat_rect}_{s,q}(\lambda r : \mathbf{Nat}_q.[\![T]\!]^{\sigma}_{r}) \\
(f\,0\star)\,(\lambda r : \mathrm{Pred}_q.\lambda a : [\![T]\!]^{\sigma}_{r}.f(\mathbf{S}r)a)\,q
\end{array}$$

where $\mathrm{Pred}_p \overset{def}{=} \{q : \mathbf{Nat} \mid q < p\}$ and $\mathbf{nat\_rect}_{s,p}$ is defined as a restriction of $\mathbf{nat\_rect}_{\mathcal{U}}$ on the set $\mathbf{Nat}_p$.

The later modality and fixpoint operator on $\mathbf{Prop}_{\mathcal{U}}$ When T is a proposition P, $\mathbf{fix}_P M$ computes a proof of P from a proof M of $\triangleright_{\mathrm{Prop}} P \to P$, This is exactly what the Löb rule

$$\textsc{Löb}\ \frac{\triangleright_{\mathrm{Prop}} P \vdash P}{\vdash P}$$

does, so $\mathbf{fix}_P$ gives the computational content of this rule. Thus, applying the Löb rule on a proof π of $\triangleright_{\mathrm{Prop}} P \to P$ simply amounts to consider the proof term $\mathbf{fix}_P(\pi) : P$.

The lifting of the later modality For every term T of type $\mathcal{U}$, there exists a lifting

$$\mathbf{next}_T : (T \to \triangleright_{\mathcal{U}} T)$$

that transports elements of the presheaf T into elements of the presheaf $\triangleright_{\mathcal{U}} T$ This morphism simply amounts to use the retractions $\theta^{\sigma,T}_{Sp\to p}$ to lift elements of the presheaf accordingly. The translation $[\mathbf{next}_T]^\sigma_p$ is given by

$$\begin{array}{l}\lambda q : \mathbf{Nat}_p.\lambda u : [\![T]\!]^\sigma_q.\texttt{match } q \texttt{ with}\\ \quad \mid 0 \texttt{ => } \star\\ \quad \mid \mathbf{S}q' \texttt{ => } \theta^{\sigma,T}_{q\to q'} u\end{array}$$

Internalizing the later modality for universes For every universe $\mathcal{U}$, we can internalize the shifting induced by the later modality directly in the presheaf. This is done by introducing a morphism

$$\mathbf{switch}_{\mathcal{U}} : (\triangleright_{\mathcal{U}} \mathcal{U} \to \mathcal{U})$$

whose translation $[\mathbf{switch}_{\mathcal{U}}]^\sigma_p$ is defined by

$$\begin{array}{l}\lambda q : \mathbf{Nat}_p.\lambda f : [\![\triangleright_{\mathcal{U}} s]\!]^\sigma_q.\\ \quad (\lambda r : \mathbf{Nat}_q.\texttt{match } r \texttt{ with}\\ \qquad \mid 0 \texttt{ => } \mathbf{1}\\ \qquad \mid \mathbf{S}r' \texttt{ => } (\pi_1 f) r'\\ \quad , \lambda r : \mathbf{Nat}_q.\lambda s : \mathbf{Nat}_r.\lambda M : T_r.\texttt{match } s \texttt{ with}\\ \qquad \mid 0 \texttt{ => } \star\\ \qquad \mid \mathbf{S}s' \texttt{ => } (\pi_2 f)(\mathrm{Pred}\, r) s' M)\end{array}$$

with $T_r \stackrel{def}{=} \texttt{match } r \texttt{ with} \mid 0 \texttt{ => } \mathbf{1} \mid \mathbf{S}r' \texttt{ => } (\pi_1 f) r'$.

The lifting and switching of the later modality are connected by the following lemma.

Lemma 4: *For every $T : \mathcal{U}$ and every forcing condition p, $[\mathbf{switch}_{\mathcal{U}}(\mathbf{next}_{\mathcal{U}} T)]^\sigma_p = [\triangleright_{\mathcal{U}} T]^\sigma_p$.*

4.4.2 Forcing equalities on the new constructors

To make the later modality, the lifting and the switching usable in practice, we introduce proof-terms that make naturality and commutativity properties explicit in the SI layer.

For for any $T : \mathcal{U}_i$ and $U : \mathcal{U}_i$, we add two terms $\mathbf{comlater}_{\mathbf{\Pi}}^{\mathbf{T,U}} : \triangleright_{\mathcal{U}_j}(\Pi x : T.U) \to (\Pi x : \triangleright_{\mathcal{U}_i} T.\, \triangleright_{\mathcal{U}_j} U)$ and $\mathbf{comlater}_{\mathbf{\Sigma}}^{\mathbf{T,U}} : \triangleright_{\mathcal{U}_j}(\Sigma x : T.U) \to \Sigma x : \triangleright_{\mathcal{U}_i} T.\, \triangleright_{\mathcal{U}_j} U$ in the SI layer. Its translation $[\![\mathbf{comlater}_{\mathbf{\Pi}}^{\mathbf{T,U}}]\!]_p^\sigma$ is given by

$$\begin{array}{l}\lambda q : \mathbf{Nat}_p.\lambda f : [\![\triangleright \Pi x : T.U]\!]_q^\sigma. \\ \quad \texttt{match } q \texttt{ with } \texttt{|} \ 0 \ \texttt{=>} \ (\lambda\_.\lambda\_.\star, \ldots) \\ \qquad\qquad\quad \texttt{|} \ \mathbf{S}q' \ \texttt{=>} \ (\lambda r : \mathbf{Nat}_{q'} \lambda u : [\![U]\!]_r^\sigma.fru, \ldots)\end{array}$$

We also add proof-terms that state the naturality of $\mathbf{next}_T$ with respect to T.

4.4.3 General recursive types

Before providing a definition of recursive types for any general recursive definition, we study the property of the translation of **fix**.

Theorem 7: *For every $f : \triangleright\mathcal{U} \to \mathcal{U}$ and every forcing condition p, $[\mathbf{fix}_{\mathcal{U}} f]_p^\sigma \equiv [f(\mathbf{next}_{\mathcal{U}}(\mathbf{fix}_{\mathcal{U}} f))]_p^\sigma$.*

The theorem above is induced by the following (conversion) equalities.

Lemma 5: *For every natural number p, the following holds:*

1. *for every $T : \mathcal{U}$, $[\![\triangleright_{\mathcal{U}} T]\!]_{\mathbf{S}p}^\sigma \equiv [\![T]\!]_p^\sigma$*
2. *for every $M : T$, $[\mathbf{next}_T M]_{\mathbf{S}p}^\sigma \equiv [M]_p^\sigma$*
3. *for every $f : \triangleright\mathcal{U} \to \mathcal{U}$, $[\mathbf{fix}_{\mathcal{U}} f]_{\mathbf{S}p}^\sigma \equiv [f]_{\mathbf{S}p}^\sigma(\mathbf{S}p)[\mathbf{fix}_{\mathcal{U}} f]_p^\sigma$*

Now, given a recursive definition $f : \mathcal{U} \to \mathcal{U}$ for any universe $\mathcal{U}$, we can define

$$\mu_{\mathcal{U}} f = \mathbf{fix}_{\mathcal{U}} \lambda x.f(\mathbf{switch}_{\mathcal{U}} x).$$

Then, using Theorem 7 and Lemma 4, we can add a proof term

$$\mathrm{rec}_{\mu_{\mathcal{U}}} : \mu_{\mathcal{U}} f = f(\triangleright_{\mathcal{U}} \mu_{\mathcal{U}} f)$$

which is just translated in the original layer using **refl** :

$$\mathcal{SI}[\mathrm{rec}_{\mu_{\mathcal{U}}}]_p^\sigma = \mathbf{refl}_{[\mu_{\mathcal{U}} f]_p^\sigma}.$$

Finally, using **eq_rect**, we can define two morphisms $\mathbf{fold}_{\mathcal{U}} : \Pi f : (\mathcal{U} \to \mathcal{U}).\mu_{\mathcal{U}} f \to f(\triangleright_{\mathcal{U}} \mu_{\mathcal{U}} f)$ and $\mathbf{unfold}_{\mathcal{U}} : \Pi f : (\mathcal{U} \to \mathcal{U}).f(\triangleright_{\mathcal{U}} \mu_{\mathcal{U}} f) \to \mu_{\mathcal{U}} f$ together with two proof terms

$$\begin{array}{l}\pi_{\mathcal{U}}^{\mathbf{fold}} : \Pi f : \_.\Pi x : \_.\mathbf{unfold}_{\mathcal{U}} f(\mathbf{fold}_{\mathcal{U}} f x) = x \\ \pi_{\mathcal{U}}^{\mathbf{unfold}} : \Pi f : \_.\Pi x : \_..\mathbf{fold}_{\mathcal{U}} f(\mathbf{unfold}_{\mathcal{U}} f x) = x\end{array}$$

translated using **refl**.

Note that compared to [BMSS11], we do not require contractiveness of the recursive definition to compute a recursive type. But the unfolding of a recursive type introduces uniformly a later modality in front of any use of the recursive variable.

Our definition of recursive types introduces a later modality $\triangleright$ on the recursive type at each unfolding. But in the particular case of recursive types satisfying the strictly positive condition, we can automatically erase the introduced $\triangleright$.

4.4.4 Simple Examples

The Natural Numbers

Let first define a type $\mathbf{Nat}_{\mathcal{SI}}$ of natural number in the forcing layer $\mathcal{SI}$, without using **Nat**. We first define $\mathbf{Nat}_{\mathcal{SI}}$ as $\mu_{\mathcal{U}}(\lambda T : \mathcal{U}.\mathbf{1}+T)$. Then, writing

$$0_{\mathcal{SI}} \stackrel{def}{=} \mathbf{unfold}_{\mathcal{U}}(\lambda T : \mathcal{U}.\mathbf{1}+T)(\mathbf{inl}(\star)),$$

we can check that $0_{\mathcal{SI}} : \mathbf{Nat}_{\mathcal{SI}}$. Similarly, we can define the successor function as

$$S_{\mathcal{SI}} \stackrel{def}{=} \lambda n : \mathbf{Nat}_{\mathcal{SI}}.\ \mathbf{unfold}_{\mathcal{U}}(\lambda T : \mathcal{U}.\mathbf{1}+T)(\mathbf{inr}(n)).$$

Note that the translation of $S_{\mathcal{SI}}$ (in the empty environment) provides a term of $\mathrm{MLTT}_{\mathcal{U}}$ which is already quite complex.

Binary Trees

As another example, we define binary trees on natural numbers in the $\mathcal{SI}$ layer :

$$\mathrm{Tree} = \mu_{\mathcal{U}}(\lambda T : \mathcal{U}.(\varepsilon : 1) + (\mathrm{node} : \mathbf{Nat} \times T \times T)).$$

The properties of $\mu_{\mathcal{U}}$ gives an induction principle annotated with the $\triangleright$ modality.

$$\begin{aligned}\mathrm{Tree}_{\mathrm{ind}} : \Pi P : \mathrm{Tree} \to \mathit{Prop}, P\ \varepsilon \to \\ (\Pi n : \mathrm{Nat}.\Pi t : \triangleright\mathrm{Tree}, \mathbf{switch}((\mathbf{next}\,P)t) \\ \to \Pi t' : \triangleright\mathrm{Tree}, \mathbf{switch}((\mathbf{next}\,P)t') \\ \to P(\mathrm{node}\ n\ t\ t')) \to \Pi t : \mathrm{Tree}, Pt\end{aligned}$$

But using the lifting $\mathbf{next}_{\mathrm{Tree}}$, and the equality of Lemma 4, we can define a derived induction principle.

$$\begin{aligned}\mathrm{Tree}'_{\mathrm{ind}} : \Pi P : \mathrm{Tree} \to \mathit{Prop}, P\ \varepsilon \to \\ (\Pi n : \mathrm{Nat}.\Pi t : \mathrm{Tree}, \triangleright(Pt) \to \Pi t' : \mathrm{Tree}, \triangleright(Pt') \\ \to P(\mathrm{node}\ n\ t\ t')) \to \Pi t : \mathrm{Tree}, Pt\end{aligned}$$

The lifting $\mathbf{next}_P : P \to \triangleright P$ (which gives a computational content to the rule MONO of Gödel-Löb logic) is now used on hypotheses to weaken the formula and get the usual induction principle

$$\begin{aligned}\mathrm{Tree}''_{\mathrm{ind}} : \Pi P : \mathrm{Tree} \to Prop, P\ \varepsilon \to \\ (\Pi n : \mathrm{Nat}.\Pi t : \mathrm{Tree}, Pt \to \Pi t' : \mathrm{Tree}, Pt' \\ \to P(\mathrm{node}\ n\ t\ t')) \to \Pi t : \mathrm{Tree}, Pt\end{aligned}$$

4.4.5 A more complex example: the pure λ-calculus

Let us consider the generalized recursive type

$$\mathcal{D} \stackrel{def}{=} \mu_{\mathcal{U}}(\lambda T : \mathcal{U}.T \to T).$$

This amounts to add in $\mathrm{MLTT}_{\mathcal{U}}$ a type satisfying the usual domain equation for the pure lambda calculus

$$\mathcal{D} = \mathcal{D} \to \mathcal{D}.$$

The idea is that, although this object does not exists in $\mathrm{MLTT}_{\mathcal{U}}$, we can manipulate it directly in the forcing layer, and only its approximations at level n will be considered by $\mathrm{MLTT}_{\mathcal{U}}$'s type checker.

For simplicity, we suppose that the function $\mathbf{next}_T$ is declared as a coercion and thus we do not write it explicitly in the rest of this section. We pose

$$\mathbf{fun} f \stackrel{def}{=} \mathbf{unfold}_{\mathcal{U}}(\lambda T : \mathcal{U}.T \to T) f$$

and

$$\mathbf{defun} f \stackrel{def}{=} \mathbf{fold}_{\mathcal{U}}(\lambda T : \mathcal{U}.T \to T) f.$$

We can introduce an analogous of $\mathbf{switch}_{\mathcal{U}}$ for $\mathcal{D}$ defined as

$$\downarrow : \triangleright\mathcal{D} \to \mathcal{D} \stackrel{def}{=} \lambda t : \triangleright\mathcal{D}.\mathbf{fun}(\lambda\_ : \triangleright\mathcal{D}.t)$$

Thus, we can define application in $\mathcal{D}$ as

$$f@s \stackrel{def}{=}\downarrow (\mathbf{fun} f)s$$

Intuitively, the operator $\downarrow$ tags each β-reduction in the term, thus we get a pure lambda calculus where we can keep track of places where a reduction has occurred. For example, we can construct the usual looping term

$$\Omega \stackrel{def}{=} \mathbf{fun}(\lambda x : \triangleright\mathcal{D}.x@x)$$

and prove that for all integers n,

$$\Omega@\Omega = \downarrow^n (\Omega@\Omega)$$

where $\downarrow^n$ denotes n applications of $\downarrow$. Note that even $\Omega@\Omega$ reduces to itself (plus a lift), there is no problem of termination because we have here an equality but the two terms are not convertible. Indeed, each β-redex is "guarded" by $\mathbf{defun} \circ \mathbf{fun}$ that has to be explicitly rewritten into $\mathbf{id}$ to enable $\mathrm{MLTT}_{\mathcal{U}}$'s conversion.

In the same way, we can define the Y fixpoint combinator as

$$\begin{array}{rrcl} & Y & \stackrel{def}{=} & \mathbf{fun}(\lambda f.(Y'f)@(Y'f)) \\ \text{with} & Y'f & \stackrel{def}{=} & \mathbf{fun}(\lambda x.f@(x@x)) \end{array}$$

Since β-reductions are tagged, we do not have the usual equation $Yg = g(Yg)$. Indeed, those terms are β-equivalent, but the places where β-reductions have to be done are different. Thus, we only get a weaker version of the unfolding lemma.

Lemma 6: *For all terms $g : \mathcal{D}$,*

$$Y@g = \downarrow^2 (g@((Y'g)@(Y'g))).$$

4.5 Forcing the Negation of the Continuum Hypothesis

We now build a forcing layer where we can prove the negation of the Continuum Hypothesis, assuming the excluded middle in the original layer (notice that, as explained in Section 4.3.4, the excluded middle is not preserved in the forcing layer). We adapt the usual construction of Cohen, using its reformulation in terms of topos of sheaves, as presented in [MLM92]. As we will see, the proof is much simpler than the usual one in classical forcing. Indeed, we do not need any hypothesis like the *countable chain condition* [Jec03] on the set of forcing conditions $\mathcal{P}$. This is due to the fact that we are working with constant presheaves to translate types like **Nat**, so that, as we will see, conservation of cardinals is straightforward. This would not be the case if we were working with constant sheaves (a constant presheaf is not a sheaf in general).

In the following, $\mathbf{P}(T)$ denotes the type $T \to \mathrm{Prop}$ corresponding to the "power set" of T. We suppose to work with resizing axiom, as defined in Section 3.1.8, so that the definition of Prop does not need to be indexed by a universe. We build a forcing layer where we can add a type A with injections

$$\mathbf{Nat} \hookrightarrow^{i_1} A \hookrightarrow^{i_1} \mathbf{P}(\mathbf{Nat})$$

and such that there is no surjection from **Nat** to A and from A to $\mathbf{P}(\mathbf{Nat})$, thus negating the Continuum Hypothesis. The set of forcing conditions is given by

$$\mathcal{P} \stackrel{def}{=} (\mathbf{P}(\mathbf{P}(\mathbf{Nat})) \times \mathbf{Nat}) \to_{fin} \mathrm{Prop}$$

where $\rightarrow_{fin}$ denotes the type of function with finite support. This type can be defined in $\mathrm{MLTT}_{\mathcal{U}}$ using dependent sums. The order is the usual inclusion order on functions: $q \subseteq p$ if the domain of p is smaller than the domain of q and the two functions coincide on their common domain.

For a closed type $T : \mathcal{U}$, we write $\widehat{T}$ for the constant presheaf defined as

$$[\![\widehat{T}]\!]^{\sigma}_{p} = (\lambda q : \mathcal{P}_p.T, \lambda q : \mathcal{P}_p.\lambda r : \mathcal{P}_q.id).$$

The set A that will negate the continuum hypothesis is $\widehat{\mathbf{P}(\mathbf{Nat})}$. The idea is to collapse $\mathbf{P}(\widehat{\mathbf{P}(\mathbf{Nat})})$ to $\mathbf{P}(\mathbf{Nat})$ in the forcing layer. Then, using the injection from $\widehat{\mathbf{P}(\mathbf{Nat})}$ to $\mathbf{P}(\widehat{\mathbf{P}(\mathbf{Nat})})$, this gives us the wanted injection $i_2 : \widehat{\mathbf{P}(\mathbf{Nat})}$ to $\mathbf{P}(\mathbf{Nat})$. To sum up, we will build injection

$$\mathbf{Nat} \hookrightarrow^{i_1} \widehat{\mathbf{P}(\mathbf{Nat})} \hookrightarrow^{i_2} \mathbf{P}(\mathbf{Nat})$$

with the proofs of non-existence of surjections

$$\mathbf{Nat} \rightarrowtail A \rightarrowtail \mathbf{P}(\mathbf{Nat})$$

We insist on the fact that $\widehat{\mathbf{P}(\mathbf{Nat})}$ is not equal to $\mathbf{P}(\widehat{\mathbf{Nat}}) = \mathbf{P}(\mathbf{Nat})$ since $[\![\mathbf{P}(\mathbf{Nat})]\!]^{\sigma}_{p} \stackrel{def}{=} \mathbf{Nat} \rightarrow (\mathcal{P}_p \rightarrow \mathrm{Prop})$ while $[\![\widehat{\mathbf{P}(\mathbf{Nat})}]\!]^{\sigma}_{p} \stackrel{def}{=} \mathbf{Nat} \rightarrow \mathrm{Prop}$.

4.5.1 Building the injection i_2

We add a new symbol f of type $\mathbf{P}(\widehat{\mathbf{P}(\mathbf{Nat})}) \rightarrow \mathbf{P}(\mathbf{Nat})$ whose translation is defined as

$$[f]^{\sigma}_{p} \stackrel{def}{=} \lambda q : \mathcal{P}_p.\lambda b : \mathbf{P}(\mathbf{P}(\mathbf{Nat})).\lambda r : \mathcal{P}_q.\lambda n : \mathrm{Nat}.\lambda s : \mathcal{P}_r. \\ (s \preccurlyeq r \wedge s(b, n) = \mathbf{true})$$

Theorem 8: *For every forcing condition p, $[f]^{\sigma}_{p}$ is an injection, which means that the proposition*

$$[\![\Pi b_1, b_2 : \mathbf{P}(\mathbf{P}(\mathbf{Nat})).f(b_1) = f(b_2) \Rightarrow b_1 = b_2]\!]_p$$

is provable in $\mathrm{MLTT}_{\mathcal{U}}$ *plus excluded middle.*

Proof Since p is a partial map from $(\mathbf{P}(\mathbf{P}(\mathbf{Nat})) \times \mathbf{Nat})$ to Prop whose domain is finite, we can find an m such that $p(b_1, m)$ and $p(b_2, m)$ are undefined. If $b_1 \neq b_2$, we can define a new forcing condition q that extend p such that $q(b_1, m) = \mathbf{true}$ and $q(b_2, m) = \mathbf{false}$. But then we can easily check that

$$[f]^{\sigma}_{p}(p)(b_1)(p)(m)(q) = \mathbf{true}$$

and

$$[f]^{\sigma}_{p}(p)(b_2)(p)(m)(q) = \mathbf{false}$$

Using the excluded middle, we can conclude that $b_1 = b_2$. ∎

Finally, i_2 is built from f as:

$$i_2 \stackrel{def}{=} \widehat{\mathbf{P}(\mathbf{Nat})} \hookrightarrow \mathbf{P}(\widehat{\mathbf{P}(\mathbf{Nat})}) \xrightarrow{f} \mathbf{P}(\mathbf{Nat})$$

The injection i_1 is just obtained as the lifting of the injection from **Nat** to $\mathbf{P}(\mathbf{Nat})$.

4.5.2 Absence of surjections

We now prove that if there is no surjection between S and T in the ground system, then there is no surjection between $\widehat{S}$ and $\widehat{T}$ in the forcing layer. This is usually called the "conservation of cardinals".

Assume the formula

$$\mathbf{NS}(T, S) \stackrel{def}{=} (\Sigma F : T \to S.\Pi s : S.\Sigma t : T.F(t) = s) \to \mathbf{false}$$

We will prove in the forcing layer the formula $\mathbf{NS}(\widehat{T}, \widehat{S})$.

As the restriction map on $\widehat{T}$ and $\widehat{S}$ are equal to the identity, $[\![\mathbf{NS}(\widehat{T}, \widehat{S})]\!]_p^\varepsilon$ just enforces that there is no $F : \mathcal{P}_q \to (T \to S)$ such that $F(q)$ is a surjection between T and S, which is given by $\mathbf{NS}(T, S)$.

4.6 Discussion and Future Work

We have introduce a modular way to extend type theory while keeping its good properties. However, some theoretical problems remain, concerning the notion of explicit coercions of conversions and the unicity of the translation.

Indeed, if we impose equations between explicit conversions they must be taken into accounts when defining $\mathbf{convEq}_{M,N}^{\sigma,p}$. But as explained in Section 3.1.7, all those equality should be based on properties on $\mathbf{transport}_{\mathcal{U}}^{\lambda T.T}$. Therefore, as $[\mathbf{c}_{T,U}(M)]_p^\sigma$ is defined using $\mathbf{transport}_{\mathcal{U}}^{\lambda T.T}$, all equations between explicit conversions would probably trivially hold in the translation.

4.6.1 Topos of trees

In [BMSS11], Birkedal et al. have studied the internal logic of the topos of presheaves on ω. They show that this logic admits general recursive types as soon as the considered recursive definitions are contractive. In Section 4.4, we have presented a syntactic translation of this logical layer using forcing.

Our presentation presents several advantages:

— Since we have access to the universe $\mathcal{U}$, we are able to define an automatic translation from any recursive definition to a contractive recursive definition.

— General recursive types are translated into $\mathrm{MLTT}_{\mathcal{U}}$ terms, and can thus be given computational content.

— The type-checking of the Step-Indexed forcing Layer is decidable.

This semantics model was later extended by Birkedal and Mogelberg to intentional Martin-Löf Type Theory with one universe in [BM13]. It should be possible to show that their model, a presheaf category $\mathbb{C}^{\omega^{op}}$, where $\mathbb{C}$ is a model of Martin-Löf Type Theory with one universe is a model of this forcing layer. Indeed, the main difference with our work is that they are using Tarksi-style notion of universes.

Of particularly interest in their work is a result relating fixpoint on universes with guarded recursive types. It should be possible to import it in our setting. This would be particularly useful because guarded recursive types as defined in [BMSS11] are easier, in certain circumstances, to work with compared to fixpoints on universes. Indeed, they satisfies a *Banach fixpoint theorem* which give unicity of fixpoints. These has to be compared to usual fixpoints theorems, like Knaster-Tarski, where we have unicity of lowest and greatest fixpoints. With Banach fixpoint theorem, this means that these two notions coincide, so we can use guarded recursive types to define *coinductive types*. Such coinductive definitions has recently been studied by McBride and Atkey [AM13], and Mogelberg [Møg]

4.6.2 Higher-Order Abstract Syntax

In Section 4.4.5, we have sketched a formalization of the untyped λ-calculus with guarded recursive types. It would be interesting to link this idea to the litterature on *Higher-Order Abstract Syntax* [PE88] (HOAS). The basic idea of HOAS is to use the binder of the theory to represent the binder of the formalized language [1]. In general, there are two problems for doing this in type theory. First, it requires to define an inductive type for $\mathcal{D}$, the domain of the language, which does not satisfy the usual strictly positive condition. Second, there can be terms in $\mathcal{D}$ that do not correspond to "real" λ-terms.

Despeyroux, Felty and Hirschowitz [DFH95] have proposed a way to define a weaker version of HOAS in Coq, where they use an auxiliary type Var of variables, so that $\mathcal{D}$ is then defined as $(\mathrm{Var} \to T) \to T$. With this workaround, function abstraction is still defined using the abstraction of the type theory, but this is no more the case for substitution. Then, they define a predicate to restrict inhabitants of this types to the one which are extensionally equal to effective λ-terms. It would be interesting to adapt their work to the new possibility of using guarded recursive types to define a general type $\mathcal{D}$ without any workaround.

Maybe more interestingly, various models of HOAS has been built using a presheaf construction over categories. One can cite the work of Fiore, Plotkin and Turi [FPT99], the work of Hofmann [Hof99] and the work of Gabbay and Pitts [GP99]. In fact, this last article was the origin of nominal logic. It would be interesting to see if these constructions could be imported in our setting, but this would necessitate first to extend our presheaf translation with categories as forcing conditions.

1. This corresponds to the definition of a shallow embedding of the language in the theory.

4.6.3 Presheaves models of HoTT

In recent works [Shu12, Shu13], Shulman has built presheaves model of Intentional Type Theory which conserve the Univalence Axiom. That is, from a model of Type Theory $\mathcal{C}$ which validates the univalence axiom (namely *sSet*, the category of simplicial sets), he has shown that the category of presheaves $\mathcal{C}^{\mathcal{D}^{op}}$ over a category $\mathcal{D}$ still validates the univalence axiom, when $\mathcal{D}$ has special property (Inverse diagrams, Elegant Reedy Category). This was then used by Lars Birkedal and Rasmus Mogelberg [BM13] to show that the category of presheaves $\mathcal{C}^{\omega^{op}}$ validates univalence when $\mathcal{C}$ already validates it.

It would be interesting, once extending our work to a presheaf translation over a category (rather than a poset), to deduce from the work of Shulman what must imposed on the set of forcing condition to preserve the univalence axiom.

4.6.4 Forcing as a program transformation

Krivine has been one of the first to study the computational content of the forcing translation [Kri11]. This work has been rephrased by Miquel in [Miq11], where the forcing translation of proofs is interpreted as a program transformation on λ-terms. They focus on classical logic while we only study an intuitionistic translation. Moreover, since they are working in a logical system with a syntactic stratification between proofs and propositions ($\mathbf{PA}\omega$ in [Miq11]), proof-terms and formulas are not translated uniformly, contrary to our work.

4.6.5 From presheaves to sheaves

As we have seen in 4.3.4, our forcing translation is intuitionistic. A solution to conserve the excluded middle would be to use sheaves instead of presheaves, for a well suited topology (the dense or double negation topology) [MLM92]. Sheaves restrict presheaves with gluing conditions, so we need to introduce a notion of topology on $\mathcal{P}$, formalized inside $\mathrm{MLTT}_{\mathcal{U}}$, to define them. The best way to do it seems to use the notion of Lawvere-Tierney topology, which provides a categorical definition of sheaves where the topology is induced by particular function $j : \mathrm{Prop} \to \mathrm{Prop}$ satisfying some commutative diagrams. The idea is then to instantiate j by $\neg\neg$ to get the dense topology.

4.6.6 Constructive Mathematics

As we said in the introduction, one of the starting point of this forcing translation was our previous work with Thierry Coquand [CJ10, CJ12] to give a computational content to a result of uniform continuity of definable functionals. It would be interested to recast it using the translation presented in this chapter. However, the translation used in this work was not purely intuitionistic, but it was rather inspired by Beth semantics [Bet59]. That

is, a notion of covering on forcing conditions—similar to what is used in sheaves theory—was used. Indeed, our work was even later generalized by Escardo and Xu [XE13, Esc13] by using sheaves models. This means that the presheaf translation presented here would not be enough to restate these works, and we would need a sheaf translation.

An other interesting example of forcing translation in constructive mathematics we wish to adapt is the proof of Levin [Lev77] of conservativity of the existence of an ultrafilter on $\mathbb{N}$ with respect to higher-order Peano Arithmetic with the axiom of dependent choice. This proof uses partial functions from **Nat** to **Bool** (with an infinite domain) as forcing conditions. Defining a forcing layer on this set could allow us to give a computational content to proofs which use an ultrafilter, like Gödel's completeness theorem or the Ramsey theorem. It would then be interesting to compare this interpretation with the interpretation of Krivine [Kri11].

5

Operational Nominal Game Semantics

To achieve our goal of proving the completeness of Kripke logical relations, defined in a direct way, for RefML, we need to link them to a *fully abstract* denotational model *i.e.* such that the denotations of two terms are equal iff they are contextually equivalent. The long-standing quest of such fully abstract models has seen its achievement with *game semantics*.

Game Semantics [HO00, AJM00] is a powerful theory to build fully abstract denotational models of various programming languages. The denotation of a term is represented as a *strategy*, a set of *plays* between that term and any context in a *game arena*, which sets the rules the plays have to satisfy. One of its most important contributions, the so-called "Abramsky Cube", is the characterization of the absence of various impure effects in terms of extra conditions on the denotation of terms, namely *well-bracketing* for the absence of control operators, *visibility* for the absence of higher-order store, *innocence* for pure terms. In recent years, game semantics has been developed to deal with languages with nominal aspects, from the ν-calculus [AGM+04], an extension with storage cells [Lai08], to ML-like languages with higher-order nominal references [MT11b].

The starting point of this section is the nominal game semantics of Murawski and Tzevelekos [MT11b], presented in Section 5.1, which is fully abstract for RefML, but also for GroundML as soon as one adds a visibility condition to strategies [MT12]. As opposed to previous games models for languages with stores, initiated by Abramsky, Honda and McCusker [AHM98], it uses nominal techniques [Pit03], already used in [Tze07], to avoid the problem of *bad variables*. In a more operational setting, Laird [Lai07] has introduced a trace semantics for a variant of RefML, and has proven its full abstraction. This model marries a trace representation inspired by game semantics with an operational definition, *i.e.* denotations of terms are computed via a rewriting system rather than defined by induction on their typing judgment.

In this section, we introduce a trace semantics for RefML, whose definition is a typed variant to the one introduced by Laird. Traces are generated by an *interactive reduction*, which can be seen as an extension of the usual operational semantics to open terms with

free functional variables. So this reduction can be performed on terms like $\lambda x.M$ or $K[f\ v]$. Then, the denotation of terms is defined via *trace-strategies*, *i.e.* sets of traces that terms generate using this reduction.

Trace semantics allows a finer study of the interaction compared to game semantics, by keeping track of the disclosure process of values (either locations or functions) between the term and contexts. Indeed, in game semantics, one cannot generally decompose plays into "subplays", since we get sequences of moves where some of these moves are no more justified, and thus left uncontrolled. With trace semantics, such unjustified moves are simply seen as disclosed values, which can be taken care of by the interactive reduction. Using this finer decomposition, we can prove a decidability result for the contextual equivalence of "pure" terms of RefML, that is terms which do not use any storage possibilities (but with unrestricted contexts). It uses crucially the control of disclosed functions to perform surgery on traces. In the next chapter, we will see how to perform such surgery for "impure" terms, by defining new techniques to reason on disclosure of locations.

In fact, traces can be seen as a representation of plays used in game semantics where the usual pointer structure, which represents the causality between the different moves, is encoded with variables. Such variables that are of functional type are called *name pointers*.

Following this idea, we prove a correspondence between this denotation of terms, defined as set of traces, and the denotation of game semantics defined using sets of plays. To do so, we impose on trace-strategies a categorical apparatus that capture call-by-value languages, namely a *closed-Freyd category* [PR97, PT99]. It consist in

— a symmetric premonoidal category $(\mathcal{C}, I, \otimes)$,

— a luff category $\mathcal{C}'$ of $\mathcal{C}$, for which $\otimes$ is a cartesian product,

— a strict premonoidal functor $(.)^\dagger$ between $\mathcal{C}'$ and $\mathcal{C}$, which is the identity on objects,

such that for every object A of $\mathcal{C}$, the functor $(\_ \otimes A)^\dagger : \mathcal{C}' \to \mathcal{C}$ has a right adjoint. To build such a structure on traces, we recast the definitions of game semantics from [Lai08, MT11b] in the setting of trace semantics. The main difficulty is that the pointer structure of traces is no more defined explicitly, but need to be rebuilt from a study of freshness of name pointers (*i.e.* functional variables).

Then, we relate the usual notion of view from game semantics to sets of *available* player name pointers, so that we can import the visibility condition which capture the behavior of terms of GroundML directly on traces. Finally, modifying the interactive reduction to restrict the opponent questions to be performed on such available player name pointers, we can capture the contextual equivalence of GroundML with trace equivalence.

Plan of the Chapter The basic notions of game semantics (in a call-by-value setting) and its categorical structure are introduced in Section 5.1. Section 5.1.7 recall the basic facts of *nominal game semantics*.

The notion of trace is formally defined in Section 5.2.1 and its correspondence with

plays in game semantics is drawn in Section 5.2.2. The interactive reduction is introduced in Section 5.2.3 and the interpretation of terms as sets of traces is defined in Section 5.2.4. Then, one of the main contribution in this chapter, presented in Section 5.3, is the comparison of this trace model to the game semantics model of Murawski and Tzevelekos [MT11b], and the proof that the two are in fact equivalent. We also answer a question asked by Laird at the end of [Lai07] about a possible trace semantics for a language with restricted references, rephrasing the usual notion of *visibility* in the setting of trace semantics. This is done in Section 6.7.2. Finally, in Section 5.5, we exemplify the usefulness of trace semantics by proving that contextual equivalence of pure terms in RefML (and also GroundML) is decidable. This gives a first example of the ideas introduced in the next two chapters to reason on contextual equivalence.

5.1 Call-By-Value Game Semantics

Let us begin by introducing standard definitions from *game semantics*. The main idea is to interpret terms as *strategies*, which are well-behaved sets of *plays*, representing their interactions between any possible contexts (also called environments) in terms of questions/answers. These plays are formed by *moves*, which are either ground values (integer, booleans, locations), or a special move $\star$ representing a functional value. Such moves also indicates in which *arena* they belong, using a notion of tag introduced subsequently, which is useful to build disjoint union of arenas. Then, plays are defined as lists of moves with a pointer structure.

Here, we focus on call-by-value game semantics. Its main difference, compared to the usual call-by-name game semantics, is that it uses two different arenas, depending on whether we represent terms or values. Then, initial moves of arenas used to represent values are *player answers*. Our presentation follows the work of Honda and Yoshida [HY99], in that we give a direct presentation of the semantics of terms, using a structure of *closed-Freyd* category [PR97, PT99] while Abramsky and McCusker [AM98] rather build a model of call-by-value using the work of Moggi [Mog91], *i.e.* via a strong monad. The two presentations are equivalent, however, our trace semantics is defined in a direct way, being more operational. This justifies the choice of closed-Freyd categories.

5.1.1 Arenas

Plays are formed by moves, which represent actions performed by the term or the context. They are elements of *arenas*.

Definition 7 (Arena). *An arena A is defined as a tuple $(M_A, I_A, \lambda_A, \vdash_A)$ such that*
— *M_A is the set of* moves,
— *$I_A \subseteq M_A$ is the set of* initial *moves,*

— $\lambda_A : M_A \to \{O, P\} \times \{Q, A\}$ *is the labeling function which associates the polarity* $\{O, P\}$ *and the statute* $\{Q, A\}$ *of a move,*
— $\vdash_A$ *is the* justification *relation between* M_A *and* $M_A \backslash I_A$*, which satisfies:*
  — *If* $m \vdash_A m'$ *then* $\lambda_A^{OP}(m) \neq \lambda_A^{OP}(m')$ (i.e. *Player and Opponent alternate).*
  — *If* $m \vdash_A m'$ *and* $\lambda_A^{QA}(m') = A$ *then* $\lambda_A^{QA}(m) = Q$ (i.e. *an answer is necessarily justified by a question).*

In the following, we write $\mathrm{i}, \mathrm{a}, q, o, p$ respectively for initial, answers, questions, opponent or player moves. Unusually, moves comes with a tag, which are words ξ from the alphabet $\{\ell, r\}$, where the concatenation is written $\xi \cdot \xi'$ and the empty tag (*i.e.* the empty word) is written ε. Such tags are used to indicate to which arena a move belongs to, which is useful when we build arenas from other ones. It avoids to use disjoint union (seen as coproduct) with the usual injections $\mathbf{in_l}, \mathbf{in_r}$. So moves m are defined as pairs (a, ξ) of an atom (which can be a ground value, a special symbol $\star$ or a tuple of moves), and a tag ξ We extend the concatenation of tags ξ to act on moves $m = (\mathrm{a}, \xi)$, written $\xi' \cdot m$, and defined as $(\mathrm{a}, \xi' \cdot \xi)$. Such concatenation operation is then extended to act pointwisely on set and relations of moves.

In call-by-value game semantics, we have to introduce two kinds of arenas, depending on whether we want to interpret terms or values. Such distinction is not present in usual call-by-name game semantics.

Definition 8 (Term-Arena). *A Term-Arena* A *is an arena* $(M_A, \lambda_A, \vdash_A)$ *which also satisfies that for all* $\mathrm{i}_A \in I_A$, $\lambda_A(m) = (O, Q)$.

Definition 9 (Value-Arena). *A Value-Arena* A *is an arena* $(M_A, \lambda_A, \vdash_A)$ *which also satisfies that for all* $\mathrm{i}_A \in I_A$, $\lambda_A(m) = (P, A)$.

Notice that we have changed the names usually used in the literature on call-by-value games, where a term-arena is called prearena and value-arena is simply called an arena. We use this new terminology to insist on the fact that the interaction of a term with a context will take place in a term-arena. Indeed, before reducing to a value, a (non-closed) term can interact with its contexts. This is for example the case of the term `let x = f 3 in λg.g x`, where `f` is a function of type Int → Int. In a call-by-value setting, an interaction of this term can be seen as:

— Player ask the value of `f` to the context with the value `3`,

— Opponent answers a value `n`,
— Player answers λ`g.g x` (we will see that it is represented by a move $\star$),
— then Opponent can interrogate this function.

Let us now illustrate those definitionsby introducing some Value-arenas and constructions on Values-arenas.

Definition 10 (Ground-Type Value-Arenas). *We associate to each ground type ι the Value-arena $[\![\iota]\!] \stackrel{def}{=} (M_{[\![\iota]\!]}, I_{[\![\iota]\!]}, \lambda_{[\![\iota]\!]}, \vdash_{[\![\iota]\!]})$ defined as*

— $M_{[\![\iota]\!]} \stackrel{def}{=} \{(v, \varepsilon \mid v \text{ a closed value of type } \iota\}$ *and* $I_{[\![\iota]\!]} \stackrel{def}{=} M_{[\![\iota]\!]}$,
— $\lambda_{[\![\iota]\!]} \stackrel{def}{=} m \mapsto (P, A)$,
— $\vdash_{[\![\iota]\!]} = \varnothing$.

So for example, the arena $[\![\text{Int}]\!]$ associated to the type Int is defined as the set of moves $M_{[\![\text{Int}]\!]} \stackrel{def}{=} \{\widehat{n} \mid n \in \mathbb{Z}\}$, $I_{[\![\text{Int}]\!]} = M_{[\![\text{Int}]\!]}$ with the labeling function $\lambda_{[\![\text{Int}]\!]} \stackrel{def}{=} \widehat{n} \mapsto (P, A)$ with $n \in \mathbb{Z}$.

Even if we do not consider type products in our language, we need to define the product of two value-arenas to interpret contexts.

Definition 11 (Product Value-Arena). *Let A, B two Value-Arenas, we define the Value-arena $A \otimes B$ as the triple $(M_{A\otimes B}, I_A \otimes B, \lambda_{A\otimes B}, \vdash_{A\otimes B})$ where*

— $I_{A\otimes B} \stackrel{def}{=} ((\ell \cdot I_A \times r \cdot I_B))$
— $M_{A\otimes B} \stackrel{def}{=} I_{A\otimes B} \uplus \ell \cdot (M_A \backslash I_A) \uplus r \cdot (M_B \backslash I_B)$
— $\lambda_{A\otimes B} \stackrel{def}{=} [(\ell \cdot \mathrm{i}_A, r \cdot \mathrm{i}_B) \mapsto OQ] \uplus \ell \cdot \lambda_A \uplus r \cdot \lambda_B$
— $\vdash_{A\otimes B} \stackrel{def}{=} \{((\ell \cdot \mathrm{i}_A, r \cdot \mathrm{i}_B), \ell \cdot m) \mid \mathrm{i}_A \vdash_A m\} \uplus \{((\ell \cdot \mathrm{i}_A, r \cdot \mathrm{i}_B), r \cdot m) \mid \mathrm{i}_B \vdash_B m\} \uplus (\ell \cdot \vdash_{A|(M_A \backslash I_A)^2}) \uplus (r \cdot \vdash_{B|(M_B \backslash I_B)^2})$

Definition 12 (Arrow Value-Arena). *Let A, B two Value-Arenas, we define the Value-arena $A \Rightarrow B$ as the triple $(M_{A\Rightarrow B}, I_{A\Rightarrow B}, \lambda_{A\Rightarrow B}, \vdash_{A\Rightarrow B})$ where*

— $I_{A\Rightarrow B} \stackrel{def}{=} \{\star\}$
— $M_{A\Rightarrow B} \stackrel{def}{=} I_{A\Rightarrow B} \uplus (\ell \cdot M_A) \uplus (r \cdot M_B)$
— $\lambda_{A\Rightarrow B} \stackrel{def}{=} [\star \mapsto PA] \uplus \ell \cdot (\overline{\lambda_A}[\mathrm{i}_A \mapsto OQ]) \uplus r \cdot \lambda_B$
— $\vdash_{A\Rightarrow B} \stackrel{def}{=} \{(\star, \ell \cdot \mathrm{i}_A)\} \uplus \{(\ell \cdot \mathrm{i}_A, r \cdot \mathrm{i}_B)\} \uplus (\ell \cdot \vdash_A) \uplus (r \cdot \vdash_B)$

Definition 13 (Arrow Term-Arena). *Let A, B two Value-Arenas, we define the Term-arena $A \to B$ as the triple $(M_{A\to B}, I_{A\to B}, \lambda_{A\to B}, \vdash_{A\to B})$ where*

— $I_{A\to B} = I_A$

— $M_{A\to B} \stackrel{def}{=} (\mathcal{l} \cdot M_A) \uplus (\mathcal{r} \cdot M_B)$

— $\lambda_{A\to B} \stackrel{def}{=} \mathcal{l} \cdot (\overline{\lambda_A}[\mathrm{i}_A \mapsto OQ]) \uplus \mathcal{r} \cdot \lambda_B$

— $\vdash_{A\to B} \stackrel{def}{=} \{(\mathcal{l} \cdot \mathrm{i}_A, \mathcal{r} \cdot \mathrm{i}_B)\} \uplus (\mathcal{l} \cdot \vdash_A) \uplus (\mathcal{r} \cdot \vdash_B)$

5.1.2 Plays

We now consider finite sequences of moves $s = m_1 \dots m_n$ of an arena A equipped with a pointer structure, that is, for every move m_i ($i \in \{2, \dots, n\}$) which is not in I_A, a *justification pointer* to a move m_j such that $j < i$ and $m_j \vdash_A m_i$. Such sequences of moves are called *justified sequences* when additionally only $m_1 \in I_A$. When a move m of a justified sequence s points to an other move m', we say that m' justifies m. If moreover m' is a question and m an answer, we say that m answers m'. Taking a justified sequence s, we write $s' \sqsubseteq s$ when s' is a prefix of s. If moreover s' is of even-length (resp. odd-length), we write $s' \sqsubseteq^{\text{even}} s$ (resp. $s' \sqsubseteq^{\text{odd}} s$). When m is a move of s, we write $s_{\leq m}$ for the prefix of s whose last element is m. Considering a justified sequence of moves s on an arena $A \to B$, we often need to consider the subsequences $s_{|A}$ (resp. $s_{|B}$) on the arena A (resp. B), defined as the subsequences of s formed by moves $m = (\mathrm{a}, \xi)$ such that $(\mathrm{a}, \mathcal{l} \cdot \xi)$ (resp. $(\mathrm{a}, \mathcal{r} \cdot \xi)$) is a move of s.

We now introduce the notion of *plays*, which are justified sequences satisfying the constraints of *alternation* and *well-bracketing*.

Definition 14 (Play). *A* play *s on an arena A is a justified sequence on A such that*

— *each answer* a *of s is justified (*i.e. *answers) by the last unanswered question of $s_{\leq a}$ (Well-Bracketing),*

— *for every adjacent moves mn of s, $\lambda_A^{OP}(m) \neq \lambda_A^{OP}(n)$ (Alternation).*

The set of plays on an arena A is written P_A. When working on GroundML, we need to restrict plays with the usual notion of *P-visibility* and *O-visibility*, defined using the *view* of a justified sequence.

Definition 15 (P-View). *The* view *$\ulcorner s \urcorner$ of an odd-length justified sequence s on A is the subsequence of s defined by induction:*

— $\ulcorner \varepsilon \urcorner = \varepsilon$,
— $\ulcorner \mathrm{i}_A \urcorner = \mathrm{i}_A$
— $\ulcorner s' \cdot m \cdot s'' \cdot n \urcorner = \ulcorner s' \urcorner \cdot m \cdot n$ *if* n *points to* m.

Definition 16 (P-Visibility). *A justified sequence* s *on* A *satisfies the* P-visibility condition *iff for all* $s'p \sqsubseteq^{\text{even}} s$, p *points to a move in* $\ulcorner s' \urcorner$.

P-visibility restricts the possibilities of interactions terms have, so that they correspond to terms of GroundML. However, contexts can still use higher-order references in this setting. To forbid it, we need to introduce the dual notion of O-visibility.

Definition 17 (O-Visibility). *A justified sequence* s *on* A *satisfies the* O-visibility condition *iff for all* $s'o \sqsubseteq^{\text{odd}} s$, o *points to a move in* $\ulcorner s' \urcorner$.

Then a justified sequence is said to be *visible* when it is both P- and O-visible. This characterization captures exactly GroundML.

5.1.3 Strategies

We now introduce the notion of *strategies*, which are sets of plays on an arena A which represent the denotation of terms.

Definition 18 (Strategy). *A* strategy υ *on an arena* A, *written* $\upsilon : A$ *is a set of even-length plays on* A *which is downward-closed,* i.e. *for all* $sop \in \upsilon$, *we have then* $s \in \upsilon$.

One of the most important strategy is the so-called copycat, which represents the identity function. Taking an arena A, we define $\mathrm{id}_A : A \to A$ as the set of plays

$$\{s \in P_{A \to A} \mid s \text{ even} \wedge s_{|A_l} = s_{|A_r}\}$$

where A_l (resp. A_r) represents the left (resp. the right) occurence of A in the arena $A \to A$. More generally, we can define the projections $\pi_i : A_1 \otimes A_2 \to A_i$ for $i \in \{1, 2\}$ in the same way.

To get a denotational model, we still have to define a way to compose the denotation of two terms of type $A \to B$ and $B \to C$. This is done in two steps, called *parallel composition* and *hiding*. More precisely, the parallel composition of a strategy $\upsilon : A \to B$ and a strategy $\kappa : B \to C$, written $\upsilon||\kappa$ is a set of legal sequence of moves of M_A, M_B, M_C. Then, the hiding simply removes the moves of M_B, and update the pointer structure, so that to get a new strategy, written $\upsilon;\kappa : A \to C$.

Let us first introduce the term-arena $A \to B \to C$ associated to three value-arenas A, B, C[1]:

— $I_{A\to B\to C} \stackrel{def}{=} (\ell \cdot \ell) \cdot I_A$

— $M_{A\to B\to C} \stackrel{def}{=} (\ell \cdot \ell) \cdot M_A \uplus (\ell \cdot r) \cdot M_B \uplus (r \cdot r) \cdot M_C$,

— $\lambda_{A\to B\to C} \stackrel{def}{=} (\ell \cdot \ell) \cdot (\overline{\lambda_A}[\mathrm{i}_A \mapsto OQ]) \uplus (\ell \cdot r) \cdot (\lambda_B[\mathrm{i}_B \mapsto PQ] \uplus (r \cdot r) \cdot (\lambda_C)$

— $\vdash_{A\to B\to C} \stackrel{def}{=} \{(\ell \cdot \ell \cdot \mathrm{i}_A, \ell \cdot r \cdot \mathrm{i}_B)\} \uplus \{(\ell \cdot r \cdot \mathrm{i}_B, r \cdot r \cdot \mathrm{i}_C)\} \uplus (\ell \cdot \ell \cdot \vdash_A) \uplus (\ell \cdot r \cdot \vdash_B) \uplus (r \cdot r \cdot \vdash_C)$

Then, taking a justified sequence $s \in A \to B \to C$, we define the following restricted justified sequences:

— $s_{|AB}$ is the justified subsequence of s formed by all moves (a, ξ) *s.t.* $(\mathrm{a}, \ell \cdot \xi)$ is a move of s,

— $s_{|BC}$ is the justified subsequence of s formed by all moves $(\mathrm{a}, \ell \cdot \xi)$ *s.t.* $(\mathrm{a}, \ell \cdot r \cdot \xi)$ is a move of s, and all moves (a, ξ) *s.t.* $(\mathrm{a}, r \cdot \xi)$ is a moves of s

— $s_{|AC}$ is the justified subsequence of s formed by all moves $(\mathrm{a}, \ell \cdot \xi)$ *s.t.* $(\mathrm{a}, \ell \cdot \ell \cdot \xi)$ is a move of s, and all moves (a, ξ) *s.t.* $(\mathrm{a}, r \cdot \xi)$ is a moves of s, and where we add pointers from moves $m \in M_C$ to $m' \in M_A$ if there exists a move $n = (\mathrm{a}, \ell \cdot r \cdot \xi)$ of s *s.t.* $r \cdot m$ points to n and n points to $\ell \cdot m'$ in s.

Notice that $s_{|AB}$ (resp. $s_{|BC}, s_{|AC}$) is a justified sequence of $A \to B$ (resp. $B \to C$, $A \to C$), so that the labeling function of their moves are supposed to be taken in these term-arenas, and not in $A \to B \to C$.

We can now define the set of interaction sequences $\mathbf{Interact}(A, B, C)$ on A, B, C as the justified sequences s over $A \to B \to C$ which are well-bracketed and which satisfy $s_{|AB} \in P_{A\to B}$, $s_{|BC} \in P_{B\to C}$ and $s_{|AC} \in P_{A\to C}$.

The parallel composition of two strategies is then simply defined as the set of interaction sequences over $\mathbf{Interact}(A, B, C)$ whose respective restrictions on AB and BC are in the corresponding strategies.

Definition 19 (Parallel Composition of strategies). *Given two strategies υ and κ respectively on the arenas $A \to B$ and $B \to C$, we define their parallel composition on the arena $A \to B \to C$, written $\upsilon||\kappa$ as the set $\{s \in \mathbf{Interact}(A, B, C) \mid s_{|AB} \in \upsilon$ and $s_{|BC} \in \kappa\}$.*

1. Notice that this arena cannot be defined neither as $(A \to B) \to C$ nor as $(A \to B) \to C$ because $\to$ transformed two value-arenas into a term-arena.

Notice that the sequentiality between υ and κ in the definition of $\upsilon||\kappa$ is imposed by the fact that only initial moves (that is player answers) of B can justify initial moves of C in the definition of $A \to B \to C$. We can now define the composition of strategies by simply *hiding* the moves occurring in the value-arena B.

Definition 20 (Composition of strategies). *Given two strategies υ and κ respectively on the arenas $A \to B$ and $B \to C$, we define the* composed *strategy $\upsilon;\kappa$ on the arena $A \to C$, as the set $\{s_{|AC} \mid s \in \upsilon||\kappa\}$.*

This composition can be shown to be associative. This means that we can build a category $\mathcal{G}$, whose objects are value-arenas A, B and morphisms are strategies on $A \to B$.

5.1.4 Pairing

Taking an arena A and a strategy $\upsilon : B \to C$, we define the left identity pairing $(A \otimes \upsilon) : A \otimes B \to A \otimes C$ as the strategy formed by the plays $\{s \in P_{A\otimes B \to A\otimes C} \mid s_{|BC} \in \upsilon \text{ and } s_{|AA} \in \mathrm{id}_A\}$. The right identity pairing $(\upsilon \otimes A) : B \otimes A \to C \otimes A$ is defined dually. Then, taking two strategies $\upsilon : A \to B$ and $\kappa : C \to D$, we define their left pairing, written $\upsilon \otimes_l \kappa : A \otimes C \to B \otimes D$ as $(\upsilon \otimes C);(B \otimes \kappa)$.

However, a problem appears: the product $\otimes$ is not cartesian—it is not even monoidal. This is due to the fact that we could have defined in the same way a right pairing, as $(A \otimes \kappa);(\upsilon \otimes D)$, and there is no reason for left and right pairings to coincide. This is related to the fact that to define an operational semantics for the pairing $\langle M_1, M_2\rangle$ in RefML, one need to fix the order of evaluation: either we begin with evaluating M_1, or with evaluating M_2. This problem does not appear in call-by-name languages. Notice that $\otimes$ is still a *premonoidal product*, *i.e.* it is functorial in each of its component.

5.1.5 Single-Threaded Strategies

To solve this lack of a cartesian product, we now define a luff category[2] $\mathcal{G}_{STT}$ of $\mathcal{G}$, formed by *single-threaded strategies*. This category $\mathcal{G}_{STT}$ is better behaved, since it is a cartesian closed category. Here, we follow the presentation of Laird [Lai08].

Definition 21. *A strategy $\upsilon : A$ is said to be* total *if for all every initial move* $\mathrm{i} \in M_A$, *there exists a player answer* a *such that* $\mathrm{ia} \in \upsilon$.

2. a luff (as opposed to full) category is a subcategory which keeps the same objects.

Definition 22. *A strategy $\upsilon : A$ is said to be* single-threaded *if it is total and for all* ia$s \in \upsilon$*, there exists at most one move of s which is justified by* a*.*

Then, taking a play $s = \mathrm{ia}s'$, we define its *current thread* in the following way:
— $\mathbf{thr}(\mathrm{ia}s'm) = \mathrm{ia}m$ if m is justified by a,
— $\mathbf{thr}(\mathrm{ia}s'm) = \mathbf{thr}(\mathrm{ia}s')m$ if m is justified by i,
— $\mathbf{thr}(\mathrm{ia}s'm) = \mathbf{thr}(\mathrm{ia}s'')m$ if m is justified in s'. Then, s'' is defined as the longest prefix of s' such that the move which justifies m is in $\mathbf{thr}(\mathrm{ia}s'')$.

This definition extends to strategies: $\mathbf{thr}(\upsilon)$ is the set of the current threads of plays $s \in \upsilon$. It is in fact the largest single-threaded strategy contained in υ.

A play $s = \mathrm{ia}s'$ over a term-arena is said to be *thread-independent* if for all $s''p \sqsubseteq^{\mathrm{even}} s$, $\mathbf{thr}(s''p) = \mathbf{thr}(s'')p$. The set of thread-independent plays on an arena A is written P_A^{ti}. Once again, the definition of thread-independent plays extends straightforwardly to strategies.

Then, we can define a "shuffle" operation $(\cdot)^\dagger$ which transforms a single-threaded strategy υ on A into a thread-independent strategy $\upsilon^\dagger$, defined as $\{s \in \upsilon \mid |s| \leq 2\} \cup \{s \in P_A^{\mathrm{ti}} \mid \forall s' \sqsubseteq^{\mathrm{even}} s.\gamma(\mathbf{thr}(s')) \in \upsilon\}$. This operation satisfies the following equality:

Lemma 7: *For any thread-independent strategy s, $\mathbf{thr}(s)^\dagger = s$.*

We can now formally define the category $\mathcal{G}_{STT}$ whose objects are value-arenas and whose morphisms are single-threaded strategies. The composition of two single threaded strategies υ, κ is then defined as $\upsilon^\dagger; \kappa$.

Then, one can easily see that $\otimes$ is a cartesian product in $\mathcal{G}_{STT}$. Indeed, taking two single-threaded strategies $\upsilon : A \to B$ and $\kappa : A \to C$, one can define their pairing $\upsilon \otimes \kappa : A \to B \otimes C$ as the set of single-threaded plays s such that $s_{|A,B} \in \upsilon$ and $s_{|A,C} \in \kappa$. From s single-threaded, we can write it as iaos', where a is a player answer and o and opponent question. Then, o is either in the arena B or C, and there is no other opponent questions justified by a in s. Thus $\upsilon \otimes \kappa$ is indeed well-defined. Moreover, the operation $(\cdot)^\dagger$ is indeed a premonoidal functor between $\mathcal{G}_{STT}$ and $\mathcal{G}$, as proven in [Lai08].

5.1.6 Interpretation of Terms

Let A, B, C three value-arenas. Then there exists a bijection $\Lambda^B_{A,C}$ between strategies on $(A \otimes B) \to C$ and single-threaded strategies on $A \to (B \Rightarrow C)$, defined as

$$\upsilon \mapsto \{\varepsilon\} \cup \{m \star \mid m \in I_A\} \cup \{m \star ns \mid \langle m, n\rangle\, s \in \upsilon\}$$

Using it, we introduce the *evaluation function*, $\mathbf{ev}_{A,B} : ((A \Rightarrow B) \otimes A) \to B$ defined as

$$\mathbf{ev}_{A,B} \stackrel{def}{=} (\Lambda^A_{A\Rightarrow B,B})^{-1}(\mathrm{id}_{A\Rightarrow B}).$$

One can check that for every strategies $\upsilon : (A \otimes B) \to C$, we have $((\Lambda^B_{A,C}(\upsilon))^\dagger \otimes B); \mathbf{ev}_{B,C} = \upsilon$. This justifies the use of $\mathbf{ev}_{B,C}$ to interpret applications of terms. Taking a value-arena A, we thus have an adjunction $(A \Rightarrow \_)$ and $(\cdot)^\dagger \otimes A$ between $\mathcal{G}$ and $\mathcal{G}_{STT}$. As explained in [Lai08], the categories $\mathcal{G}, \mathcal{G}_{STT}$ together with the premonoidal functor $(\cdot)^\dagger$ is a *closed-Freyd category* [PT99]. We now have all the ingredients to define the interpretation of basic terms of a call-by-value λ-calculus:

$$\begin{array}{lcl} [\![x_1 : \tau_1 \dots x_n : \tau_n \vdash x_i : \tau_i]\!] & \stackrel{def}{=} & \pi_i \\ [\![\Gamma \vdash \lambda x.M : \sigma \to \tau]\!] & \stackrel{def}{=} & \Lambda^{[\![\sigma]\!]}_{[\![\Gamma]\!],[\![\tau]\!]}([\![\Gamma, x : \sigma \vdash M : \tau]\!])^\dagger \\ [\![\Gamma \vdash M\, N : \tau]\!] & \stackrel{def}{=} & \delta^\dagger_{[\![\Gamma]\!]} \,;\, ([\![\Gamma \vdash M : \sigma \to \tau]\!] \otimes_l [\![\Sigma; \Gamma \vdash N : \sigma]\!]) \,;\, \mathbf{ev}_{[\![\sigma]\!],[\![\tau]\!]} \end{array}$$

where $\delta_A : A \to A \otimes A$ is the usual diagonal coming from the cartesian product of $\mathcal{G}_{STT}$.

5.1.7 Game Semantics for RefML

We now briefly sketch the game semantics model of RefML developed by Murawski and Tzevelekos in [MT11b]. Previous works have studied references in game semantics, however they generally represent them in an indirect way, using two methods to read and write them. However, such presentations suffer from the problem of "bad variables", a problem first identified by Reynolds [Rey78]. Indeed, using this representation, there is inhabitants of reference types which are not "real" locations of the heap. This forbid to perform any test of equality of locations.

To build a model of "good" references only, nominal sets are used to represent strategies. In their model, plays are formed by *moves-with-heap*[3], that is a pair formed by a move m and a heap h, written m^h. Elements stored in theses heaps are not standard values, but rather moves. For ground types, this makes no difference, but for functional types τ, this means they can only store initial moves $\star$ of $M_{[\![\tau]\!]}$. The domain of these heaps contain all the locations that both the term and the context are aware of.

To define a play over an arena A, one must allow extra moves which are not from A, but from the set of moves $M_\phi \stackrel{def}{=} \biguplus_{\tau,\sigma} M_{[\![\tau \to \sigma]\!]}$. That is, plays over an arena A are formed by moves in $M_A \uplus s \cdot M_\phi$, where s is a new tag used to distinguish these moves. Such moves represent callbacks generated from higher-order references. This forces us to extend the notion of justification pointers, such that a move-with-heap m^h of a justification sequence s can be justified by a location l which appears in the heap h' of a previous move $n^{h'}$. Such moves are said to be *l-justified*.

To control the domain of heaps which can appears in move-with-heaps of a play, we define the set of available names of a justified sequence s, noted $\mathbf{Av}(s)$, in the following way:

— $\mathbf{Av}(\varepsilon) = \varnothing$
— $\mathbf{Av}(sm^h) = h^*(\mathbf{Av}(s) \cup \nu(m))$

3. Heaps are called *stores* in [MT11b].

where $h^*(S) \stackrel{def}{=} \bigcup_{j \leq 0} h^j(S)$ and $h^0(S) = S$, $h^{j+1}(S) = h(h^j(S)) \cap \mathrm{Loc}$.

Every play must satisfy the *frugality condition*: for any prefix $s'm^h$ of $\mathbf{s}$, $\mathrm{dom}(h) = \mathbf{Av}(s'm^h)$. Then, we enforce the definition of strategies to be nominally closed:

Definition 23 (Strategy). *A* strategy υ *on an arena* A *is a set of even-length plays on* A *satisfying:*
- *if* $so^h p^{h'} \in \upsilon$ *then* $s \in \upsilon$,
- *if* $s_1 p_1^{h_1}, s_2 p_2^{h_2} \in \upsilon$ *and* $s_1 \sim s_2$ *then* $s_1 p_1^{h_1} \sim s_2 p_2^{h_2}$,
- *if* $s_1 \in \upsilon$ *and* $s_1 \sim s_2$ *then* $s_2 \in \upsilon$.

The definition of the composition of strategies has to be modified, to control the way these extra moves from M_ϕ are used. This is done via the notion of *copycat triple*, that will be directly introduced in the setting of trace semantics later.

Finally, an important fact to notice is that there is no circularity issue with the definition of this model, unlike what happened for other operational or denotational model of higher-order references. For example, this was not the case of the previous nominal game model developed by Tzevelekos [Tze07, Tze08], where the circularity was resolved using tools from domain theory. Here, the circularity is avoided by cutting it, *i.e.* by representing strategies of higher-order functions stored in the heap just with their first moves, namely $\star$. Then, global conditions are imposed when composing strategies to control the way callbacks performed via these higher order references are done. The circularity would have appeared if heaps had stored plays rather than initial moves.

5.1.8 Full Abstraction of Game Semantics

We now state that the game semantics model for RefML presented in [MT11b] is fully abstract, namely that it captures exactly the contextual equivalence. Due to the presence of divergence, one must restrict the strategies we consider so that only *complete plays* are taking into account.

Definition 24. *A play* s *is said to be* complete *if every questions of* s *are answered. The set of complete plays of a strategy* υ *is written* $\mathbf{comp}(\upsilon)$.

Then, one can state the wanted theorem.

Theorem 9 (Full Abstraction of the Game Semantics): *Let* M_1, M_2 *two terms such that* $\Sigma; \Gamma \vdash M_1, M_2 : \tau$. *Then* $\Sigma; \Gamma \vdash M_1 \simeq_{ctx} M_1 : \tau$ *iff* $\mathbf{comp}([\![\Sigma; \Gamma \vdash M_1 : \tau]\!]) = \mathbf{comp}([\![\Sigma; \Gamma \vdash M_2 : \tau]\!])$.

5.2 Trace Semantics

We now introduce a semantics for RefML where denotations of terms are sets of *traces*. It is a variant of the work of Laird [Lai07] more amenable to a comparison with game semantics by taking track of type informations. Traces are used to represent all possible interactions of terms with contexts, and are generated by an *interactive reduction* which generalizes the small-step reduction of Figure 3.8 in two ways. First, it allows us to reduce open terms, and more specifically terms with free functional variables. That is, it becomes possible to reduce a term $K[f\,u]$, where f is a free variable of type $\sigma \to \tau$, either to $K[v]$ for any value v of type τ, if τ is a ground type; or to $K[f']$ for any variable f' free in K if τ is a functional type. This is used to reduce $\lambda x.M$ when x is of functional type, so that we can deal with x symbolically.

Second, to be able to generate all possible executions, we need to keep track of values disclosed to contexts, namely location—so that a context can set arbitrary values in it—or λ-abstraction—so that a context can call it at any time when it takes control back.

Notice that this reduction is history-independent, *i.e.* the reduction of a callback $K[f\,v]$ does not depend on the possible previous occurrences of $K'[f\,v]$ in the reduction. This is due to the fact that our language has references, so that contexts can keep track of the number of times their functional arguments provided to the term are called, and thus give each time a different answer. This corresponds to the fact that strategies for RefML are not innocent. Due to this last point, the definition of a similar interactive reduction for a pure language would be definitely more tedious.

5.2.1 Game-like Definitions

We start introducing traces following the usual presentation of game semantics, mimicking the definitions of the previous section. The notion corresponding to a game move is called here an *action*. Actions are formed over ground values and variables, used to represent higher-order values. These variables, of functional type, are called *opponent* and *player* name pointers. opponent name pointers represent higher-order values provided by contexts (*i.e.* opponent) to terms (*i.e.* player), while it is the opposite for player name pointers. The set of name pointers, which is a subset of Var is written $\mathbb{P}$, and the set of free name pointers (*i.e.* its supports) of an element X is written $\nu_{\mathbb{P}}(X)$. There are four kinds of *basic* actions:

- a question of the term (here named Player) via a name pointer x with argument v, represented by the action $\bar{x}\,\langle v\rangle$,
- a question of the context (here named Opponent) via a name pointer x with argument v, represented by the action $x\,\langle v\rangle$,
- an answer by Player of the value v, represented by the action $\langle \bar{v}\rangle$,
- an answer by Opponent of the value v, represented by the action $\langle v\rangle$.

A name pointer y appearing as an argument of a player question $\bar{x}\,\langle y\rangle$ or in a player answer $\langle \bar{y}\rangle$ is called a *player name pointer*. In the same way, a name pointer y appearing

as an argument of an opponent question $x\,\langle y\rangle$ or in an opponent answer $\langle y\rangle$ is thus called an *opponent name pointer*. This means that being an opponent or a player name pointer depends on the action, and is not inherent to the name pointer. The set of free player and opponent name pointers (*i.e.* their supports) of an element X formed by actions is respectively written $\nu^P_{\mathbb{P}}(X)$ and $\nu^O_{\mathbb{P}}(X)$. We also consider special opponent questions $?\,\langle v\rangle$ where the name-pointer which is questioned is omitted, which represent initial moves.

Actions a are defined as pairs (a, ξ) of a basic action a and a tag ξ, which is a word over the alphabet $\{\ell, r, s\}$, where the concatenation is written $\xi \cdot \xi'$ and the empty word is written ε. Such tags are used to indicate to which set (*i.e.* trace-arena) an basic action belongs to. This is useful to avoid the use of disjoint unions (as coproduct) and the corresponding injections $\mathbf{in_l}, \mathbf{in_r}$ which are usually used in game semantics. In our setting, such injections are represented respectively by tags beginning with ℓ and r, while s is used to represent actions corresponding to functions stored in heaps.

Player, Opponent and initial actions are respectively written p, o and i. The labeling of actions (*i.e.* the fact they are player or opponent and question or answer actions) is hard-wired, while labeling of moves in game semantics depends on the underlying arena the moves belong to (via the function λ). As we will see, this complicates some definitions (arrow arenas and restrictions of traces to a given arena) where we need to change the labeling of actions.

We define the operation $a^\perp$ as the operation which simply transform an opponent action into the corresponding player action, and vice-versa (leaving the tag unchanged). It is extended to sets and relations of actions. Then, we introduce the notion of *trace arenas*, which are simply pairs $(\mathfrak{M}, \mathfrak{I}, \vdash)$ of a set of actions $\mathfrak{M}$, a set of initial actions $\mathfrak{I} \subseteq \mathfrak{M}$ and a justification relation $\vdash \subseteq \mathfrak{M} \times \mathfrak{M}\backslash\mathfrak{I}$. Following the correspondence with game semantics, we define: *value-arenas* (resp. *term-arenas*) as arenas whose initial actions are player answers (resp. opponent question).

From two trace value-arenas $\mathfrak{A}, \mathfrak{B}$ we construct the value arenas $\mathfrak{A} \otimes \mathfrak{B}$ and $\mathfrak{A} \Rightarrow \mathfrak{B}$ and the term arena $\mathfrak{A} \to \mathfrak{B}$ in Figure 5.1. In the definition of $\mathfrak{A} \to \mathfrak{B}$, the symbol ? represent the initial opponent question where no name pointer is present. To each type τ, we associate a trace value-arena $[\tau]$ as:

— $[\mathrm{Unit}] \stackrel{def}{=} (\mathfrak{M}_{\mathrm{Unit}}, \mathfrak{M}_{\mathrm{Unit}}, \varnothing)$ where $\mathfrak{M}_{\mathrm{Unit}} \stackrel{def}{=} \{\bar{\langle\rangle}\}$,

— $[\mathrm{Int}] \stackrel{def}{=} (\mathfrak{M}_{\mathrm{Int}}, \mathfrak{M}_{\mathrm{Int}}, \varnothing)$ where $\mathfrak{M}_{\mathrm{Int}} \stackrel{def}{=} \{\langle \bar{n}\rangle \mid n \in \mathbb{Z}\}$,

— $[\mathrm{ref}\,\tau] \stackrel{def}{=} (\mathfrak{M}_{\mathrm{ref}\,\tau}, \mathfrak{M}_{\mathrm{ref}\,\tau}, \varnothing)$ where $\mathfrak{M}_{\mathrm{ref}\,\tau} \stackrel{def}{=} \{\bar{l} \mid l \in \mathrm{Loc}_\tau\}$,

— $[\sigma \to \tau] \stackrel{def}{=} [\sigma] \Rightarrow [\tau]$.

To relate actions to the evolution of the heap, we introduce *actions-with-heap* on a trace-arena $\mathfrak{A}$, *i.e.* pairs (a, h) of an action $a \in \mathfrak{M}_{\mathfrak{A}}$ and a *functional-free heap* h, that is a heap where stored higher-order values are represented by name pointers. An action-with-heap (a, ξ, h) is said to introduce the name pointer x if either a is of the form $\bar{y}\,\langle x\rangle, y\,\langle x\rangle, \langle\bar{x}\rangle$ or $\langle x\rangle$, or if x is in the co-domain of h (written $\mathrm{codom}(h)$). In the latter case, we say that x is l-introduced when $h(l) = x$.

$$
\begin{array}{lcl}
\mathfrak{I}_{\mathfrak{A}\otimes\mathfrak{B}} & \stackrel{def}{=} & \ell\cdot\mathfrak{I}_{\mathfrak{A}}\times r\cdot\mathfrak{I}_{\mathfrak{B}}\\
\mathfrak{M}_{\mathfrak{A}\otimes\mathfrak{B}} & \stackrel{def}{=} & \mathfrak{I}_{\mathfrak{A}\otimes\mathfrak{B}}\uplus\ell\cdot(\mathfrak{M}_{\mathfrak{A}}\backslash\mathfrak{I}_{\mathfrak{A}})\uplus r\cdot(\mathfrak{M}_{\mathfrak{B}}\backslash\mathfrak{I}_{\mathfrak{B}})\\
\vdash_{\mathfrak{A}\otimes\mathfrak{B}} & \stackrel{def}{=} & \{((\ell\cdot\mathrm{i}_{\mathfrak{A}},r\cdot\mathrm{i}_{\mathfrak{B}}),\ell\cdot a)\mid\mathrm{i}_{\mathfrak{A}}\vdash_{\mathfrak{A}}a\}\uplus\{((\ell\cdot\mathrm{i}_{\mathfrak{A}},r\cdot\mathrm{i}_{\mathfrak{B}}),r\cdot a)\mid\mathrm{i}_{\mathfrak{B}}\vdash_{\mathfrak{B}}a\}\\
& & \uplus(\ell\cdot\vdash_{\mathfrak{A}_{|(\mathfrak{M}_{\mathfrak{A}}\backslash\mathfrak{I}_{\mathfrak{A}})^2}})\uplus(r\cdot\vdash_{\mathfrak{B}_{|(\mathfrak{M}_{\mathfrak{B}}\backslash\mathfrak{I}_{\mathfrak{B}})^2}})\\
\mathfrak{I}_{\mathfrak{A}\Rightarrow\mathfrak{B}} & \stackrel{def}{=} & \{\langle\bar{x}\rangle\mid x\in\mathbb{P}\}\\
\mathfrak{M}_{\mathfrak{A}\Rightarrow\mathfrak{B}} & \stackrel{def}{=} & \mathfrak{I}_{\mathfrak{A}\Rightarrow\mathfrak{B}}\uplus\ell\cdot(\mathfrak{M}_{\mathfrak{A}}\backslash\mathfrak{I}_{\mathfrak{A}})^{\perp}\uplus\{(y\,\langle u\rangle,\ell\cdot\xi)\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}},\ y\in\mathbb{P}\}\uplus r\cdot\mathfrak{M}_{\mathfrak{B}}\\
\vdash_{\mathfrak{A}\Rightarrow\mathfrak{B}} & \stackrel{def}{=} & \{(\langle\bar{x}\rangle,(x\,\langle u\rangle,\ell\cdot\xi))\mid x\in\mathbb{P},(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}}\}\\
& & \uplus\{((x\,\langle u\rangle,\ell\cdot\xi),\ell\cdot a)\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}},a\in\mathfrak{M}_{\mathfrak{A}},(\langle\bar{u}\rangle,\xi)\vdash_{\mathfrak{A}}a\}\\
& & \uplus\{((x\,\langle u\rangle,\ell\cdot\xi),r\cdot\mathrm{i}_{\mathfrak{B}})\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}}\}\uplus(\ell\cdot\vdash_{\mathfrak{A}_{|(\mathfrak{M}_{\mathfrak{A}}\backslash\mathfrak{I}_{\mathfrak{A}})^2}})^{\perp}\uplus(r\cdot\vdash_{\mathfrak{B}})\\
\mathfrak{I}_{\mathfrak{A}\to\mathfrak{B}} & \stackrel{def}{=} & \{(?\,\langle u\rangle,\ell\cdot\xi)\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}}\}\\
\mathfrak{M}_{\mathfrak{A}\to\mathfrak{B}} & \stackrel{def}{=} & \mathfrak{I}_{\mathfrak{A}\to\mathfrak{B}}\uplus\ell\cdot(\mathfrak{M}_{\mathfrak{A}}\backslash\mathfrak{I}_{\mathfrak{A}})^{\perp}\uplus r\cdot\mathfrak{M}_{\mathfrak{B}}\\
\vdash_{\mathfrak{A}\to\mathfrak{B}} & \stackrel{def}{=} & \{((?\,\langle u\rangle,\ell\cdot\xi),r\cdot\mathrm{i}_{\mathfrak{B}})\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}},\mathrm{i}_{\mathfrak{B}}\in\mathfrak{I}_{\mathfrak{B}}\}\uplus(\ell\cdot\vdash_{\mathfrak{A}_{|(\mathfrak{M}_{\mathfrak{A}}\backslash\mathfrak{I}_{\mathfrak{A}})^2}})^{\perp}\uplus(r\cdot\vdash_{\mathfrak{B}})\\
& & \uplus\{((?\,\langle u\rangle,\ell\cdot\xi),\ell\cdot a)\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{A}},a\in\mathfrak{M}_{\mathfrak{A}},(\langle\bar{u}\rangle,\xi)\vdash_{\mathfrak{A}}a\}
\end{array}
$$

Figure 5.1: Definition of compound arenas

Such actions which l-introduce name pointers, called *ϕ-actions*, correspond to callbacks coming from disclosed locations storing functions. They are living in the set $\mathfrak{M}_\phi$ defined as $\bigcup_{\tau,\tau'}\mathfrak{M}_{[\tau\to\tau']}$. Using it, we define the set $\mathrm{Trace}_{\mathfrak{A}}$ over a trace-arena $\mathfrak{A}$ as the set of sequences T of actions-with-heap on $\mathfrak{M}_{\mathfrak{A}}\uplus(s\cdot\mathfrak{M}_\phi)$ such that for each name pointer x in T, x is introduced by at most one action-with-heap in T. $\mathrm{Trace}_{\mathfrak{A}}$ can be seen as a nominal set over Loc and $\mathbb{P}$. We write $T'\sqsubseteq T$ when T' is a prefix of T, and $T'\sqsubseteq^{\mathrm{even}}T$ when furthermore T' is of even length. Taking an action (a,h) of T, we define $T_{\leq(a,h)}$ as the prefix of T whose last action is (a,h).

We say that a trace $T\in\mathrm{Trace}_{\mathfrak{A}}$ is *justified* if every name pointer x in T is introduced in T. Then, we define the *depth* of an action (a,h) in a trace T, written $\mathbf{depth}_T(a,h)$ as the difference between the number of questions and the number of answers of T_1, where $T=T_1\cdot(a,h)\cdot T_2$.

Definition 25. *Let $(a_1,h_1),(a_2,h_2)$ two actions-with-heap such that (a_1,h_1) appears before (a_2,h_2) in a trace T. We say that (a_1,h_1) justifies (a_2,h_2) when:*

- *a_2 is an answer and a_1 is the latest question of T appearing before a_1* s.t. *$\mathbf{depth}_T(a_2,h_2)=\mathbf{depth}_T(a_1,h_1)+1$,*
- *or a_2 is a question $\bar{x}\,\langle u\rangle$ or $x\,\langle u\rangle$, and (a_1,h_1) is the first action introducing x, so a_1 is either equal to $\langle\bar{x}\rangle,\langle x\rangle,\bar{y}\,\langle x\rangle,y\,\langle x\rangle$ or $x\in\mathrm{codom}(h_1)$. In the latter case, we say that x is l-justified when $h_1(l)=x$.*

A question of T which does not justify any answer is said to be pending.

We define the set of *available locations* of a trace T, written $\mathbf{Av}(T)$, as $\mathbf{Av}(\varepsilon) \stackrel{def}{=} \varnothing$ and $\mathbf{Av}(T \cdot (a,h)) \stackrel{def}{=} h^*(\mathbf{Av}(T) \cup \nu_L(a))$, where $h^*(S) \stackrel{def}{=} \bigcup_{j \leq 0} h^j(S)$ with $h^0(S) \stackrel{def}{=} S$ and $h^{j+1}(S) \stackrel{def}{=} h(h^j(S)) \cap \mathrm{Loc}$.

A justified trace over $\mathfrak{A}$ is said to be *legal* if only its first action-with-heap is in $\mathfrak{I}_{\mathfrak{A}}$ and it alternates between player and opponent actions, and is said to be a *play* if it is furthermore *frugal*, *i.e.* for all $T' \cdot (a,h) \sqsubseteq T$, $\mathrm{dom}(h) = \mathbf{Av}(T' \cdot (a,h))$. The set of legal traces (resp. plays) on $\mathfrak{A}$ is written $\mathfrak{L}_{\mathfrak{A}}$ (resp. $\mathfrak{P}_{\mathfrak{A}}$).Using all this definitions, we can finally introduce the notion of *trace-strategy* used to define the denotation of terms.

Definition 26. *A* trace-strategy $\mathfrak{s}$ *over a trace arena* $\mathfrak{A}$ *as a set of even-length plays on* $\mathfrak{A}$ *such that:*

— *If* $T \cdot (o,h) \cdot (p,h') \in \sigma$ *then* $T \in \sigma$.
— *If* $T \in \sigma$ *and* $T \sim T'$ *then* $T' \in \sigma$.
— *If* $T_1 \cdot (p_1,h_1)$ *and* $T_2 \cdot (p_2,h_2)$ *are in* σ *and* $T_1 \sim T_2$, *then* $T_1 \cdot (p_1,h_1) \sim T_2 \cdot (p_2,h_2)$.

5.2.2 A correspondence between Traces and Plays

There is a direct correspondence between actions introduced in this section and moves in game semantics as introduced before, where we suppose that injections coming from coproducts are also represented by tags. It is obtained by transforming questions $\bar{x}\,\langle v \rangle$ and $x\,\langle v \rangle$ into v, transforming answers $\langle \bar{v} \rangle$ and $\langle v \rangle$ into v, and then transforming every remaining name pointers into $\star$, the initial move of game arenas for functional types [4]. The function which performs this two-step translation is written θ, which transforms actions from a trace-arena $\mathfrak{A}$ to moves to the corresponding game-arena A, leaving tags unchanged. The labeling function is then defined straightforwardly.

Then, we extend this correspondence between justified traces and justified sequences of game semantics, with the extra property that justified traces are always well-bracketed. More precisely, we first extend straightforwardly the function θ from actions-with-heaps to move-with-heaps, which transform name pointers stored in heaps into the move $\star$, and then pointwisely from traces to sequences of moves. So we define a function Θ which transforms a justified trace T on an arena $\mathfrak{A}$ to a sequence of moves $\theta(T)$ on the corresponding arena A, and such that for two actions-with-heaps (a_1,h_1), (a_2,h_2) of T, there is a pointer from $\theta(a_2,h_2)$ to $\theta(a_1,h_1)$ when (a_2,h_2) is justified by (a_1,h_1). Notice that two traces which are $\mathbb{P}$-nominal equivalent give rise to the same sequence of moves.

Extending Θ to sets of traces, it is direct that $\Theta(\mathfrak{s})$ is a game-strategy on an arena A when $\mathfrak{s}$ is a trace-strategy on the corresponding arena $\mathfrak{A}$.

4. We do not need to transform the symbol "?" since it is automatically removed, appearing only in $?\,\langle v \rangle$.

5.2.3 Interactive Reduction

We now introduce an *interactive reduction* which generates traces from terms, representing their interactions with any possible applicative contexts $K[\bullet_{\tau,\xi}]$, where the symbol $\bullet$, representing "hole" (*i.e.* a pending question), is tagged with a type τ and a tag ξ, representing the type and the arena of the expected answer which will fill the hole. This reduction is defined on "stacks" $(M, \tau, \xi) \cdot \overrightarrow{(K_i, \tau_i, \xi_i)}$ formed by a term M and contexts $\overrightarrow{K_i}$ for *player configurations*, or on stacks $\overrightarrow{(K_i, \tau_i, \xi_i)}$ for *opponent configurations*. Such elements of the stacks comes also with a type τ and a tag ξ. The empty stack is simply written $\Diamond$, and we write F for the top element of a stack, which can either be a term M, a context $K[\bullet]$ or the empty stack $\Diamond$. When Player provides a higher-order value to Opponent, either via a callback (*i.e.* a question) or directly when reducing to a λ-abstraction (*i.e.* an answer), they are stored in environments γ, which are partial maps from $\mathbb{P}$ to Val. Then Opponent can interrogate what is stored in γ, by asking a question. Opponent only provides opponent name pointers to represent higher-order values. They are stored in a set $\mathcal{I} \subseteq \mathbb{P}$. They can also be interrogated by player.

To represent disclosure of locations, we use a set D which grows as the term or the context discloses new locations. To determine which locations are disclosed when a value v is played with a heap h, we define a function $\mathrm{discl}(v, h, D)$ as

— $h^*(D \cup \{l\})$ if v is a location l,
— $h^*(D)$ otherwise.

We simply write $\mathrm{discl}(h, D)$ for $h^*(D)$. The complement of the set D is written $\overline{D}$.

Then, the interactive reduction is defined between player configurations $\langle M \cdot \overrightarrow{K_i}, \gamma, \mathcal{I}, h, D\rangle$ and opponent configurations $\langle K[\bullet] \cdot \overrightarrow{K_i}, \gamma, \mathcal{I}, h, D\rangle$. It is defined in Figure 5.2.

Let us explain the different rules:

— The rule **P-Intern** allows us to perform the usual (operational) reduction of terms. Notice that it uses the non-deterministic reduction $\mapsto_{nd}$ rather than the usual $\mapsto$, in order to be exhaustive w.r.t. names of locations created.
— The rules **P-AnsG** and **P-Ans** represent player answers. If the answer is a ground value, then if it is a location it is put in D', otherwise it is simply forgotten by Opponent since it has no meaning to interrogate it. Otherwise, it is a higher-order value, which is thus stored in γ'. Player can also disclose indirectly either new locations or new higher-order values via the already disclosed locations. The new disclosed locations are caught via $\mathrm{discl}(v, h, D)$, while disclosed higher-order values live in $\mathrm{dom}(h_{\mathrm{fn}}) \cap \mathcal{D}'$ (recall that h_{fn} is the subheap of h which stores higher-order values), so that they are replaced by fresh player name pointers in h' and γ is updated consequently.
— The rules **O-AnsG** and **O-Ans** represent opponent answers. When Opponent answers a location, it cannot be one which is private to Player. This explain the condition "v is ground and not in $\mathrm{dom}(h) \cap \overline{D}$" in the definition of **O-AnsG**. And when it should be a higher-order value, it is simply represented by a *fresh* opponent

P-Intern $\langle (M,\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \longrightarrow \langle (M',\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h',D\rangle$
(when $(M,h)\mapsto_{nd}(M',h')$)

P-AnsG $\langle (v,\iota,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\langle\bar{v}\rangle,r\cdot\xi,h'_{|D'})} \langle\overrightarrow{K_i},\gamma',\mathcal{I},h',D'\rangle$
(v of type ι, $\gamma'=\gamma\cdot\overrightarrow{[x_i\hookrightarrow(h(l_i),\tau_i,s)]}$)

P-Ans $\langle (v,\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\langle\bar{x}\rangle,r\cdot\xi,h'_{|D'})} \langle\overrightarrow{K_i},\gamma',\mathcal{I},h',D'\rangle$
(x fresh, $\gamma'=\gamma\cdot[x\hookrightarrow(v,\tau,r\cdot\xi)]\cdot\overrightarrow{[x_i\hookrightarrow(h(l_i),\tau_i,s)]}$)

P-QuestG $\langle (K[x\,v],\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\bar{x}\langle v\rangle,\ell\cdot\xi',h'_{|D'})} \langle (K[\bullet_{\iota,\xi'}],\tau,\xi)\cdot\overrightarrow{K_i},\gamma',\mathcal{I},h',D'\rangle$
($(x,\iota\to\sigma,\xi')\in\mathcal{I}$, v of type ι, the x_i fresh, $\gamma'=\gamma\cdot\overrightarrow{[x_i\hookrightarrow(h(l_i),\tau_i,s)]}$)

P-Quest $\langle (K[x\,v],\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\bar{x}\langle y\rangle,\ell\cdot\xi',h'_{|D'})} \langle (K[\bullet_{\sigma',\xi'}],\tau,\xi)\cdot\overrightarrow{K_i},\gamma',\mathcal{I},h',D'\rangle$
($(x,\sigma\to\sigma',\xi')\in\mathcal{I}$, y fresh, $\gamma'=\gamma\cdot[y\hookrightarrow(v,\sigma,\ell\cdot\xi')]\cdot\overrightarrow{[x_i\hookrightarrow(h(l_i),\tau_i,s)]}$)

in all P-rules: $D'=\mathrm{discl}(v,h,D)$ and $h'=h\overrightarrow{[l_i\hookrightarrow x_i]}$ with the x_i fresh
where l_i ranges over $\mathrm{dom}(h_{\mathrm{fn}})\cap D'$ with $l_i\in\mathrm{Loc}_{\tau_i}$)

O-AnsG $\langle (K[\bullet_{\iota,\xi'}],\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\langle v\rangle,r\cdot\xi',h'_{|D'})} \langle (K[v],\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
(v of type ι, $v\notin\mathrm{dom}(h)\cap\overline{D}$, $\mathcal{I}'=\mathcal{I}\cdot\overrightarrow{(h(l_i),\tau_i,s)}$)

O-Ans $\langle (K[\bullet_{\sigma,\xi'}],\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\langle x\rangle,r\cdot\xi',h'_{|D'})} \langle (K[x],\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
(x fresh, $\mathcal{I}'=\mathcal{I}\cdot(x,\sigma,r\cdot\xi')\cdot\overrightarrow{(h(l_i),\tau_i,s)}$)

O-QuestG $\langle\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(x\langle v\rangle,\ell\cdot\xi,h'_{|D'})} \langle (u\,v,\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
($\gamma(x)=(u,\iota\to\tau,\xi)$, v of type ι, $v\notin\mathrm{dom}(h)\cap\overline{D}$, $\mathcal{I}'=\mathcal{I}\cdot\overrightarrow{(h(l_i),\tau_i,s)}$)

O-Quest $\langle\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(x\langle y\rangle,\ell\cdot\xi,h'_{|D'})} \langle (u\,y,\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
($\gamma(x)=(u,\sigma\to\tau,\xi)$, y fresh, $\mathcal{I}'=\mathcal{I}\cdot(y,\tau,\ell\cdot\xi))\cdot\overrightarrow{(h(l_i),\tau_i,s)}$)

in all O-Rules: $D'=\mathrm{discl}(v,h',D)$, l_i ranges over $\mathrm{dom}(h'_{\mathrm{fn}})\cap\mathcal{D}'$ *s.t.* $l_i\in\mathrm{Loc}_{\tau_i}$,
$h'_{|\overline{D'}}=h_{|\overline{D}}$ and $h'_{|D'}$ is closed and functional-free)

Figure 5.2: Definitions of the interaction semantics

name pointer. When answering, Opponent can also provide new name pointers via the disclosed part of the heap. Those ones live in the disclosed part of h'_{fn}, *i.e.* in $\text{codom}(h'_{\text{fn}|\mathcal{D}'})$. An important point here is that the context can also disclose *indirectly* new locations via the already disclosed ones. This explains the great liberty Opponent has when it extends h to h' with new disclosed locations. It must however satisfy the equation $D' = \text{discl}(h', D)$, so that it cannot add as many new (necessarily disclosed) locations as it wants.

— The rules **P-QuestG** and **P-Quest** represent player questions. That is, player interrogates an opponent name pointer of $\mathcal{I}$ with a value. We find here the same behavior about indirect disclosure we have encountered for the Player answers.

— Finally, the rules **O-QuestG** and **O-Quest** represent opponent questions. In that case, Opponent adds to the current execution stack a new thread, corresponding to the higher-order values stored in γ. Here again, we find the same behavior about indirect disclosure we have encountered for the Opponent answers.

This reduction is highly non-deterministic, since we consider the interactions with all possible contexts. Moreover, the choice of name pointers and locations is also non-deterministic (even in the rule **P-Intern** with the use of $\mapsto_{nd}$) so that the reduction do not perform arbitrary choice. One can easily check that the following properties are preserved by the interactive reductions:

— the stack and the codomain of γ do not contain any player name pointer,

— the higher-order values stored in the disclosed part of the heap are name pointers contained in $\mathcal{I}$ or $\text{dom}(\gamma)$,

— the heap h and its disclosed part are both closed.

We say that a trace T is *generated* by a configuration C if it can be written as a sequence $(a_1, h_1) \cdots (a_n, h_n)$ of actions-with-heap such that $C \xrightarrow{a_1,h_1} C_1 \xrightarrow{a_2,h_2} C_2 \ldots \xrightarrow{a_n,h_n} C_n$, and we write $C \xrightarrow{T} C_n$. We can see that it is indeed a trace due to the freshness conditions in the rules P-Ans, P-Quest, O-Ans and O-Quest. The set of traces generated by C is written $\mathbf{Tr}(C)$. Notice that such traces are not in general justified. This is due to the fact that name-pointers of C are not introduced. As we will see, the initial (opponent question) action is missing. Moreover, $\mathbf{Tr}(C)$ is not nominally-closed: if π is a permutation such that $\pi(a) \neq a$ for $a \in \nu_{\mathbb{A}}(C)$, then taking $T \in \mathbf{Tr}(C)$, $\pi * T$ is not in general in $\mathbf{Tr}(C)$. This is useful to distinguish sets $\mathbf{Tr}\langle x, \gamma, \mathcal{I}, h, D\rangle$ and $\mathbf{Tr}\langle x', \gamma, \mathcal{I}, h, D\rangle$ for different opponent name pointers $x, x' \in \mathcal{I}$, or to distinguish $\mathbf{Tr}\langle l, \gamma, \mathcal{I}, h, D\rangle$ and $\mathbf{Tr}\langle l', \gamma, \mathcal{I}, h, D\rangle$ for locations $l, l' \in \text{dom}(h)$.

Example 1: *let us consider the term M_{inc} defined as*

$$\texttt{let x} = \ \text{ref}\,\texttt{0}\ \texttt{in let f} = \text{ref}\,(\lambda\_.\texttt{x} := !\texttt{x} + \texttt{1})\ \texttt{in}\ \lambda\texttt{g.g f}; !\texttt{f}(); !\texttt{x}$$

The reader can check that one possible trace of this term starting from the initial configuration $\langle M_{inc}, \varepsilon, \varepsilon, \varepsilon, \varepsilon\rangle$ is

$$(\langle \bar{a}\rangle, \varepsilon) \cdot (a\langle b\rangle, \varepsilon) \cdot (\bar{b}\langle l\rangle, [l \hookrightarrow c]) \cdot (c\langle ()\rangle, [l \hookrightarrow c]) \cdot (\left\langle \bar{()} \right\rangle, [l \hookrightarrow c]) \cdot$$
$$(\langle ()\rangle, [l \hookrightarrow d]) \cdot (\bar{c}\langle ()\rangle, [l \hookrightarrow d]) \cdot (\langle ()\rangle, [l \hookrightarrow d]) \cdot (\left\langle \bar{1} \right\rangle, [l \hookrightarrow d]).$$

Intuitively, this trace corresponds to the interaction with the context defined as $\bullet(\lambda \mathtt{c}.!\mathtt{c}(); \mathtt{c} := \lambda\_.())$. *Notice that the value stored in* x *is incremented not by the call to* $!\mathtt{f}()$ *in* M_{inc}, *but by the call made by the context after the disclosure of* f *via* g. *Indeed, the call to* $!\mathtt{f}()$ *corresponds to the call to* $\lambda\_.()$ *since the context has modified the function stored in* f.

Traces generated by a configuration are not altered when the stack is extended, as shown by the following lemma:

Lemma 8: *Suppose that* $\langle F \cdot \overrightarrow{K_i}, \gamma, \mathcal{I}, h, D\rangle \xrightarrow{T} \langle F' \cdot \overrightarrow{K'_j}, \gamma', \mathcal{I}', h', D'\rangle$ *then* $\langle F \cdot \overrightarrow{K_i} \cdot \overrightarrow{K''_l}, \gamma, \mathcal{I}, h, D\rangle \xrightarrow{T} \langle F' \cdot \overrightarrow{K'_i} \cdot \overrightarrow{K''_l}, \gamma', \mathcal{I}', h', D'\rangle$.

An important point to notice is that the tags of actions of traces generated by the interactive reduction can be infer knowing just the tag of initial actions, as shown by the following lemma:

Lemma 9: *Let* $(a_1, \xi_1, h_1), (a_2, \xi_2, h_2)$ *two actions of a trace* T, *such that* (a_2, ξ_2, h_2) *is justified by* (a_1, ξ_1, h_1). *Then:*

- *If* a_2 *is an answers, then* $\xi_1 = \ell \cdot \xi$ *and* $\xi_2 = r \cdot \xi$,
- *If* a_2 *is a question, then*
  - *if* a_2 *is l-justified by* a_1, $\xi_2 = \ell \cdot s \cdot \xi_1$,
  - *otherwise* $\xi_2 = \ell \cdot \xi_1$.

This justifies the fact that in the following, we often omit tags when considering actions and traces.

5.2.4 Interpretation of Terms

Taking a term M *s.t.* $\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau$, we now define a trace strategy associated to this term, which is generated using the interactive reduction. To do so, we first define the list of opponent name pointers $\mathcal{I}_\xi^{\Gamma_f}$ as $\mathcal{I}_\xi^{\diamond} = \varepsilon$ and $\mathcal{I}_\xi^{(x:\tau),\Gamma} \stackrel{def}{=} (x, \tau, \ell \cdot \xi) \cdot \mathcal{I}_{r\cdot\xi}^{\Gamma}$.

Definition 27 (Trace Semantics). *Let* M *a term such that* $\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau$. *We define* $[\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau]$ *as the set of even-length traces belonging to the nominal closure over* Loc *and* $\mathbb{P}$ *of*

$$\left\{ \left(? \left\langle \Sigma, \overrightarrow{\gamma_g(x_i)}, \mathcal{I} \right\rangle, \ell, h\right) \cdot \mathbf{Tr}\langle(\gamma_g(M), \tau, \varepsilon), \varepsilon, \mathcal{I}, h, D\rangle \;\middle|\; \begin{array}{l} \gamma_g : \Gamma_g \to \mathrm{Val} \\ h \in \mathbf{Cl}(\Sigma, \mathrm{codom}(\gamma_g)) \\ \mathrm{codom}(h_{\mathrm{fn}}) \subseteq \mathbb{P} \end{array} \right\}$$

where $\overrightarrow{x_i}$ *ranges overs the variables of* Γ_g, $D = \mathrm{dom}(h)$ *and* $\mathcal{I} = \mathcal{I}_\xi^{\Gamma_f} \cdot \overrightarrow{(h(l_i), \tau_i, s)}$ s.t. l_i *ranges over* $\mathrm{dom}(h_{\mathrm{fn}})$ *and* $l_i \in \mathrm{Loc}_{\tau_i}$.

Recall that h_{fn} is the subheap of h formed by higher-order references. We reason up to nominal equivalence of $\nu_{\text{Loc}}(M)$ (*i.e.* Σ) and $\nu_{\mathbb{P}}(M)$ (*i.e.* Γ_f) so that $[\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau]$ is nominally closed. Moreover, the substitution γ_g of ground variables of M introduce new locations for variables of type ref τ, so we must consider them in h to have a closed heap. Finally, h is functional-free (*i.e.* $\text{codom}(h_{\text{fn}}) \subseteq \mathbb{P}$), so that for any location $l \in \text{Loc}_\phi$, if $h(l)$ or $h'(l)$ is defined, it has to store an opponent name pointer.

Let us define $[\Gamma]$ as $[\tau_1] \otimes \ldots \otimes [\tau_m]$ when $\Gamma = (x_1 : \tau_1) \ldots (x_m : \tau_m)$ and $[\Sigma]$ is defined as $[\text{ref}\, \tau_1] \otimes \ldots \otimes [\text{ref}\, \tau_n]$ when $\Sigma = (l_1 : \tau_1) \ldots (l_n : \tau_n)$.

Theorem 10: *Let M a term* s.t. $\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau$, *then* $[\Sigma; \Gamma_g, \Gamma_f \vdash M : \tau]$ *is a trace-strategy over the arena* $[\Sigma] \otimes [\Gamma_g, \Gamma_f] \to [\tau]$.

A trace $T \in \mathbf{Tr}\langle M \cdot \overrightarrow{K_i}, \gamma, \mathcal{I}, h, D\rangle$ is said to be *complete* if the number of answers occurring in the trace is greater than its number of questions plus the length of the sequence $\overrightarrow{K_i}$. The set of complete traces of a configuration C is written $\mathbf{comp}(\mathbf{Tr}(C))$. This is useful to give a correspondence with contextual equivalence via game semantics, as we will see in the next part.

Notice that we introduce $[\Sigma; \Gamma \vdash M : \tau]$ to get an exact correspondence with the denotation of terms in game semantics. However, this notion has many drawbacks, since it is not stable by reduction. More precisely, if $\Sigma; \Gamma \vdash M : \tau$ and $(M, h) \mapsto^* (M', h')$, then h' has possibly new locations which have not been disclosed to the context, so we cannot relate $[\Sigma'; \Gamma \vdash M' : \tau]$ to $[\Sigma'; \Gamma \vdash M : \tau]$ since this semantics suppose that all the locations have been disclosed. Moreover, the traces of $[\Sigma'; \Gamma \vdash M : \tau]$ all begin with no (functional) values disclosed to the context, *i.e.* the environment γ is empty in the starting configuration. So when $(M, h) \mapsto^* (K[f\, v], h')$ we cannot relate easily the denotation of M to the denotation of $K[\bullet]$ and v. This problem will be solved in Section 6.3 where we introduce *Kripke trace semantics* to distinguish between private and disclosed locations.

5.2.5 Nominal Equivalence of Traces

In practice, one does want to work with $\Sigma; \Gamma \vdash M : \tau$, which does not really use the full possibilities of trace semantics, imposing disclosure of all locations and the absence of disclosed higher-order values. We prefer to work directly on $\mathbf{Tr}\langle M, \gamma, \mathcal{I}, h, D\rangle$. However, as we have said before, this set is not nominally closed, since the initial move is missing. So to compare traces, we use *spans* over disclosed locations, as introduced in Section 3.3.2. They are generally written $\mathcal{D}$ in the following. We also use spans on player and opponent name pointers, written e_P and e_O which can be seen as binary environment relating variables of functional types of two terms. The nominal equivalence on heaps generated by the spans $\mathcal{D}, e_P, e_O$, written $h_1 \sim^{\mathcal{D}}_{e_P, e_O} h_2$, can then be characterized as:

$$\forall (l_1, l_2) \in \mathcal{D}_{\text{Int}}.\ h_1(l_1) = h_2(l_2) \wedge \forall (l_1, l_2) \in \mathcal{D}_{\tau \to \sigma}.(h_1(l_1), h_2(l_2)) \in e_P \uplus e_O$$
$$\wedge \forall (l_1, l_2) \in \mathcal{D}_{\text{ref}\, \iota}.\ (h_1(l_1), h_2(l_2)) \in \mathcal{D}_\iota$$

Then, actions are compared using spans on locations and names pointers. Given a span e_P on player name pointers, a span e_O on opponent name pointers and a span $\mathcal{D}$ on disclosed locations, we define the equivalence $(a_1, h_1) \sim^{\mathcal{D}}_{e_P,e_O} (a_2, h_2)$ on actions when $h_1 \sim^{\mathcal{D}}_{e_P,e_O} h_2$ and:

— either both a_i are player answers $\langle \bar{v_i} \rangle$ (resp. opponent answers $\langle v_i \rangle$) such that either both v_i are name pointers x_i with (x_1, x_2) in e_P (resp. in e_O), or both v_i are locations l_i with $(l_1, l_2) \in \mathcal{D}$, or both v_i are equal ground values,

— or both a_i are player questions $\bar{f_i}\,\langle v_i \rangle$ (resp. opponent questions $f_i\,\langle v_i \rangle$) such that (f_1, f_2) is in e_O (resp. in e_P) and either both v_i are name pointers x_i with (x_1, x_2) in e_P (resp. in e_O), or both v_i are locations l_i with $(l_1, l_2) \in \mathcal{D}$, or both v_i are equal ground values.

This equivalence is extended pointwisely on traces T_1, T_2, written $T_1 \simeq^{\mathcal{D}}_{e_P,e_O} T_2$. Then, taking two sets of traces $\mathrm{Tr}_1, \mathrm{Tr}_2$, we define $\mathrm{Tr}_1 \precsim^{\mathcal{D}}_{e_P,e_O} \mathrm{Tr}_2$ (resp. $\mathrm{Tr}_1 \succsim^{\mathcal{D}}_{e_P,e_O} \mathrm{Tr}_2$) when for all $T_1 \in \mathrm{Tr}_1$ (resp. $T_2 \in \mathrm{Tr}_2$), there exists three spans $e'_P, e'_O, \mathcal{D}'$ extending respectively e_P, e_O and $\mathcal{D}$ and $T_2 \in \mathrm{Tr}_2$ (resp. $T_1 \in \mathrm{Tr}_1$) such that $T_1 \simeq^{\mathcal{D}'}_{e'_P,e'_O} T_2$. We write $\mathrm{Tr}_1 \simeq^{\mathcal{D}}_{e_P,e_O} \mathrm{Tr}_2$ when $\mathrm{Tr}_1 \precsim^{\mathcal{D}}_{e_P,e_O} \mathrm{Tr}_2$ and $\mathrm{Tr}_1 \succsim^{\mathcal{D}}_{e_P,e_O} \mathrm{Tr}_2$.

Theorem 11: *Let M_1, M_2 two terms such that $\Sigma; \Gamma_g, \Gamma_f \vdash M_1, M_2 : \tau$. Then*

$$[\Sigma; \Gamma_g, \Gamma_f \vdash M_1 : \tau] = [\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$$

iff for all $\gamma_g : \Gamma_g \to \mathrm{Val}$, for all $h \in \mathbf{Cl}(\Sigma)$, $\mathbf{Tr}\langle \gamma_g(M_1), \varepsilon, \mathcal{I}, h, D \rangle \simeq^{\mathcal{D}}_{\varepsilon,\widetilde{\Gamma_f}} \mathbf{Tr}\langle \gamma_g(M_2), \varepsilon, \mathcal{I}, h, D \rangle$ where $\mathcal{I} = \nu_{\mathbb{P}}(\Gamma_f) \cup \mathrm{codom}(h_{\mathrm{fn}})$, $D = \mathrm{dom}(h)$ and $\mathcal{D} = \widetilde{D}$.

Proof Suppose that

$$[\Sigma; \Gamma_g, \Gamma_f \vdash M_1 : \tau] = [\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$$

and let $T_1 \in \mathbf{Tr}\langle \gamma_2(M_2), \varepsilon, \mathcal{I}, h, D \rangle$. Then from the nominal-closure of $[\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$, we get the existence of two (type-preserving) finite permutation π_L on Loc and $\pi_{\mathbb{P}}$ on $\mathbb{P}$ such that there exists $T_2 \in [\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$ with

$$\left(? \left\langle \Sigma, \overrightarrow{\gamma_g(x_i)}, \mathcal{I} \right\rangle, h\right) \cdot T_1 = \pi_L * \left(\pi_{\mathbb{P}} * \left(\left(? \left\langle \Sigma, \overrightarrow{\gamma(x_i)}, \mathcal{I} \right\rangle, h\right) T_2\right)\right)$$

Then, as explain in Section 3.3.2, there exists two spans $\mathcal{D}'$ and e'_O corresponding to π_L and $\pi_{\mathbb{P}}$. And from the equality on the initial (opponent questions moves), we get that $\mathcal{D}' \sqsupseteq \mathcal{D}$ an $e'_O \sqsupseteq e_O$. Thus

$$\mathbf{Tr}\langle \gamma_1(M_1), \varepsilon, \mathcal{I}, h, D \rangle \precsim^{\mathcal{D}}_{\varepsilon,\widetilde{\Gamma_f}} \mathbf{Tr}\langle \gamma_2(M_2), \varepsilon, \mathcal{I}, h, D \rangle$$

and the reverse inclusion is proven in the same way.

Reciprocally, suppose that $\mathbf{Tr}\langle\gamma_1(M_1),\varepsilon,\mathcal{I},h,D\rangle \simeq^{\mathcal{D}}_{\varepsilon,\widetilde{\Gamma_f}} \mathbf{Tr}\langle\gamma_2(M_2),\varepsilon,\mathcal{I},h,D\rangle$ and let $\left(?\left\langle\Sigma,\overrightarrow{\gamma_g(x_i)},\mathcal{I}\right\rangle,h\right)\cdot T_1 \in [\Sigma;\Gamma_g,\Gamma_f\vdash M_1:\tau]$, then we want to prove that there exists two (type-preserving) finite permutation π_L on Loc and $\pi_{\mathbb{P}}$ on $\mathbb{P}$ such that

$$\pi_L * \left(\pi_{\mathbb{P}} * \left(\left(?\left\langle\Sigma,\overrightarrow{\gamma(x_i)},\mathcal{I}\right\rangle,h\right)T_1\right)\right) \in [\Sigma;\Gamma_g,\Gamma_f\vdash M_2:\tau]\,. \qquad \blacksquare$$

Using again what have been presented in Section 3.3.2, these permutations are simply the extension of the two spans.

5.3 A Correspondence between Trace and Game Semantics

We now prove a formal link between the denotation of a term in trace semantics and in game semantics. The problem is that the definition of $[\Sigma;\Gamma\vdash M:\tau]$ is done operationally, while $[\![\Sigma;\Gamma\vdash M:\tau]\!]$—the game interpretation of terms defined in [MT11b]—is given denotationally, by induction on the typing judgment $\Sigma;\Gamma\vdash M:\tau$. To fill this gap, we show in this section that $[\Sigma;\Gamma\vdash M:\tau]$ can actually be decomposed by similar induction steps on the typing judgment. In Section 5.2.1 we have introduced the definition of a function Θ which transforms any justified trace on a trace-arena $\mathfrak{A}$ into a justified sequence of the corresponding game arena A. This correspondence allows us to state the following theorem.

Theorem 12 (Equivalence of the trace and the game semantics): *Let M a term of RefML such that $\Sigma;\Gamma\vdash M:\tau$, then $[\![\Sigma;\Gamma\vdash M:\tau]\!]$ is equal to $\Theta([\Sigma;\Gamma\vdash M:\tau])$.*

5.3.1 Denotation of variables

Consider $T\in[x:\tau\vdash x:\tau]$ with τ a functional type. We can write T as

$$(?\,\langle x\rangle\,,\varepsilon)\cdot(\langle\bar{y}\rangle\,,\varepsilon)\cdot T'$$

such that $\langle x,\cdot,x,\varepsilon,\varepsilon\rangle \xrightarrow{\langle\bar{y}\rangle,\varepsilon} \langle\Diamond,y\hookrightarrow x,x,\varepsilon,\varepsilon\rangle$. Then, when opponent questioned y with u in T', player directly interrogates x with either u if it is atomic, or with a fresh name pointer that points to u.

Such behavior is called copycat. More precisely, following [MT11b], a pair of actions $(a_1,h_1),(a_2,h_2)$ are a *copycat pair* in a play-trace T if

— they are consecutive in T,
— $\theta((a_1,h_1))=\theta((a_2,h_2))$,

— if (a_1, h_1) is justified by (a'_1, h'_1) then (a_2, h_2) is justified by (a'_2, h'_2) *s.t.* $(a'_2, h'_2), (a'_1, h'_1)$ are consecutive in T, and moreover if (a_1, h_1) is l-justified with $l \in \mathrm{dom}(h'_1)$, then (a_2, h_2) is l-justified so $l \in \mathrm{dom}(h'_2)$.

Notice that a copycat pair $(\langle u_1 \rangle, h_1) \cdot (\langle \bar{u_2} \rangle, h_2))$ will not satisfy $u_1 = u_2$ nor $h_1 = h_2$ when the u_i are name pointers and when the h_i store name pointers, due to the freshness condition of name pointers in the rules of the interactive reduction. This justifies the use of θ in the definition.

Then, taking a trace-arena $\mathfrak{A}$ we define the identity trace-strategy on $\mathfrak{A} \to \mathfrak{A}$, written $\mathfrak{id}_{\mathfrak{A}}$ as the set of play-traces $T \in \mathfrak{P}_{\mathfrak{A} \to \mathfrak{A}}$ such that for all subtraces $(o, h) \cdot (p, h')$ of length two of T, $((o, h), (p, h'))$ is a copycat pair of T. We can easily see that $\Theta(\mathfrak{id}_{\mathfrak{A}})$ is equal to id_A, the usual identity game-strategy over the corresponding game-arena $A \to A$.

We want to prove that $[x : \tau \vdash x : \tau]$ is equal to $\mathfrak{id}_{[\tau]}$. To do so, we first need the following lemma, which generalize this equality when τ is functional.

Lemma 10: *Let $\mathcal{I}$ a set of opponent name pointers, γ a substitution on player name pointers such that* $\mathrm{codom}(\gamma) \subseteq \mathcal{I}$*, h a functional-free heap such that* $\mathrm{codom}(h_{\mathrm{fn}}) \subseteq \mathcal{I}$ *and* $D = \mathrm{dom}(h)$*. Then a trace $T \in \mathrm{Tr}\langle \underbrace{\bullet \cdot \ldots \cdot \bullet}_{n \text{ times}}, \gamma, \mathcal{I}, h, D \rangle$ is formed by copycat pairs.*

Proof By induction on the length of T. If T begins with an opponent question, then $T = (x \langle u \rangle, h_1) \cdot (\bar{y} \langle \widetilde{u} \rangle, h_2) \cdot T'$ with

$$\begin{array}{rcl} \langle \bullet \cdot \ldots \cdot \bullet, \gamma, \mathcal{I}, h, D \rangle & \xrightarrow{x\langle u \rangle, h_1} & \langle yu \cdot \bullet \cdot \ldots \cdot \bullet, \gamma, \mathcal{I}_1, h_1, D_1 \rangle \\ & \xrightarrow{\bar{y}\langle \widetilde{u} \rangle, h_2} & \langle \bullet \cdot \bullet \cdot \ldots \cdot \bullet, \gamma_1, \mathcal{I}_1, h_2, D_1 \rangle \end{array}$$

where $y = \gamma(x)$ and if u is atomic then $\widetilde{u} = u$, otherwise $u \in \mathcal{I}_1$, $\widetilde{u}$ is a player name pointer and $\gamma_1(\widetilde{u}) = u$. Moreover, due to conditions of the rules **O-Quest** and **P-Quest**, $\mathrm{codom}(h_{1,fn}) \subseteq \mathcal{I}_1$ and $\mathrm{codom}(h_{2,fn}) \subseteq \mathrm{dom}(\gamma_1)$ with $\gamma_1(\mathrm{codom}(h_{2,fn})) \subseteq \mathcal{I}_1$. Thus, $\mathrm{codom}(\gamma_1) \subseteq \mathcal{I}_1$, and we can apply the induction hypothesis on T' to conclude.

Otherwise, T begins with an opponent answer (so $n > 0$), so $T = (\langle u \rangle, h_1) \cdot (\left\langle \bar{\widetilde{u}} \right\rangle, h_2) \cdot T'$ with

$$\begin{array}{rcl} \langle \bullet \cdot \bullet \cdot \ldots \cdot \bullet, \gamma, \mathcal{I}, h, D \rangle & \xrightarrow{\langle u \rangle, h_1} & \langle u \cdot \bullet \cdot \ldots \cdot \bullet, \gamma, \mathcal{I}_1, h_1, D_1 \rangle \\ & \xrightarrow{\left\langle \bar{\widetilde{u}} \right\rangle, h_2} & \langle \bullet \cdot \ldots \cdot \bullet, \gamma_1, \mathcal{I}_1, h_2, D_1 \rangle \end{array}$$

s.t. if u is atomic then $\widetilde{u} = u$, otherwise $u \in \mathcal{I}_1$, $\widetilde{u}$ is a player name pointer and $\gamma_1(\widetilde{u}) = u$. Then, using the same argument as before, $\mathrm{codom}(\gamma_1) \subseteq \mathcal{I}_1$, and we can apply the induction hypothesis on T' to conclude.

We now prove the wanted theorem.

Theorem 13: *The trace-strategy $[x : \tau \vdash x : \tau]$ is equal to $\mathfrak{id}_{[\tau]}$.*

Proof If τ is equal to $\mathrm{Unit}, \mathrm{Bool}$ or Int, then the result is straightforward. If τ is equal to $\mathrm{ref}\,\sigma$, then taking $T \in [x : \mathrm{ref}\,\sigma \vdash x : \mathrm{ref}\,\sigma]$ there exists a location $l \in \mathrm{Loc}_\sigma$

such that $T \in (?\langle l\rangle, h) \cdot \mathbf{Tr}\langle l, \varepsilon, \mathcal{I}, h, D\rangle$ with $D = \mathrm{dom}(h)$ and $\mathcal{I} = \mathrm{codom}(h_{\mathrm{fn}})$. Then, $\langle l, \varepsilon, \mathcal{I}, h, D\rangle \xrightarrow{\langle \vec{l}\rangle h} \langle l, \varepsilon, \mathcal{I}, h, D\rangle$ and the previous lemma applied.

Finally, if τ is functional, the equality is also deduced from the previous theorem.

In the same way, taking n trace-arenas $\mathfrak{A}_1, \ldots, \mathfrak{A}_n$ we define the ith-projection trace-strategy on $\mathfrak{A}_1 \otimes \ldots \mathfrak{A}_n \to \mathfrak{A}_i$, written $\mathfrak{p}_i$ as the set of play-traces $T \in \mathfrak{P}_{\mathfrak{A}_1 \otimes \ldots \mathfrak{A}_n \to \mathfrak{A}_i}$ such that either T is empty, or the two first actions of T are $(?\langle \overrightarrow{u_j}\rangle, h) \cdot (\langle \overrightarrow{\widetilde{u_i}}\rangle, h')$ with $\widetilde{u_i} = u_i$ if u_i is a ground value, otherwise it is a pointer, and for all other subtraces $(o, h) \cdot (p, h')$ of length two of T, $((o, h), (p, h'))$ is a copycat pair of T. Again, we can easily see that $\Theta(\mathfrak{p}_i)$ is equal to π_i, the usual projection game-strategy over the corresponding game-arena A.

Theorem 14: *Let $i \in \{1, \ldots, n\}$ then $[x_1 : \tau_1, \ldots, x_n : \tau_n \vdash x_i : \tau_i] = \mathfrak{p}_i$*

5.3.2 Denotation of λ-abstractions

Next, we proceed by showing that $[\Gamma \vdash \lambda x.M : \sigma \to \tau]$ can be reconstructed from $[\Gamma, x : \sigma \vdash M : \tau]$ exactly in the same way as in game semantics.

Let $T \in \mathbf{Tr}\langle \lambda x.M, \gamma, \mathcal{I}, h, D\rangle$ then the first action of T is $(\langle \bar{y}\rangle, h'_{|D})$ such that

$$\langle \lambda x.M, \gamma, \mathcal{I}, h, D\rangle \xrightarrow{\langle \bar{y}\rangle, h'} \langle \Diamond, \gamma', \mathcal{I}, h', D\rangle$$

where $\mathrm{dom}(h') = D$, $\theta(h_{|D}) = \theta(h'_{|D})$ and for all $l \in D \cap \mathrm{Loc}_\phi$, $h(l), h'(l)$ are two name pointers which form a copycat pair in T when they are questioned. This justifies the following two definitions, imported from [MT11b].

We first introduce the notion of *copycat triple*, which allows us to control the callbacks performed indirectly with functions stored in the heap. Indeed, just as in nominal game semantics, only name pointers are stored in disclosed functional part of heaps. And even if what is stored in a location is not modified, the interaction rule refreshes this name pointer. We use the following definition to control this refreshing.

Definition 28. *Let T a legal trace and $T' \sqsubseteq T$, with T' ending with $(a_1, h_1) \cdot (a_2, h_2)$ and $l \in \mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) \cap \mathrm{Loc}_\phi$ a location of functional type, we say that (T, T', l) is a* copycat triple *if for all ϕ-actions (a'_1, h'_1) of T which have (a_1, h_1) or (a_2, h_2) as an l-ancestor, there exists an action (a'_2, h'_2) such that:*

- *if a'_1 has the same player as a_1, then $(a'_1, h'_1) \cdot (a'_2, h'_2)$ is a copycat pair of T,*
- *if a'_1 has the same player as a_2, then $(a'_2, h'_2) \cdot (a'_1, h'_1)$ is a copycat pair of T.*

Using it, we define the notion of total trace strategies.

Definition 29. *A trace-strategy $\mathfrak{s}$ on the arena $\mathfrak{A}$ is said to be* total *if*
- *for every initial opponent question $(?\langle u_1\rangle, h_1) \in \mathfrak{P}_{\mathfrak{A}}$, there exists a player answer $(\langle \bar{u_2}\rangle, h_2)$ such that $\theta(h_1) = \theta(h_2)$ and $(?\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2) \in \mathfrak{s}$,*
- *for $(?\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T \in \mathfrak{s}$, and $l \in \mathrm{dom}(h)$, $\Big((?\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T, (?\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2), l\Big)$ is a copycat triple.*

This means, among others, that traces of $\mathfrak{s}$ begin with an opponent question $(?\langle u_1\rangle, h_1)$ followed by a player answer $(\langle \bar{u_2}\rangle, h_2)$, without modifying the heap. Then, we easily see that the trace strategy $[\Sigma; \Gamma \vdash \lambda x.M : \sigma \to \tau]$ is a total trace strategy, due to the fact that the interactive execution of $\lambda x.M$ will update the heap with fresh player name pointers which point to the initial opponent name pointers, just generating copycat triples.

Then, for such total strategies, we define the notion of *threads*, which are subtraces which are generated by the opponent questions of the player answer $(\langle \bar{u_1}\rangle, h_1)$. So, taking a total trace $T = (\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T'\cdot(a, h_3)$, we define inductively $\mathbf{Thread}(T)$ as:
- $(\langle u_1\rangle, h_1')\cdot(\langle \bar{u_2}\rangle, h_2')\cdot(a, h_3)$ if (a, h_3) is justified by $\langle \bar{u_2}\rangle$ where h_i' is a heap whose domain equal to $h_3^*(\nu_L(u_1, u_2))$, $h_i'(l) = h_i(l)$ for all $l \in \mathrm{dom}(h_i') \cap \mathrm{Loc}_\phi$, $h_i'(l) = h_3(l)$ elsewhere (*i.e.* on location storing ground values),
- $(\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)$ if (a, h_3) is l-justified by $\langle \bar{u_2}\rangle$,
- $\mathbf{Thread}((\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T')\cdot(a, h_3)$ if (a, h_3) is justified by $(\langle u_1\rangle, h_1)$,
- $\mathbf{Thread}((\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T'')\cdot(a, h_3)$ if (a, h_3) is justified by an action in T', where T'' is the longest prefix of T' such that the action which justified (a, h_3) is in $\mathbf{Thread}((\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T'')$.

This notion is extended straightforwardly to sets of traces, so to trace-strategies. In general, $\mathbf{Thread}(T)$ does not satisfy the frugality condition, so we define a function **Frug** which removes part of the heap which has not been disclosed:
- $\mathbf{Frug}(\varepsilon) = \varepsilon$
- $$\mathbf{Frug}(T\cdot(a,h)) = \begin{cases} \mathbf{Frug}(T) & \text{if } (a,h) \text{ is hereditarily } l\text{-justified by} \\ & (a',h') \text{ and } l \notin \mathbf{Av}(T_{\leq(a',h')}) \\ \mathbf{Frug}(T)\cdot(a, h_{|\mathbf{Av}(T\cdot(a,h))}) & \text{otherwise.} \end{cases}$$

Trace-strategies which are formed by such frugal threads can be characterized, they are called *strongly single-threaded* trace-strategies [5]

5. The case were an action could be l-justified by $(\langle \bar{u_2}\rangle, h_2)$ was left unclear in this definition and the following in [MT11b]. Nikos Tzevelekos has proposed (in a private communication) the solution followed here to forbid them in the definition of strongly single-threaded strategies, and adding them in the definition of $(.)^\dagger$.

Definition 30. *A trace-strategy $\mathfrak{s}$ is said* strongly single-threaded *when it is total and for all $(?\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T \in \mathfrak{s}$,*
— *there is at most one move which is justified by $\langle \bar{u_2}\rangle$ in T,*
— *there is no move which is l-justified by $(\langle \bar{u_2}\rangle, h_2)$,*
and for all $(?\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T, (?\langle u_1\rangle, h_1')\cdot(\left\langle \bar{u_2'}\right\rangle, h_2')\cdot T' \in \mathfrak{s}$, $\theta(u_2)=\theta(u_2')$.

Then, we define a function $\mathfrak{L}^{\mathfrak{B}}_{\mathfrak{A},\mathfrak{C}}$ which maps trace-strategies of $(\mathfrak{A}\otimes\mathfrak{B})\to\mathfrak{C}$ into trace-strategy of $\mathfrak{A}\to(\mathfrak{B}\Rightarrow\mathfrak{C})$ as the transformation of the trace $(?\langle u,v\rangle, h\cdot h_{\mathfrak{B}})\cdot T$ into the set of traces which can be written as $(?\langle u\rangle, h)\cdot(\langle \bar{y}\rangle, h')\cdot(y\langle v\rangle, h\cdot h_{\mathfrak{B}})$ where
— $\mathrm{dom}(h)=\mathbf{Frug}(?\langle u\rangle, h)$ and $\mathrm{dom}(h_{\mathfrak{B}})=\mathbf{Frug}(?\langle v\rangle, h_{\mathfrak{B}})$,
— $\mathrm{dom}(h')=\mathrm{dom}(h)$ and $h'_{\mathrm{g}}=h_{\mathrm{g}}$,
— y and $\mathrm{codom}(h'_{\mathrm{fn}})$ are fresh player name pointers.
So we get the following theorem:

Theorem 15: *The set of traces* $\mathbf{Frug}(\mathbf{Thread}([\Gamma\vdash\lambda x.M:\sigma\to\tau]))$ *is equal to*

$$\mathfrak{L}^{[\sigma]}_{[\Gamma],[\tau]}([\Gamma, x:\sigma\vdash M:\tau])$$

Proof The inclusion $\mathbf{Frug}(\mathbf{Thread}([\Gamma\vdash\lambda x.M:\sigma\to\tau]))$ into $\mathfrak{L}^{[\sigma]}_{[\Gamma],[\tau]}([\Gamma, x:\sigma\vdash M:\tau])$ comes from the fact that elements of $\mathfrak{L}^{[\sigma]}_{[\Gamma],[\tau]}([\Gamma, x:\sigma\vdash M:\tau])$ are strongly-single threaded plays which are, by construction of $\mathfrak{L}^{[\sigma]}_{[\Gamma],[\tau]}$, in $[\Gamma\vdash\lambda x.M:\sigma\to\tau]$.

We prove the reverse inclusion. Let us decompose Γ into Γ_g,Γ_f, and let $\gamma:\Gamma_g\to\mathrm{Val}$. We write $\widetilde{M}$ for $\gamma(M)$. Let h a heap such that $h^*(\nu_L(\mathrm{codom}(\gamma_g)))=\mathrm{dom}(h)$. We define D as $\mathrm{dom}(h)$. For sake of simplicity, we work with $\mathcal{I}$ equal to Γ_f rather than nominally equivalent. Suppose that x is of functional type. We prove that

$$\mathbf{Frug}\left(\mathbf{Thread}((?\left\langle\overrightarrow{\gamma(x_i)},\mathcal{I}\right\rangle, h)\cdot\mathbf{Tr}\langle\lambda x.\widetilde{M},\varepsilon,\mathcal{I},y,h,D\rangle)\right)$$

is included in

$$\{\mathfrak{L}^{[\sigma]}_{[\Gamma],[\tau]}\left((?\left\langle\overrightarrow{\gamma(x_i)},\mathcal{I},y\right\rangle, h)\cdot\mathbf{Tr}\langle\widetilde{M}\,\{y/x\},\varepsilon,\mathcal{I},y,h,D\rangle\right)\mid y\sim_{\mathbb{P}} x\}$$

Let $T\in\mathbf{Tr}\langle\lambda x.\widetilde{M},\varepsilon,\mathcal{I},y,h,D\rangle$. Then, $T=(\langle\bar{z}\rangle, h')\cdot T'$ with
— $\mathrm{dom}(h')=\mathrm{dom}(h)$,
— $h'_{\mathrm{g}}=h_{\mathrm{g}}$,
— and z and $\mathrm{codom}(h'_{\mathrm{fn}})$ are fresh player name pointers.
We reason by induction on the size of T'. If $|T|=0$, then by definition of $\mathfrak{L}^{[\sigma]}_{[\Gamma],[\tau]}$ we get the wanted property. Otherwise, consider the last action a, h'' of T. Then the crucial property is that this action cannot be l-justified by a location in $\mathrm{dom}(h)$, by definition of the current thread. ∎

From this, we can define the notion of *thread-independent* trace plays. They correspond to plays where there is no interaction between their threads.

Definition 31. *A trace-play $T = (?\,\langle u_1\rangle, h_1)\cdot(\langle \bar{u_2}\rangle, h_2)\cdot T'$ is thread-independent if for all $T''\cdot(p,h) \sqsubseteq^{\text{even}} T$, we have:*
- $\mathbf{thr}(T''\cdot(p,h)) = \mathbf{thr}(T'')\cdot(p,h)$,
- $\nu_L(\gamma(\mathbf{thr}(T''\cdot(p,h)))\cap\nu_L(T'')\subseteq\nu_L(\gamma(\mathbf{thr}(T''))))$,
- *if T'' ends in (o,h') and $l\in \mathrm{dom}(h')\backslash\nu_L(\mathbf{thr}(T''))$ then $\theta(h(l)) = \theta(h'(l))$, and if $l\in \mathrm{Loc}_\phi$ then $(T, T''\cdot(p,h), l)$ is a copycat triple.*

The set of thread-independent trace-play over $\mathfrak{A}$ is written $\mathfrak{P}^{\mathrm{ti}}_{\mathfrak{A}}$.

Thus, following [MT11b], we can define a "shuffle" operation $(\cdot)^\dagger$ which transforms a strongly single-threaded strategy $\mathfrak{s}$ on $\mathfrak{A}$ into a thread-independent strategy $\mathfrak{s}^\dagger$, defined as

$$\{T\in\mathfrak{s}\mid |T|\leq 2\}\cup\{T\in\mathfrak{P}^{\mathrm{ti}}_{\mathfrak{A}}\mid \forall T'\sqsubseteq^{\text{even}} T.\gamma(\mathbf{thr}(T'))\in\mathfrak{s}$$
$$\text{or } T' = (?\,\langle u_1\rangle, h_1)\cdot(\langle\bar{u_2}\rangle, h_2)\cdot(a,h)\cdot T'' \text{ with } (a,h)\ l\text{-justified by some } l\in\mathrm{dom}(h_2)$$
$$\text{and } (T', ?\,\langle u_1\rangle, h_1)\cdot(\langle\bar{u_2}\rangle, h_2), l) \text{ is a copycat triple.}\}$$

Notice that we cannot just enforce that $\gamma(\mathbf{thr}(T'))\in\mathfrak{s}$ otherwise actions which are l-justified with $l\in\mathrm{dom}(h_2)$ would not be in $\mathfrak{s}^\dagger$ since the definition of the current thread removes them. This shuffle operation satisfies the following equality:

Lemma 11: *For any thread-independent trace strategy $\mathfrak{s}$, $(\gamma(\mathbf{thr}(\mathfrak{s})))^\dagger = \mathfrak{s}$.*

Then, we can easily check that $[\Sigma;\Gamma\vdash\lambda x.M:\sigma\to\tau]$ is thread-independent. Using it, we can decompose the denotation of λ-abstraction exactly as it is done in game semantics:

Corollary 1: $[\Sigma;\Gamma\vdash\lambda x.M:\sigma\to\tau] = (\Lambda([\Sigma;\Gamma, x:\sigma\vdash M:\tau]))^\dagger$.

5.3.3 Composition of Trace Strategies

Let $\mathfrak{A},\mathfrak{B},\mathfrak{C}$ three trace value-arenas, we define the term-arena $\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}\stackrel{def}{=}(\mathfrak{M}_{\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}}, \mathfrak{I}_{\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}}, \vdash_{\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}})$ where:
- $\mathfrak{I}_{\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}}\stackrel{def}{=}(\ell\cdot\ell)\cdot\mathfrak{I}_{\mathfrak{A}}$,
- $\mathfrak{M}_{\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}}\stackrel{def}{=}\ell\cdot(\mathfrak{M}_{\mathfrak{A}\to\mathfrak{B}}\backslash\mathfrak{I}_{\mathfrak{B}})\uplus\{(\bar{?}\,\langle u\rangle, \ell\cdot r\cdot\xi)\mid(\langle\bar{u}\rangle,\xi)\in\mathfrak{I}_{\mathfrak{B}}\}\uplus(r\cdot r)\cdot\mathfrak{M}_{\mathfrak{C}}{}^{\perp}$,
- $\vdash_{\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}}\stackrel{def}{=}\{(\ell\cdot\ell\cdot\mathrm{i}_A, \ell\cdot r\cdot\mathrm{i}_B)\}\uplus\{(\ell\cdot r\cdot\mathrm{i}_B, r\cdot r\cdot\mathrm{i}_C)\}\uplus(\ell\cdot\ell\cdot\vdash_A)\uplus(\ell\cdot r\cdot\vdash_B)\uplus(r\cdot r\cdot\vdash_C)$

Taking a trace T on $\mathfrak{A}\to\mathfrak{B}\to\mathfrak{C}$, we define the following restricted traces:
- $T_{|(\mathfrak{A},\mathfrak{B})}$ is defined as the traces formed by the actions of T in $\mathfrak{M}_{\mathfrak{A}\to\mathfrak{B}}\backslash\mathfrak{I}_{\mathfrak{B}}$ plus the player answers $(\langle\bar{u}\rangle, h)\in\mathfrak{I}_{\mathfrak{B}}$ for every player question $(\bar{?}\,\langle u\rangle, h)$ of T,

— $T_{|(\mathfrak{B},\mathfrak{C})}$ is defined as the traces formed by the actions $(a^\perp, h)$ for every actions (a, h) of T such that $(a, h) \in \mathfrak{M}_{\mathfrak{B}} \backslash \mathfrak{I}_{\mathfrak{B}}$, the actions $(. \langle u \rangle, h)$ for every player questions $(\bar{?} \langle u \rangle, h)$ of T with $(\langle \bar{u} \rangle, h) \in \mathfrak{I}_{\mathfrak{B}}$, and the actions $(a^\perp, h)$ for every actions (a, h) of T such that $(a^\perp, h) \in \mathfrak{M}_{\mathfrak{C}}$,
— $T_{|(\mathfrak{A},\mathfrak{C})}$ is defined as the traces formed by the actions (a, h) of T such that $(a^\perp, h) \in \mathfrak{M}_{\mathfrak{A}} \backslash \mathfrak{I}_{\mathfrak{A}}$, the opponent questions $(x \langle u \rangle, h)$ of T such that $(\langle \bar{u} \rangle, h) \in \mathfrak{I}_{\mathfrak{A}}$, and the actions $(a^\perp, h)$ for every actions (a, h) of T such that $(a^\perp, h) \in \mathfrak{M}_{\mathfrak{C}}$.

In the definition of $T_{|(\mathfrak{A},\mathfrak{C})}$, the case where an action of $\mathfrak{C}$ is justified by an action coming from $\mathfrak{B}$ which itself is justified by an action coming from $\mathfrak{A}$ is not problematic, contrary to game semantics where we have to add a new pointer. Indeed, since the action from $\mathfrak{C}$ is an answer and the action from $\mathfrak{A}$ is a question, the justification is automatic.

Then, for any $X \in \{(\mathfrak{A}, \mathfrak{B}), (\mathfrak{B}, \mathfrak{C}), (\mathfrak{A}, \mathfrak{C})\}$, we define $T_{|_{\mathbf{F}} X}$ as $\mathbf{Frug}(T_{|X})$. Using this, we can define the set of *interaction traces*. To do so, we first characterize the introduction actions of locations of $\nu_L(T)$.

Definition 32. *An action (a, h) is said to* introduce *a location l if, writing T as $T_1 \cdot (a, h) \cdot T_2$, we have $l \in \nu_L(a, h) \backslash \nu_L(T_1)$. A location is P-introduced (resp. O-introduced) in T if the action which introduces it is a player action (resp. opponent action).The set of P-introduced (resp. O-introduced) locations of $\nu_L(T)$ is written $\mathbf{P}(T)$ (resp. $\mathbf{O}(T)$).*

It is straightforward that $\nu_L(T) = \mathbf{P}(T) \uplus \mathbf{O}(T)$. We can now define the interaction traces, following the definition from game semantics.

Definition 33. *A justified trace T on $\mathfrak{A} \to \mathfrak{B} \to \mathfrak{C}$ is an* interaction trace *if*
— *it is frugal,*
— *$T_{|(\mathfrak{A},\mathfrak{B})}, T_{|(\mathfrak{B},\mathfrak{C})}$ and $T_{|(\mathfrak{A},\mathfrak{C})}$ are legal,*
— *$\mathbf{P}(T_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{B})}) \cap \mathbf{P}(T_{|_{\mathbf{F}}(\mathfrak{B},\mathfrak{C})}) = \varnothing$,*
— *$\mathbf{O}(T_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{C})}) \cap (\mathbf{P}(T_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{B})}) \cup \mathbf{P}(T_{|_{\mathbf{F}}(\mathfrak{B},\mathfrak{C})})) = \varnothing$.*

The set of interaction traces over the arenas $\mathfrak{A}, \mathfrak{B}, \mathfrak{C}$ is written $\mathbf{Interact}(\mathfrak{A}, \mathfrak{B}, \mathfrak{C})$.

So we define the parallel composition of two trace strategies $\mathfrak{s}, \mathfrak{t}$, written $\mathfrak{s}||\mathfrak{t}$ as the set of interaction traces $T \in \mathfrak{A} \to \mathfrak{B} \to \mathfrak{C}$ *s.t.* $T_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{B})} \in \mathfrak{s}$ and $T_{|_{\mathbf{F}}(\mathfrak{B},\mathfrak{C})} \in \mathfrak{t}$.

Lemma 12: *Let $T \in \mathfrak{A} \to \mathfrak{B} \to \mathfrak{C}$, then $T_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{B})}$, $T_{|_{\mathbf{F}}(\mathfrak{B},\mathfrak{C})}$ and $T_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{C})}$ are play traces.*

Then, taking $\mathfrak{s}, \mathfrak{t}$ two trace-strategies defined respectively on $\mathfrak{A} \to \mathfrak{B}$ and $\mathfrak{B} \to \mathfrak{C}$, we define their composition, written $\mathfrak{s}; \mathfrak{t}$, as the trace-strategy on $\mathfrak{A} \to \mathfrak{C}$ formed by traces $T \in \mathfrak{P}_{\mathfrak{A} \to \mathfrak{C}}$ such that there exists $T' \in \mathfrak{s}||\mathfrak{t}$ with $T = T'_{|_{\mathbf{F}}(\mathfrak{A},\mathfrak{C})}$.

Lemma 13: $[\Gamma \vdash v : \sigma] ; [x : \sigma \vdash M : \tau] = [\Gamma \vdash M\{v/x\} : \tau]$.

Theorem 16: $[\Gamma \vdash N : \sigma] ; [x : \sigma \vdash M : \tau] = [\Gamma \vdash \texttt{let}\ x = N\ \texttt{in}\ M : \tau]$.

5.3.4 Pairing

We now define the pairing of two strategies following the definition of game semantics. Let $\mathfrak{A}$ a trace-arena and $\mathfrak{s}$ a trace strategy defined on $\mathfrak{B} \to \mathfrak{C}$, we define the left identity pairing strategy $\mathfrak{A} \otimes \mathfrak{s}$ on the trace term-arena $\mathfrak{A} \otimes \mathfrak{B} \to \mathfrak{A} \otimes \mathfrak{C}$ as the set of legal traces T in $\mathfrak{L}_{\mathfrak{A}\otimes\mathfrak{B}\to\mathfrak{A}\otimes\mathfrak{C}}$ such that

— $T_{|(\mathfrak{B},\mathfrak{C})} \in \mathfrak{s}$,

— $T_{|(\mathfrak{A},\mathfrak{A})} \in \mathfrak{id}_{\mathfrak{A}}$.

In a dual way, we can also define the right identity pairing strategy $\mathfrak{s} \otimes \mathfrak{A}$ on the trace term-arena $\mathfrak{B} \otimes \mathfrak{A} \to \mathfrak{C} \otimes \mathfrak{A}$. Using the two, we can define the pairing of two trace strategies $\mathfrak{s}, \mathfrak{t}$ defined respectively on $\mathfrak{A} \to \mathfrak{B}$ and $\mathfrak{C} \to \mathfrak{D}$, written $\mathfrak{s} \otimes \mathfrak{t}$. It is defined as $\mathfrak{s} \otimes \mathfrak{C}; \mathfrak{B} \otimes \mathfrak{t}$. Then, we can characterize the pairing of two trace strategies when they represent the denotation of terms.

Theorem 17: *Let M_1, M_2 two terms such that $\underbrace{\Gamma_{1,g}, \Gamma_{1,f}}_{\Gamma_1} \vdash M_1 : \sigma$ and $\underbrace{\Gamma_{2,g}, \Gamma_{2,f}}_{\Gamma_2} \vdash M_2 : \tau$ with Γ_1, Γ_2 disjoints. Then $([\Gamma_1 \vdash M_1 : \sigma] \otimes [\Gamma_2 \vdash M_2 : \tau])$ is equal to the nominal closure of the set of even prefix of traces belonging to*

$$\left(? \left\langle \overrightarrow{\gamma_{1,g}(x_i)}, \mathcal{I}_1, \overrightarrow{\gamma_{2,g}(x_i)}, \mathcal{I}_2 \right\rangle, h_1 \cdot h_2\right) \cdot T_1 \cdot T_2 \cdot \left(\langle \overline{u_1, u_2} \rangle, h'_{|D'}\right) \cdot \mathbf{Tr}\langle \Diamond, \gamma', \mathcal{I}', h', D' \rangle$$

with $\gamma_{i,g} : \Gamma_{i,g} \to \mathrm{Val}$ and $h_i \in \mathbf{Cl}(\nu_L(\mathrm{codom}(\gamma_{i,g})))$ with h_1, h_2 disjoint (so necessarily $\nu_L(\mathrm{codom}(\gamma_{1,g})), \nu_L(\mathrm{codom}(\gamma_{2,g}))$ disjoint too) and functional-free, D_i defined as $\mathrm{dom}(h_i)$ and $\mathcal{I}_i$ as $\nu_{\mathbb{P}}(\Gamma_{f,i}, \mathrm{codom}(\gamma_{g,i}))$, x_i ranging over $\Gamma_{i,g}$ and

— $\langle \gamma_{1,g}(M_1), \varepsilon, \mathcal{I}_1, h_1, D_1 \rangle \xrightarrow{T_1} \langle u'_1, \gamma_1, \mathcal{I}'_1, h'_1, D'_1 \rangle$ *with u'_1 a value, and h'_1 disjoint of h_2, $\mathcal{I}'_1$ disjoint of $\mathcal{I}_2$*

— $\langle \gamma_{2,g}(M_2), \gamma_1, \mathcal{I}'_1 \cdot \mathcal{I}_2, h'_1 \cdot h_2, D'_1 \cup D_2 \rangle \xrightarrow{T_2} \langle u'_2, \gamma_2, \mathcal{I}', h'_2, D'_2 \rangle$ *with u'_2 a value,*

— $D' = \mathrm{discl}(\{u_1, u_2\}, h', D_2)$,

— $h' = h'_2[\overrightarrow{l_i \hookrightarrow x_i}]$ *with l_i ranging over $h'_{2,\mathrm{fn}} \cap D'$ and the x_i not in γ_2 and distinct from u_1, u_2,*

— *γ' is the extension of γ_2 with $(u_i \mapsto u'_i)$ if u'_i is of functional type (otherwise $u'_i = u_i$) and with all the $(x_i \mapsto h_2(l_i))$ for l_i ranging over $h_{2,fn|D'}$.*

Proof Let $T_3 \in \mathbf{Tr}\langle \Diamond, \gamma', \mathcal{I}', h', D' \rangle$, we first prove that

$$\left(? \left\langle \overrightarrow{\gamma_{1,g}(x_i)}, \mathcal{I}_1, \overrightarrow{\gamma_{2,g}(x_i)}, \mathcal{I}_2 \right\rangle, h_1 \cdot h_2\right) \cdot T_1 \cdot T_2 \cdot \left(\langle \overline{u_1, u_2} \rangle, h'_{|D'}\right) \cdot T_3$$

belongs to $([\Gamma_1 \vdash M_1 : \sigma] \otimes [\Gamma_2 \vdash M_2 : \tau])$, using the fact that this last set is nominally closed to avoid to reason up-to nominal equivalence on the former trace. For each $y_j \in \mathcal{I}_2$, we associate a fresh player name pointer $\tilde{y_j}$, which form a new set $\widetilde{\mathcal{I}_2}$ nominally equivalent to $\mathcal{I}_2$. Similarly, for each $z_k \in \nu_{\mathbb{P}}(\mathrm{codom}(h_2))$, we associate a fresh name pointer $\tilde{z_k}$, and we define h'_2 as $h_2\overrightarrow{\{z_k/\tilde{z_k}\}}$. We consider the trace $\widetilde{T_2} \in \mathbf{Tr}\langle \gamma_{2,g}(M_2\overrightarrow{\{y_j/\tilde{y_j}\}}), \widetilde{\gamma_1}, \mathcal{I}'_1 \cdot \widetilde{\mathcal{I}_2} \cdot \overrightarrow{\tilde{z_k}}, h'_1 \cdot \widetilde{h_2}, D'_1 \cup D_2\rangle$ where $\widetilde{\gamma_1} = \gamma_1 \cdot \overrightarrow{[\tilde{y_j} \mapsto y_j]} \cdot \overrightarrow{[\tilde{z_k} \mapsto z_k]}$. In the same way, we define $\widetilde{u_1}$ as u_1 if it is a ground value, and as a fresh name pointer otherwise, in which case we define $\widetilde{\gamma'}$ as $\gamma' \cdot [\widetilde{u_1} \hookrightarrow u_1]$. Then, let $\widetilde{T_3} \in \mathbf{Tr}\langle \Diamond, \widetilde{\gamma'}, \mathcal{I}', h', D'\rangle$

$$\underbrace{\left(?\left\langle \overrightarrow{\gamma_{1,g}(x_i)}, \mathcal{I}_1, \overrightarrow{\gamma_{2,g}(x_i)}, \mathcal{I}_2 \right\rangle, h_1 \cdot h_2\right) \cdot T_1 \cdot \left(\bar{?}\left\langle u'_1, \overrightarrow{\gamma_{2,g}(x_i)}, \widetilde{\mathcal{I}_2} \right\rangle, h'_{1|D'_1} \cdot \widetilde{h_2}\right) \cdot \widetilde{T_2}^{\perp} \cdot \left(\langle \overrightarrow{u_1, u_2} \rangle, h'_{|D'}\right) \cdot \widetilde{T_3}}_{T}$$

is in $([\Gamma_1 \vdash M_1 : \sigma] \otimes [\Gamma_2]) || ([\sigma] \otimes [\Gamma_2 \vdash M_2 : \tau])$. To do so, we prove that T it is indeed an interaction trace over $([\Gamma_1] \otimes [\Gamma_2]) \to ([\sigma] \otimes [\Gamma_2]) \to ([\sigma] \otimes [\tau])$.

First, $T_{|([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\Gamma_2])}$ is legal, since it is formed from

— $\left(?\left\langle \overrightarrow{\gamma_{1,g}(x_i)}, \mathcal{I}_1, \overrightarrow{\gamma_{2,g}(x_i)}, \mathcal{I}_2 \right\rangle, h_1 \cdot h_2\right)$,

— T_1,

— $\left(\left\langle u'_1, \overrightarrow{\gamma_{2,g}(x_i)}, \mathcal{I}'_2 \right\rangle, h'_{1|\mathcal{D}'_1} \cdot \widetilde{h_2}\right)$ (recall that the restriction of T changes the initial action of the middle arena from player question to player answer),

— the actions of $\widetilde{T_2}$ which are the copycat pairs on the arena $[\Gamma_2] \to [\Gamma_2]$,

— the actions of $\widetilde{T_3}$ which are the first elements of the copycat pairs on the arena $[\sigma] \to [\sigma]$

Moreover, $T_{|_{\mathbf{F}}([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\Gamma_2])}$ removes part of the heaps of the actions coming from $\widetilde{T_2}$ and $\widetilde{T_3}$, such that this restriction is indeed in $([\Gamma_1 \vdash M_1 : \sigma] \otimes [\Gamma_2])$ since all the actions on $[\Gamma_2] \to [\Gamma_2]$ are copycat pairs, thus in $\mathfrak{id}_{[\Gamma_2]}$.

Then, $T_{|_{\mathbf{F}}([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\tau])}$ is indeed equal to $\left(?\left\langle \overrightarrow{\gamma_{1,g}(x_i)}, \mathcal{I}_1, \overrightarrow{\gamma_{2,g}(x_i)}, \mathcal{I}_2 \right\rangle, h_1 \cdot h_2\right) \cdot T_1 \cdot T_2 \cdot \left(\langle \overrightarrow{u_1, u_2} \rangle, h'_{|D'}\right) \cdot T_3$, due to the construction of $\widetilde{T_2}$ and $\widetilde{T_3}$.

Finally, $\mathbf{P}(T_{|_{\mathbf{F}}([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\Gamma_2])}) \cap \mathbf{P}(T_{|_{\mathbf{F}}([\sigma]\otimes[\Gamma_2]),([\sigma]\otimes[\tau])})$ is equal to $\varnothing$, since h'_1, h_2 are disjoint

We now prove the reverse inclusion. We take T an interaction trace over $([\Gamma_1] \otimes [\Gamma_2]) \to ([\sigma] \otimes [\Gamma_2]) \to ([\sigma] \otimes [\tau])$ such that

— $T_{|_{\mathbf{F}}([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\Gamma_2])}$ is in $([\Gamma_1 \vdash M_1 : \sigma] \otimes [\Gamma_2])$,

— and $T_{|_{\mathbf{F}}([\sigma]\otimes[\Gamma_2]),([\sigma]\otimes[\tau])}$ is in $([\sigma] \otimes [\Gamma_2 \vdash M_2 : \tau])$.

Suppose the T is not empty, so the first action a_1 of T is necessarily of the form

$$\left(?\left\langle \overrightarrow{v_{1,j}}, \mathcal{I}_1, \overrightarrow{v_{2,k}}, \mathcal{I}_2 \right\rangle, h_1 \cdot h_2\right)$$

such that

— we can define two functions $\gamma_{g,1} : \Gamma_{g,1} \to \text{Val}$ and $\gamma_{g,1} : \Gamma_{g,1} \to \text{Val}$ as $\gamma_{g,1}(x_j) = v_{1,j}$ and $\gamma_{g,2}(x_k) = v_{2,k}$.
— h_i are in $\mathbf{Cl}(\nu_L(\text{codom}(\gamma_{g,i})))$ and are functional-free, and are disjoint,
— $\mathcal{I}_i = \mathcal{I}_i^\Gamma \cup \mathcal{I}_i^h$ with $\mathcal{I}_i^\Gamma \sim_{\mathbb{P}} \Gamma_{f,i}$ and $\mathcal{I}_i^h = \nu_{\mathbb{P}}(\text{codom}(h_i))$.

We now consider the maximal prefix $a_1 \cdot T_1$ of $T_{|_{\mathbf{F}([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\Gamma_2])}}$ until the player answer to this first opponent question appears to this trace. Then,

$$T_1 \in \mathbf{Tr}\langle \gamma_{1,g}(M_1), \varepsilon, \mathcal{I}_1 \cdot \mathcal{I}_2, h_1 \cdot h_2, D_1 \cup D_2 \rangle$$

and $a_1 \cdot T_1$ is also a prefix of T, since there could have been no actions in any other arena before an answer to the first opponent question appears.

Then, if this player answer a_2 exists, it can be written as $\left(\left\langle \overrightarrow{u_1, v_{2,k}, \mathcal{I}_2} \right\rangle, h_1' \cdot \widetilde{h_2} \right)$ with $\widetilde{h_2} \sim_{\mathbb{P}} h_2$ and

$$\langle \gamma_{1,g}(M_1), \varepsilon, \mathcal{I}_1, h_1, D_1 \rangle \xrightarrow{T_1} \langle u_1', \gamma_1, \mathcal{I}_1', h_1', D_1' \rangle$$

Indeed, by definition of $([\Gamma_1 \vdash M_1 : \sigma] \otimes [\Gamma_2])$, it has to play copycat over $[\Gamma_2]$. Moreover, h_1' is indeed disjoint from h_2 since $\mathbf{P}(T_{|_{\mathbf{F}([\Gamma_1]\otimes[\Gamma_2]),([\sigma]\otimes[\Gamma_2])}}) \cap \mathbf{P}(T_{|_{\mathbf{F}([\sigma]\otimes[\Gamma_2]),([\sigma]\otimes[\tau])}})$ is equal to $\varnothing$. The rest of the proof is done in the same way. ∎

Then, since the product of strongly single-threaded trace strategies over a trace term-arena $\mathfrak{A}$ is cartesian, one can define the diagonal morphism $\mathfrak{d}_{\mathfrak{A}} : \mathfrak{A} \to \mathfrak{A} \otimes \mathfrak{A}$. Using it, we get the following corollary of the previous theorem, when M_1, M_2 share the same typing context Γ.

Corollary 2: *Let M_1, M_2 two terms such that $\underbrace{\Gamma_f, \Gamma_g}_{\Gamma} \vdash M_1 : \sigma$ and $\Gamma_f, \Gamma_g \vdash M_2 : \tau$. Then $(\mathfrak{d}_{[\Gamma]})^\dagger; ([\Gamma \vdash M_1 : \sigma] \otimes [\Gamma \vdash M_2 : \tau])$ is equal to the nominal closure of the set of even-length traces belonging to*

$$T_1 \cdot T_2 \cdot ((\langle \bar{u_1} \rangle, \langle \bar{u_2} \rangle), h'_{|D'}) \cdot \mathbf{Tr}\langle \Diamond, \gamma', \mathcal{I}', h', D' \rangle$$

s.t. *there exists $\gamma_g : \Gamma_g \to \text{Val}$ and $h \in \mathbf{Cl}(\nu_L(\text{codom}(\gamma_g)))$ with D defined as $\text{dom}(h)$ and $\mathcal{I}$ as $\text{dom}(\Gamma_f)$, and we have*

— $\langle \gamma_g(M_1), \varepsilon, \mathcal{I}, h, D \rangle \xrightarrow{T_1} \langle u_1', \gamma_1, \mathcal{I}_1, h_1, D_1 \rangle$ *with u_1' a value,*
— $\langle \gamma_g(M_2), \gamma_1, \mathcal{I}_1, h_1, D_1 \rangle \xrightarrow{T_2} \langle u_2', \gamma_2, \mathcal{I}', h_2, D_2 \rangle$ *with u_2' a value,*
— $D' = \text{discl}(\{u_1, u_2\}, h', D_2)$,
— $h' = h_2[\overrightarrow{l_i \hookrightarrow x_i}]$ *with l_i ranging over $h_{2,\text{fn}} \cap D'$ and the x_i not in γ_2 and distinct from u_1, u_2,*
— *γ' is the extension of γ_2 with $(u_i \mapsto u_i')$ if u_i' is of functional type–otherwise $u_i' = u_i$– and with all the $(x_i \mapsto h_2(l_i))$ for l_i ranging over $h_{2,fn|D'}$.*

5.3.5 Denotation of Applications

Using this characterization of pairings of trace-strategies representing terms, we can now decompose the trace semantics of application of two terms. To do so, we consider the function $\mathfrak{ev}_{\mathfrak{A},\mathfrak{B}}$ which transforms a trace of $((\mathfrak{A} \Rightarrow \mathfrak{B}) \otimes \mathfrak{B}) \Rightarrow \mathfrak{B}$ into a trace of $\mathfrak{B}$, defined as $(\mathfrak{L}^{\mathfrak{B}}_{\mathfrak{A}\Rightarrow\mathfrak{B},\mathfrak{A}})^{-1}(\mathfrak{id}_{\mathfrak{A}\Rightarrow\mathfrak{B}})$. The inverse of $\mathfrak{L}^{\mathfrak{B}}_{\mathfrak{A}\Rightarrow\mathfrak{B},\mathfrak{A}}$ is indeed defined here because $\mathfrak{id}_{\mathfrak{A}\Rightarrow\mathfrak{B}}$ is strongly single-threaded.

Using the previous characterization of pairings, we can prove the following theorem which decompose traces of applications of two terms.

Theorem 18: *Let M_1, M_2 two terms such that $\Sigma;\Gamma \vdash M_1 : \sigma \to \tau$ and $\Sigma;\Gamma \vdash M_2 : \sigma$. Then $[\Sigma;\Gamma \vdash M_1 M_2 : \tau] = (\mathfrak{d}_{[\Gamma]})^{\dagger}; ([\Sigma;\Gamma \vdash M_1 : \sigma \to \tau] \otimes [\Sigma;\Gamma \vdash M_2 : \sigma]); \mathfrak{ev}_{[\sigma],[\tau]}$.*

5.3.6 Full Abstraction of Trace Semantics

Using this correspondence with game semantics, we can import the full abstraction result of [MT11b] (Theorem 36) to trace semantics. Notice that Laird has already proven this result directly [Lai07], where he needed a complex proof of definability of trace strategy to achieve it.

Theorem 19: *For M_1, M_2 terms of RefML such that $\Sigma;\Gamma \vdash M_1, M_2 : \tau$, we have $\Sigma;\Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$ iff $\mathbf{comp}([\Sigma;\Gamma \vdash M_1 : \tau]) = \mathbf{comp}([\Sigma;\Gamma \vdash M_2 : \tau])$.*

5.4 Trace Semantics for GroundML

In this section, we refine our trace semantics to handle also GroundML. This is done in two steps:

- we rephrase the notion of visibility defined in Section 5.1.2 for trace strategies in order to characterize trace strategies coming from terms of GroundML.
- we refine the interactive reduction of Figure 5.2 to restrict the power of contexts, since they cannot keep track of disclosed λ-abstractions during the whole execution but only inside the current scope, because of the lack of higher-order references.

We adapt the characterizations of ground-refererences terms of RefML of [MT11a] into trace semantics, namely the fact that they are represented by *visible* strategies, which restrict the pointer structure of actions of a trace T to be in the view of T.

Definition 34 (View). *The* view $\ulcorner T \urcorner$ *of a legal trace T on $\mathfrak{A}$ is a subsequence of T defined by induction:*

- $\ulcorner \varepsilon \urcorner = \varepsilon$,
- $\ulcorner (a,h) \urcorner = (a,h)$,
- $\ulcorner T' \cdot (a,h) \cdot T'' \cdot (a',h') \urcorner = \ulcorner T' \urcorner \cdot (a,h) \cdot (a',h')$ *when (a',h') is justified by (a,h).*

Then, as in game semantics, we say that a trace T is *P-visible* (resp. O-visible) if for all $T' \cdot (a,h) \sqsubseteq^{\text{even}} T$ (resp. $T' \cdot (a,h) \sqsubseteq^{\text{odd}} T$ with (a,h) a *player* (resp. *opponent*) action-with-heap, the justifier of (a,h) is in $\ulcorner T' \urcorner$. A trace strategy is said to be *X-visible* if all its traces are X-visible, for $X \in \{P,O\}$.

In our setting, justification is defined using freshness of name pointers in traces, so we introduce the notion of *available X-name pointers*, for $X \in \{P,O\}$, to reason on the view:

Definition 35 (Available Name-Pointers). *We define the set of* available opponent *(resp.* player*)* name pointers $\mathbf{Av_O}(T)$ *(resp.* $\mathbf{Av_O}(T)$*) inductively as*

- $\mathbf{Av_X}(\varepsilon) \stackrel{def}{=} \varnothing$,
- $\mathbf{Av_X}((a,h)) \stackrel{def}{=} \nu^X_{\mathbb{P}}(a)$,
- $\mathbf{Av_X}(T \cdot (a_1,h_1) \cdot T' \cdot (a_2,h_2)) \stackrel{def}{=} \mathbf{Av_X}(T) \cup \nu^X_{\mathbb{P}}(a_2)$ *when* (a_1,h_1) *is justified by* (a_2,h_2).

with $X \in \{P,O\}$.

Notice in the previous definition that we do not need to consider name pointers in heaps, since we consider terms of GroundML, which do not store any functions in heaps, and for which contexts cannot disclose higher-order references. Moreover, in the third clause, we do not need to consider the name pointers of a_1 since its polarity is opposed as the one of a_2.

We now link the definition of available name pointers to the notion of view:

Lemma 14: *Let T a justified trace, then for all name pointers $x \in \mathbf{Av_X}(T)$ ($X \in \{P,O\}$), x is introduced by an action (a,h) which appears in $\ulcorner T \urcorner$.*

5.4.1 Ground-refererences Terms of RefML and P-visible strategies

The characterization of trace-strategies coming from terms of GroundML is given by the following theorem.

Theorem 20: *Let M a term of GroundML such that $\Sigma;\Gamma \vdash M : \tau$, then $[\Sigma;\Gamma \vdash M : \tau]$ is a P-visible trace strategy.*

Even if the theorem is similar to the one stated in game semantics, the proof is radically different because the definition of the denotation is operational and so the proof can no longer be done by inductively showing that the visibility condition is preserved. To conduct the proof in this operational setting, we need to analyze the structure of traces more closely. We first notice a crucial property of the interactive reduction of such terms,

namely the fact that the undisclosed part of the heap cannot store name pointers. Thus, when reducing (M, h) to (M', h'), we know that the name pointers contained in M' are also in M.

We now prove that a term M' appearing in the interactive reduction of M via a trace T only contains name pointers from $\mathbf{Av_O}(T)$.

Lemma 15: *Let M a term of GroundML such that $\Sigma;\Gamma \vdash M : \tau$, and $((a_0, h_0) \cdot T \cdot (a, h)) \in [\Sigma;\Gamma \vdash M : \tau]$ s.t. $\langle M, \gamma, \mathcal{I}, h_0, D\rangle \xrightarrow{T} \langle M' \cdot \overrightarrow{K_i}, \gamma', \mathcal{I}', h', D'\rangle$ Then $\nu^O_{\mathbb{P}}(M') \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T)$.*

Proof By induction on the length of T. If $T = \varepsilon$ it is straightforward from the definition of $[\Sigma;\Gamma \vdash M : \tau]$. Otherwise, let us write T as $T_1 \cdot (a_1, h_{1|D_1}) \cdot T_2 \cdot (a_2, h_{2|D_2})$ with $(a_2, h_{2|D_2})$ an opponent action and $(a_1, h_{1|D_1})$ its (player) justifier. We have $\langle M, \gamma, \mathcal{I}, h_0, D\rangle \xrightarrow{T_1 \cdot (a_1, h_{1|D_1})} \langle \overrightarrow{K'_j}, \gamma_1, \mathcal{I}_1, h_1, D_1\rangle \xrightarrow{T_2 \cdot (a_2, h_{2|D_2})} \langle M' \cdot \overrightarrow{K_i}, \gamma', \mathcal{I}', h_2, D_2\rangle$. Let us reason by case analysis on a_2:

- Suppose a_2 is an opponent answer $\langle v\rangle$, then we have $M' = K[v]$. We have $\mathbf{Av_O}((a_0, h_0) \cdot T) = \mathbf{Av_O}((a_0, h_0) \cdot T_1) \cup \nu^O_{\mathbb{P}}(v)$, so from $\nu^O_{\mathbb{P}}(M_2) = \nu^O_{\mathbb{P}}(K[\bullet]) \cup \nu^O_{\mathbb{P}}(v)$ we just have to prove that $\nu^O_{\mathbb{P}}(K[\bullet]) \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T_1)$. Then a_1 is a player question $\bar{f}\,\langle v'\rangle$ such that $\langle M, \gamma, \mathcal{I}, h, D\rangle \xrightarrow{T_1} \langle M_1 \cdot \overrightarrow{K_i}, \gamma_1, \mathcal{I}_1, h'_1, D'_1\rangle$ and $(M_1, h'_1) \mapsto (K[f\, v'], h''_1)$. Then the induction hypothesis on T_1 gives us $\nu^O_{\mathbb{P}}(M_1) \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T_1)$, and we conclude using $\nu^O_{\mathbb{P}}(K[f\, v']) \subseteq \nu^O_{\mathbb{P}}(M_1)$.
- Suppose that a_2 is an opponent question $x\,\langle v\rangle$ justified by the player action $(a_1, h_{1|D_1})$, with $\gamma(x) = u$ and $M_2 = u\, v$. From $\mathbf{Av_O}((a_0, h_0) \cdot T) = \mathbf{Av_O}((a_0, h_0) \cdot T_1) \cup \nu^O_{\mathbb{P}}(v)$, we simply have to prove that $\nu^O_{\mathbb{P}}(u) \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T_1)$. Suppose $\langle M, \gamma, \mathcal{I}, h_0, D\rangle \xrightarrow{T_1} \langle M'_1 \cdot \overrightarrow{K'_j}, \gamma_1, \mathcal{I}, h'_1, D'_1\rangle$ and $(M'_1, h'_1) \mapsto^* (M''_1, h''_1)$, then the induction hypothesis gives us that $\nu^O_{\mathbb{P}}(M'_1) \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T_1)$. We reason by case analysis on a_1:
  - If $a_1 = \langle \bar{x}\rangle$, then $M''_1 = u$. Since $\nu^O_{\mathbb{P}}(u) \subseteq \nu^O_{\mathbb{P}}(M'_1)$ we get that $\nu^O_{\mathbb{P}}(u) \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T_1)$.
  - Otherwise $a_1 = \bar{f}\,\langle x\rangle$, then $M''_1 = K[f\, u]$. Since $\nu^O_{\mathbb{P}}(K[f\, u]) \subseteq \nu^O_{\mathbb{P}}(M'_1)$ we get that $\nu^O_{\mathbb{P}}(u) \subseteq \mathbf{Av_O}((a_0, h_0) \cdot T_1)$. ∎

From this lemma we get directly the following corollary:

Corollary 3: *Let M a term of GroundML such that $\Sigma;\Gamma \vdash M : \tau$, and $T \cdot (\bar{x}\,\langle u\rangle, h) \in [\Sigma;\Gamma \vdash M : \tau]$, then $x \in \mathbf{Av_O}(T)$.*

We can finally prove Theorem 20 using the conjunction of lemma 14 and corollary 3.

5.4.2 Full Abstraction for GroundML

We have seen in Section 5.4.1 that terms of GroundML give rise to P-visible traces. However, the trace semantics of Section 5.2.3 is not fully abstract, since there are still

P-Intern $\langle (M,\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \longrightarrow \langle (M',\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h',D\rangle$
(when $(M,h)\mapsto_{nd}(M',h')$)

P-AnsG $\langle (v,\iota,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\langle\overline{v}\rangle,r\cdot\xi,h'_{|D})} \langle\overrightarrow{K_i},\gamma',\mathcal{I},h,D',\mathcal{A}\rangle$
(v of type ι, $D'=\mathrm{discl}(v,h,D)$)

P-Ans $\langle (v,\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\langle\bar{x}\rangle,r\cdot\xi,h_{|D})} \langle\overrightarrow{K_i},\gamma',\mathcal{I},h,D,x\cdot\mathcal{A}\rangle$
(x fresh, $\gamma'=\gamma\cdot[x\hookrightarrow(v,\tau,r\cdot\xi)]$)

P-QuestG $\langle (K[x\,v],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\bar{x}\langle v\rangle,\ell\cdot\xi',h_{|D'})} \langle (K[\bullet_{\iota,\xi'}],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D',\mathcal{A}'\rangle$
($(x,\iota\to\sigma,\xi',\mathcal{A}')\in\mathcal{I}$, v of type ι, $D'=\mathrm{discl}(v,h,D)$)

P-Quest $\langle (K[x\,v],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D\rangle \xrightarrow{(\bar{x}\langle y\rangle,\ell\cdot\xi',h_{|D})} \langle (K[\bullet_{\sigma',\xi'}],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma',\mathcal{I},h,D,y\cdot\mathcal{A}'$
($(x,\sigma\to\sigma',\xi',\mathcal{A}')\in\mathcal{I}$, y fresh, $\gamma'=\gamma\cdot[y\hookrightarrow(v,\sigma,\ell\cdot\xi')]$)

O-AnsG $\langle (K[\bullet_{\iota,\xi'}],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D,\mathcal{A}'\rangle \xrightarrow{(\langle v\rangle,r\cdot\xi',h'_{|D'})} \langle (K[v],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h',D'\rangle$
(v of type ι, $v\notin\mathrm{dom}(h)\cap\overline{D}$)

O-Ans $\langle (K[\bullet_{\sigma,\xi'}],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I},h,D,\mathcal{A}'\rangle \xrightarrow{(\langle x\rangle,r\cdot\xi',h'_{|D'})} \langle (K[x],\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
(x fresh, $\mathcal{I}'=\mathcal{I}\cdot(x,\sigma,r\cdot\xi',\mathcal{A}')$)

O-QuestG $\langle\overrightarrow{K_i},\gamma,\mathcal{I},h,D,\mathcal{A}\rangle \xrightarrow{(x\langle v\rangle,\ell\cdot\xi,h'_{|D'})} \langle (u\,v,\tau,\xi,\mathcal{A})\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
($x\in\mathcal{A}$, $\gamma(x)=(u,\iota\to\tau,\xi)$, v of type ι and not in $\mathrm{dom}(h)\cap\overline{D}$)

O-Quest $\langle\overrightarrow{K_i},\gamma,\mathcal{I},h,D,\mathcal{A}\rangle \xrightarrow{(x\langle y\rangle,\ell\cdot\xi,h'_{|D'})} \langle (u\,y,\tau,\xi)\cdot\overrightarrow{K_i},\gamma,\mathcal{I}',h',D'\rangle$
($x\in\mathcal{A}$, $\gamma(x)=(u,\sigma\to\tau,\xi)$, y fresh, $\mathcal{I}'=\mathcal{I}\cdot(y,\tau,\ell\cdot\xi,\mathcal{A})$)

in all O-Rules: $D'=\mathrm{discl}(v,h',D)$, $h'_{|\overline{D'}}=h_{|\overline{D}}$ and $h'_{|D'}$ is closed)

Figure 5.3: Definitions of the interaction semantics for GroundML

traces which are not generated by an interaction between contexts and terms of GroundML. To get full abstraction, we need to constrain traces to be O-visible by modifying the interactive reduction to control the scope of name pointers such that only pointers appearing in the view of an action are available. We present the modified interactive reduction in Figure 5.3. Firstly, notice that since we do not have higher-order references in our language, we do not need to extend $\mathcal{I}$ and γ with pointers representing functions stored in the disclosed part of the heap. Then, we keep track of available player name pointers, represented by a set $\mathcal{A}$, in different places of the configuration:

— in opponent configurations $\langle \overrightarrow{K_i}, \gamma, \mathcal{I}, h, D, \mathcal{A} \rangle$, representing the current available player name pointers,

— within each element $(M, \tau, \xi, \mathcal{A})$ or $(K[\bullet_{\tau,\xi}], \sigma, \xi', \mathcal{A})$ of the execution stack, representing the player name pointers available when opponent add this element to the execution stack via a question.

— within each opponent name pointers $x \in \mathcal{I}$, representing the player name pointers available when x has been introduced by an opponent action.

Using this set of available player name pointers, we can control the questions Opponent can ask, as shown by the condition $x \in \mathcal{A}$ in the rules **O-QuestG** and **O-Quest**. This idea is formalized in the following lemma:

Lemma 16: *Let* $(\mathrm{i}, h_0) \cdot T \in [\Sigma; \Gamma_f \vdash M : \tau]$ *such that* $\langle (M, \tau, \xi, \varnothing), \gamma, \mathcal{I}, h, \mathcal{D} \rangle \xrightarrow{T} \langle \overrightarrow{K_i}, \gamma', \mathcal{I}', h', \mathcal{D}', \mathcal{A} \rangle$ *then* $\mathbf{Av_O}(T) = \mathcal{A}$.

Using our restricted interactive reduction, we define the ground interpretation of a judgment $[\Sigma; \Gamma \vdash M : \tau]_G$ in the same way than in Definition 27, with the empty set of pointers associate to M in the execution stack. From Lemma 16, we can deduce that the ground interpretation always gives rise to visible strategy.

Theorem 21: *Let M a term of GroundML such that $\Sigma; \Gamma \vdash M : \tau$, then $[\Sigma; \Gamma \vdash M : \tau]_G$ is a visible strategy (*i.e. *both P-visible and O-visible).*

Proof Let $\gamma_g : \Gamma_g \to \mathrm{Val}$ a ground-variable substitution, $h : \Sigma$ and $T \cdot a \in \mathbf{Tr}\langle M, \cdot, \widetilde{\Gamma_f}, h, \widetilde{\Sigma} \rangle$. If a is a player action, the P-visibility of T can be deduce from theorem 20, since we can easily see that $[\Sigma; \Gamma \vdash M : \tau]_G \subseteq [\Sigma; \Gamma \vdash M : \tau]$.

If a is an opponent action (o, h), let us decompose T into $T' \cdot (p', h') \cdot T''$ such that (o, h) is justified by (p, h'). Then we must prove that (p, h') is in $\ulcorner T \urcorner$.

— If o is an opponent answer, then due to the well-bracketing condition, (p, h') is indeed in $\ulcorner T \urcorner$.

— If o is an opponent question $f\,\langle x \rangle$, then the depth associated to f is lower than the depth of the last unanswered player question. Thus using lemma 16, we get that (p, h'), the player action which introduces f, is in the view of $\ulcorner T \urcorner$. ∎

It is straightforward to see that the notion of visibility introduced here corresponds exactly to the usual notion of visibility of game semantics. So importing the full abstraction result of [MT12], we get the following theorem:

$$
\begin{array}{lcl}
\mathrm{S}_V[\![\mathrm{Int}]\!]_{\mathcal{I}}(u) & \stackrel{def}{=} & \langle \bar{u} \rangle \\
\mathrm{S}_V[\![\mathrm{Int} \to \sigma]\!]_{\mathcal{I}}(v) & \stackrel{def}{=} & \left(\left\langle \bar{\dagger} \right\rangle \cdot \bigcup_{n \in \mathbb{Z}} ((\dagger \, \langle n \rangle) \cdot \mathrm{S}_E[\![\sigma]\!]_{\mathcal{I}}(v \ \widehat{n})) \right) \\
\mathrm{S}_V[\![\tau \to \sigma]\!]_{\mathcal{I}}(v) & \stackrel{def}{=} & \left(\left\langle \bar{\dagger} \right\rangle \cdot (\dagger \, \langle y \rangle) \cdot \underbrace{\mathrm{S}_E[\![\sigma]\!]_{\mathcal{I} \cdot (y,\tau)}(v \ y)}_{S} \right) \text{ with } y \notin (\mathcal{I} \cup \nu^O_{\mathbb{P}}(S)) \\
\mathrm{S}_K[\![\mathrm{Int}, \tau]\!]_{\mathcal{I}}(K) & \stackrel{def}{=} & \bigcup_{n \in \mathbb{Z}} (\langle \widehat{n} \rangle \cdot \mathrm{S}_E[\![\tau]\!]_{\mathcal{I}}(K[\widehat{n}])) \\
\mathrm{S}_K[\![\sigma, \tau]\!]_{\mathcal{I}}(K) & \stackrel{def}{=} & (\langle y \rangle \cdot \underbrace{\mathrm{S}_E[\![\tau]\!]_{\mathcal{I} \cdot (y,\sigma)}(K[y])}_{S}) \text{ with } y \notin (\mathcal{I} \cup \nu^O_{\mathbb{P}}(S)) \\
\\
\mathrm{S}_E[\![\tau]\!]_{\mathcal{I}}(M) & \stackrel{def}{=} & \begin{cases} (\bar{f} \, \langle \dagger \rangle \, , \mathrm{S}_V[\![\sigma]\!]_{\mathcal{I}}(v)) \cdot \mathrm{S}_K[\![\sigma', \tau]\!]_{\mathcal{I}}(K) \\ \qquad \text{if } M \mapsto^* K[f \, v] \text{ and } (f, \sigma \to \sigma') \in \mathcal{I} \\ \mathrm{S}_V[\![\tau]\!]_{\mathcal{I}}(v) \qquad \text{if } M \mapsto^* v \end{cases}
\end{array}
$$

Figure 5.4: Definitions of the callback structure

Theorem 22 (Full Abstraction for GroundML): *Let M_1, M_2 two terms of GroundML such that $\Sigma; \Gamma \vdash M_1, M_2 : \tau$. Then $\Sigma; \Gamma \vdash M_1 \simeq^G_{ctx} M_1 : \tau$ iff* $\mathbf{comp}([\Sigma; \Gamma \vdash M_1 : \tau]_G) = \mathbf{comp}([\Sigma; \Gamma \vdash M_2 : \tau]_G)$.

When M_1, M_2 are divergent-free terms of GroundML, they cannot possibly diverge, so we do not need the restriction to complete trace plays in the theorem.

5.5 Decidability of the Pure Fragment

In this last part, as a preview to the next chapters, we show how to perform surgery on traces to get decidability properties. We consider the pure fragment of RefML, *i.e.* diverging-free terms M such that $\Gamma \vdash M : \tau$ with τ not containing any subtype $\mathrm{ref}\, \tau$, and M not containing any subterms $\mathrm{ref}\, M'$ (and consequently no subterms $M_1 := M_2$ and $!M_1$). Thus, in the sequel, configurations are simply written $\langle E \cdot \overrightarrow{K_i}, \gamma, \mathcal{I} \rangle$.

5.5.1 Callback Structure

To introspect a pure term, we introduce the notion of *callback structure*, depicted in Figure 5.4. They are formed as list of trees of *partial actions* in pAction, where player name pointers are omitted and replaced by a symbol †. Such structure is in fact really closed to *abstract Bohm trees* [Cur98], which have been used to represent strategies in a call-by-name setting [CH09]. $\mathrm{S}_E[\![\tau]\!]_{\mathcal{I}}$ and $\mathrm{S}_V[\![\tau]\!]_{\mathcal{I}}$ are elements of $\mathcal{S}truct_n$ where n is the maximum of the order of τ and the types of $\mathcal{I}$—with $\mathcal{S}truct_0 \stackrel{def}{=} \mathrm{Val}$ and $\mathcal{S}truct_{n+1} \stackrel{def}{=}$

$List(\text{pAction} \cup (\text{pAction} \times \mathcal{S}truct_n) \cup (\mathcal{P}(\mathcal{S}truct_n)))$. When Int is restricted to bounded integers, $\mathrm{S}_E[\![\tau]\!]_{\mathcal{I}}(M)$ is clearly computable.

For example, consider the term $M \stackrel{def}{=} \lambda \mathtt{f}.\lambda \mathtt{n}.\mathtt{if\ n} = \mathtt{0\ then\ f}(\lambda \mathtt{x}.\mathtt{x})\ \mathtt{else\ f}(\lambda \mathtt{x}.\mathtt{n})$ of type $\tau \stackrel{def}{=} ((\text{Int} \to \text{Int}) \to \text{Unit}) \to \text{Int} \to \text{Unit}$. The reader can check that its callback structure is

$$\begin{aligned}\mathrm{S}_V[\![\tau]\!]_{\varnothing}(M) \;=\; \left\langle \bar{\dagger} \right\rangle \cdot \dagger\langle f \rangle \cdot \left\langle \bar{\dagger} \right\rangle \cdot \Big(\Big\{ \dagger \left\langle \widehat{0} \right\rangle \cdot (\bar{f}\,\langle \dagger \rangle, \left\langle \bar{\dagger} \right\rangle \cdot \dagger \langle \widehat{n} \rangle \cdot \left\langle \bar{\widehat{n}} \right\rangle) \mid n \in \mathbb{Z} \Big\} \cup \\ \Big\{ \dagger \langle \widehat{m} \rangle \cdot (\bar{f}\,\langle \dagger \rangle, \left\langle \bar{\dagger} \right\rangle \cdot \dagger \langle \widehat{n} \rangle \cdot \left\langle \bar{\widehat{m}} \right\rangle) \mid m \in \mathbb{Z}^*, n \in \mathbb{Z} \Big\} \Big)\end{aligned}$$

To compare callback structures, we use a notion of nominal equivalence, indexed by a span on (opponent) name pointers.

Definition 36. *We say that two callback structures S_1, S_2 of $\mathcal{S}truct_n$ are equivalent for a span e_O, written $S_1 \simeq^n_{e_O} S_2$, when*

- $\left\langle \bar{k} \right\rangle \simeq^n_{e_O} \left\langle \bar{k} \right\rangle$
- $\left\langle \bar{\dagger} \right\rangle \cdot S'_1 \simeq^n_{e_O} \left\langle \bar{\dagger} \right\rangle \cdot S'_2$ *if* $S'_1 \simeq^n_{e_O} S'_2$
- $((\bar{f}_1\,\langle \dagger \rangle, S'_1) \cdot S''_1) \simeq^{n+1}_{e_O} ((\bar{f}_2\,\langle \dagger \rangle, S'_2) \cdot S''_2)$ *if* (f_1, f_2) *is in* e_O, $S'_1 \simeq^n_{e_O} S'_2$ *and* $S''_1 \simeq^{n+1}_{e_O} S''_2$,
- $\{(\langle j \rangle, S^j_1) \mid j \in \mathbb{Z}\} \simeq^n_{e_O} (\{(\langle j \rangle, S^j) \mid j \in \mathbb{Z}\}$ *if* $S^j_1 \simeq^n_{e_O} S^j_2$ *for all* $j \in \mathbb{Z}$.
- $(\langle x_1 \rangle \cdot S_1) \simeq^n_{e_O} (\langle x_2 \rangle \cdot S_2)$ *if* $S_1 \simeq^n_{e_O \cdot (x_1, x_2)} S_2$.

When integers are bounded, this equivalence is again decidable since we reason on finite objects. In the following, we omit the index n when it is clear from the context. We also define an equivalence between two substitutions γ_1, γ_2, written $\gamma_1 \simeq_{e_P, e_O} \gamma_2$, as $\mathrm{S}_V[\![\tau]\!]_{e_{O,1}}(\gamma_1(x_2)) \simeq_{e_O} \mathrm{S}_V[\![\tau]\!]_{e_{O,2}}(\gamma_1(x_2))$ for all $(x_1, x_2, \tau) \in e_P$.

Notice that callback structures are not closed by nominal equivalence on name pointers (*i.e.* on variables). Indeed, in the definition of $\mathrm{S}_V[\![\tau \to \sigma]\!]_{\mathcal{I}}(v)$ and $\mathrm{S}_K[\![\sigma, \tau]\!]_{\mathcal{I}}(K)$, the choice of the fresh opponent name pointer y is supposed to be deterministic. But we can prove that the callback structure equivalence is preserved by nominal equivalence:

Lemma 17: *Let M_1, M_2 two ground-closed pure terms and e_O a span on their free functional variables such that $\mathrm{S}_E[\![\tau]\!]_{e_{O,i}}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(M_2)$. Let π_1, π_2 two finitely supported permutations on name pointers, then writing e'_O for the span formed by the $\pi_1(x_1), \pi_2(x_2)$ with $(x_1, x_2) \in e_O$* (i.e. *the span which result from the action of $\pi_1 \times \pi_2$ on e_O seen as a nominal element on $\mathbb{P} \times \mathbb{P}$), we have $\mathrm{S}_E[\![\tau]\!]_{\pi_1 * M_1}(e'_{O,1})(\pi_1 * M_1) \simeq_{e'_O} \mathrm{S}_E[\![\tau]\!]_{\pi_1 * M_1}(e'_{O,2})(\pi_2 * M_2)$.*

The two following lemmas are used to show that equivalence of callback structures of terms is preserved by reduction.

Lemma 18: *Let e_P and e_O two spans respectively on player and opponent name pointers, and γ_1, γ_2 two substitutions whose domains are equal respectively to $e_{P,1}, e_{P,2}$, with $\gamma_1 \simeq_{e_P,e_O} \gamma_2$. Let M_1, M_2 two terms such that $\mathrm{S}_E[\![\tau]\!]_{e_{O,1}}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(M_2)$ and $T_i \in \mathbf{Tr}\langle M_i, \gamma_i, e_{O,i}\rangle$ such that $T_1 \simeq_{e_P,e_O} T_2$ and $\langle M_i, \gamma_i, e_{O,i}\rangle \xrightarrow{T_i} \langle v_i, \gamma'_i, e'_{O_i}\rangle$.*

Then there exists $e'_P \sqsupseteq e_P$ and $e'_O \sqsupseteq e_O$ such that $\gamma'_1 \simeq_{e'_P,e'_O} \gamma'_2$ and $\mathrm{S}_V[\![\tau]\!]_{e'_{O,1}}(v_1) \simeq_{e'_O} \mathrm{S}_V[\![\tau]\!]_{e'_{O,2}}(v_2)$.

Lemma 19: *Let e_P and e_O two spans respectively on player and opponent name pointers, and γ_1, γ_2 two substitutions whose domains are equal respectively to $e_{P,1}, e_{P,2}$, with $\gamma_1 \simeq_{e_P,e_O} \gamma_2$. Let $T_i \in \mathbf{Tr}\langle \Diamond, \gamma_i, e_{O,i}\rangle$ such that $T_1 \simeq_{e_P,e_O} T_2$ and $\langle \Diamond, \gamma_i, e_{O,i}\rangle \xrightarrow{T_i} \langle \Diamond, \gamma'_i, e'_{O,i}\rangle$.*

Then there exists $e'_P \sqsupseteq e_P$ and $e'_O \sqsupseteq e_O$ such that $\gamma'_1 \simeq_{e'_P,e'_O} \gamma'_2$.

Proof The two lemmas are proved by a mutual induction on the length of the callback structure of γ_i.

Proof (Proof of Lemma 18) From $\mathrm{S}_E[\![\tau]\!]_{e_{O,1}}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(M_2)$, there is two possibilities:

— Both $M_i \mapsto v_i$ with $\mathrm{S}_V[\![\tau]\!]_{e_{O,1}}(v_1) \simeq_{e_O} \mathrm{S}_V[\![\tau]\!]_{e_{O,2}}(v_2)$. Then we get directly the wanted property.

— Or both $M_i \mapsto K_i[f_i\, u_i^1]$ such that there exists two types σ, σ' with $(f_1, f_2) \in e_{O,\sigma\to\sigma'}$ and $\mathrm{S}_V[\![\sigma]\!]_{e_{O,1}}(u_1^1) \simeq_{e_O} \mathrm{S}_V[\![\sigma]\!]_{e_{O,2}}(u_2^1)$. Then we can decompose T_i into $\bar{f}_i\left\langle\widehat{u_i^1}\right\rangle \cdot U_i^1 \cdot \langle u_i^2\rangle \cdot U_i^2$ with

$$\begin{array}{lll} \langle M_i, \gamma_i, e_{O,i}\rangle & \xrightarrow{\bar{f}_i\left\langle\widehat{u_i^1}\right\rangle} & \langle K_i[\bullet], \gamma_i^1, e_{O,i}\rangle \\ & \xrightarrow{U_i^1} & \langle K_i[\bullet], \gamma_i^2, e_{O,i}^1\rangle \\ & \xrightarrow{u_i^2} & \langle K_i[u_i^2], \gamma_i^2, e_{O,i}^2\rangle \end{array}$$

such that $\widehat{u_i^1} = u_i^1$, $\gamma_i^1 = \gamma_i$ and $e_P^1 = e_P$ if σ is equal to Int, otherwise $\widehat{u_i^1}$ is equal to some fresh name x_i, $\gamma_i^1 = \gamma_i \cdot (x_i \mapsto u_i^1)$ and $e_P^1 = e_P \cdot (x_1, x_2, \sigma)$. Using the induction hypothesis from the second lemma, we get the existence of $e_P^2 \sqsupseteq e_P^1$ and $e_O^1 \sqsupseteq e_O$ such that $\gamma_1^1 \simeq_{e_P^2,e_O^1} \gamma_2^1$. Then, we conclude using the induction hypothesis on the first lemma, since $\mathrm{S}_E[\![\tau]\!]_{e_{O,1}^2}(K_1[u_1^2]) \simeq_{e_O^2} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}^2}(K_2[u_2^2])$

Proof (Proof of Lemma 19) We can suppose than $T_i = (f_i\,\langle u_i\rangle) \cdot U_i^1 \cdot U_i^2$ with $(f_1, f_2) \in e_{O,\sigma\to\sigma'}$, $\gamma_i(f_i) = v_i$ and

$$\begin{array}{lll} \langle \Diamond, \gamma_i, e_{O,i}\rangle & \xrightarrow{f_i\left\langle u_i^1\right\rangle} & \langle v_i\ u_i, \gamma_i, e_{O,i}^1\rangle \\ & \xrightarrow{U_i^1} & \langle \Diamond, \gamma_i^1, e_{O,i}^2\rangle \\ & \xrightarrow{U_i^2} & \langle \Diamond, \gamma'_i, e_{O,i}^3\rangle \end{array}$$

such that and $e_O^1 = e_O$ if σ is equal to Int, otherwise $e_O^1 = e_O \cdot (y_1, y_2, \sigma)$ for some fresh opponent name pointers y_i, . Using the induction hypothesis on the first lemma, we get

the existence of $e_P^1 \sqsupseteq e_P$ and $e_O^2 \sqsupseteq e_O^1$ such that $\gamma_1^1 \simeq_{e_P^1, e_O^2} \gamma_2^1$. Then using the induction hypothesis on the second lemma, we can conclude, since we get the existence of $e_P^2 \sqsupseteq e_P^1$ and $e_O^3 \sqsupseteq e_O^1$ such that $\gamma_1' \simeq_{e_P^2, e_O^3} \gamma_2'$. ∎

Those two lemmas are used to prove that two equivalent substitution functions generates two sets of equivalent traces.

Lemma 20: *Let e_P, e_O two spans respectively on player and opponent name pointers, and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions such that $\gamma_1 \simeq_{e_P, e_O} \gamma_2$. Then $\mathrm{Tr}\langle \Diamond, \gamma_1, e_{O,1} \rangle \simeq_{e_P, e_O} \mathrm{Tr}\langle \Diamond, \gamma_2, e_{O,2} \rangle$*

This lemma is proven via a mutual induction with the following three theorems which state that the callback structure gives a sound and complete model of the pure fragment of RefML. Lemma 20 is used for the soundness of callback structures, while the completeness is rather straightforward due to the fact that the callback structure of a term M can be seen as a set of subtraces of M.

Theorem 23: *Let v_1, v_2 a pair of pure ground-closed values of RefML, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions such that $\gamma_1 \simeq_{e_P, e_O} \gamma_2$. Then $\mathrm{S}_V[\![\tau]\!]_{e_{O,1}}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(v_2)$ iff $\mathrm{Tr}\langle v_1, \gamma_1, e_{O,1} \rangle \simeq_{e_P, e_O} \mathrm{Tr}\langle v_2, \gamma_2, e_{O,2} \rangle$.*

Theorem 24: *Let K_1, K_2 a pair of pure ground-closed contexts of RefML, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions such that $\gamma_1 \simeq_{e_P, e_O} \gamma_2$. Then $\mathrm{S}_K[\![\sigma, \tau]\!]_{e_{O,1}}(K_1) \simeq_{e_O} \mathrm{S}_K[\![\sigma, \tau]\!]_{e_{O,2}}(K_2)$ iff $\mathrm{Tr}\langle K_1, \gamma_1, e_{O,1} \rangle \simeq_{e_P, e_O} \mathrm{Tr}\langle K_2, \gamma_2, e_{O,2} \rangle$.*

Theorem 25: *Let M_1, M_2 a pair of pure ground-closed terms of RefML, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions such that $\gamma_1 \simeq_{e_P, e_O} \gamma_2$. Then $\mathrm{S}_E[\![\tau]\!]_{e_{O,1}}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(M_2)$ iff $[\langle M_1, \gamma_1, e_{O,1} \rangle] \simeq_{e_P, e_O} [\langle M_2, \gamma_2, e_{O,2} \rangle]$.*

Proof The three theorems and the lemmas are proved by a mutual induction on types and on the length of the callback structures.

Proof (Lemma 20) let $T_1 \in \mathbf{Tr}\langle \Diamond, \gamma_1, e_{O,1} \rangle$, we want to build $T_2 \in \mathbf{Tr}\langle \Diamond, \gamma_2, e_{O,2} \rangle$ such that $T_1 \simeq_{e_P, e_O} T_2$. We can decompose T_1 into $(f_1\, \langle u_1^1 \rangle) \cdot U_1^1 \cdot \left\langle \overline{u_1^2} \right\rangle \cdot U_1^2$ with $\gamma_1(f_1) = v_1$, $U_1^1 \cdot \left\langle \overline{u_1^2} \right\rangle \in \mathbf{Tr}\langle v_1\ u_1^1, \gamma_1^1, e_{O,1}^1 \rangle$, $U_1^2 \in \mathbf{Tr}\langle \Diamond, \gamma_1^2, e_{O,1}^2 \rangle$ and

$$\begin{array}{lll} \langle \Diamond, \gamma_1, e_{O,1} \rangle & \xrightarrow{f_1 \langle u_1^1 \rangle} & \langle v_1 u_1^1, \gamma_1, e_{O,1}^1 \rangle \\ & \xrightarrow{U_1^1} & \langle \widetilde{u_1^2}, \gamma_1^1, e_{O,1}^2 \rangle \\ & \xrightarrow{\left\langle \overline{u_1^2} \right\rangle} & \langle \Diamond, \gamma_1^2, e_{O,1}^2 \rangle \end{array}$$

Let f_2 such that $(f_1, f_2) \in e_P$, *i.e.* there exists σ, σ such that $(f_1, f_2) \in e_{P, \sigma \to \sigma'}$. We reason by case analysis on σ:

— If σ is equal to Int, then $e^1_{O,1} = e_{O,1}$ so we define u^1_2 as u^1_1 and e^1_O as e_O,
— Otherwise, u^1_1 is equal to some opponent name pointer y_1 and $e^1_{O,1} = e_{O,1} \cdot y_1$. So we define u^1_2 as y_2, a fresh opponent name pointer not in $e_{O,2}$, and $e^1_O = e_O \cdot (y_1, y_2, \sigma)$

Then, from $\gamma_1 \simeq_{e_P,e_O} \gamma_2$ we get $\mathrm{S}_V[\![\sigma \to \sigma']\!]_{e_{O,1}}(v_1) \simeq_{e_O} \mathrm{S}_V[\![\sigma \to \sigma']\!]_{e_{O,2}}(v_2)$, so $\mathrm{S}_E[\![\sigma']\!]_{e^1_{O,1}}(v_1\ u^1_1) \simeq_{e^1_O} \mathrm{S}_E[\![\sigma']\!]_{e^1_{O,2}}(v_2\ u^1_2)$. Then by induction hypothesis on Terms, we get the existence of $U^1_2 \cdot \left\langle \bar{u^2_2} \right\rangle \in \mathbf{Tr}\langle v_1\ u^1_2, \gamma_1, e^1_{O,1}\rangle$ such that $U^1_1 \cdot \left\langle \bar{u^2_1} \right\rangle \simeq_{e_P,e^1_O} U^1_2 \cdot \left\langle \bar{u^2_2} \right\rangle$.

Finally, using lemma 19, we get the existence of two spans e^2_P and e^2_O extending e_P and e^1_O such that $\gamma^2_1 \simeq_{e^2_P,e^2_O} \gamma^2_2$. So applying the induction hypothesis, we get the existence of $U^2_2 \in \mathbf{Tr}\langle \Diamond, \gamma^2_2, e^2_{O,2}\rangle$ such that $U^2_1 \simeq_{e^2_P,e^2_O} U^2_2$. Thus, defining T_2 as $(f_2\,\langle u^1_2\rangle) \cdot U^1_2 \cdot \left\langle \bar{u^2_2} \right\rangle \cdot U^2_2$ we get that $T_1 \simeq_{e_P,e_O} T_2$. ∎

Proof (Full Abstraction on Values) We have $\mathrm{Tr}\langle v_i, \gamma_i, e_{O,i}\rangle = \left\langle \bar{\tilde{v_i}} \right\rangle \cdot \mathrm{Tr}\langle \Diamond, \gamma^1_1, e_{O,1}\rangle$ with $\tilde{v_i} = v_i$ and $\gamma^1_i = \gamma_i$ if τ is equal to Int, otherwise $\tilde{v_i}$ is equal to a player name pointer f_i not in $e_{P,i}$ and $\gamma^1_i = \gamma_i \cdot (f_i \mapsto v_i)$. Then we get the direct implication using the induction hypothesis on lemma 20. To prove the reverse implication, we reason by case analysis on τ:

— If τ is equal to Int, then $\mathrm{Tr}\langle v_1, \gamma_1, e_{O,1}\rangle \simeq_{e_P,e_O} \mathrm{Tr}\langle v_2, \gamma_2, e_{O,2}\rangle$ gives us that $v_1 = v_2$.
— Otherwise, defining $e^1_P = e_P \cdot (f_1, f_2, \tau)$, we have $\mathrm{Tr}\langle \Diamond, \gamma^1_1, e_{O,1}\rangle \simeq_{e^1_P,e_O} \mathrm{Tr}\langle \Diamond, \gamma^1_2, e_{O,2}\rangle$. If $\tau = \mathrm{Int} \to \sigma$, then let $n \in \mathbb{Z}$, we have $\mathrm{Tr}\langle v_1\ \hat{n}, \gamma^1_1, e_{O,1}\rangle \simeq_{e^1_P,e_O} \mathrm{Tr}\langle v_2\ \hat{n}, \gamma^1_2, e_{O,2}\rangle$ so we conclude using the induction hypothesis on terms. Otherwise, we have $\tau = \sigma \to \sigma'$, and defining $e^1_O = e_O \cdot (y_1, y_2, \sigma)$ we get

$$\mathrm{Tr}\langle v_1\ y_1, \gamma^1_1, e^1_{O,1}\rangle \simeq_{e^1_P,e^1_O} \mathrm{Tr}\langle v_2\ y_2, \gamma^1_2, e^1_{O,2}\rangle$$

and we conclude using again the induction hypothesis on terms. ∎

Proof (Full Abstraction on Context) We first prove the direct implication. Suppose that

$$\mathrm{S}_K[\![\sigma, \tau]\!]_{e_{O,1}}(K_1) \simeq_{e_O} \mathrm{S}_K[\![\sigma, \tau]\!]_{e_{O,2}}(K_2)$$

then taking $T_1 \in \mathbf{Tr}\langle K_1, \gamma_1, e_{O,1}\rangle$, we have $T_1 = U^1_1 \cdot \langle u_1\rangle \cdot U^2_1$ with

$$\begin{array}{rcl} \langle K_1, \gamma_1, e_{O,1}\rangle & \xrightarrow{U^1_1} & \langle K_1, \gamma^1_1, e^1_{O,1}\rangle \\ & \xrightarrow{u_1} & \langle K_1[u_1], \gamma^2_1, e^2_{O,1}\rangle \end{array}$$

We want to build $T_2 \in \mathbf{Tr}\langle K_2, \gamma_2, e_{O,2}\rangle$ such that $T_1 \simeq_{e_P,e_O} T_2$. Using the induction hypothesis on the first lemma, we get the existence of $U^1_2 \in \langle \Diamond, \gamma_1, e_{O,1}\rangle$ such that $U^1_1 \simeq_{e_P,e_O} U^1_2$ and $\langle K_2, \gamma_2, e_{O,2}\rangle \xrightarrow{U^1_2} \langle K_2, \gamma^1_2, e^1_{O,2}\rangle$. Then, we reason by case analysis on σ

— If $\sigma = \mathrm{Int}$, then $e^2_{O,1} = e^1_{O,1}$ and we define u_2 as u_1 and e^2_O as e^1_O.
— If σ is of functional type, then u_1 is an opponent name pointer x_1 and $e^2_{O,1} = e^1_{O,1} \cdot (x_1, \sigma)$. Then, taking x_2 an opponent name pointer not in $e^1_{O,2}$, we define e^2_O as $e^1_O \cdot (x_1, x_2, \sigma)$.

So from $\mathrm{S}_K[\![\sigma,\tau]\!]_{e^1_{O,1}}(K_1) \simeq_{e^1_O} \mathrm{S}_K[\![\sigma,\tau]\!]_{e^1_{O,2}}(K_2)$ we get that

$$\mathrm{S}_E[\![\tau]\!]_{e^2_{O,1}}(K_1[u_1]) \simeq_{e^2_O} \mathrm{S}_E[\![\tau]\!]_{e^2_{O,2}}(K_2[u_2])$$

and the induction hypothesis on terms gives us the existence of

$$U^2_2 \in \mathbf{Tr}\langle K_2[u_2], \gamma^1_2, e^2_{O,2}\rangle$$

such that $U^2_1 \simeq_{e^1_P, e^2_O} U^2_2$. Then defining T_2 as $U^1_2 \cdot \langle u_2\rangle \cdot U^2_2$, we get $T_1 \simeq_{e_P,e_O} T_2$.

Let us prove now the reverse implication. If $\mathbf{Tr}\langle K_1,\gamma_1,e_{O,1}\rangle \simeq_{e_P,e_O} \mathbf{Tr}\langle K_2,\gamma_2,e_{O,2}\rangle$ we reason by case analysis on σ:

— If σ is of type Int, then for all $n \in \mathbb{Z}$ and $T_1 \in \mathbf{Tr}\langle K_1[\hat{n}],\gamma_1,e_{O,1}\rangle$, we have $\langle\hat{n}\rangle \cdot T_1 \in \mathbf{Tr}\langle K_i,\gamma_i,e_{O,i}\rangle$. Thus there exists $T_2 \in \mathbf{Tr}\langle K_2[\hat{n}],\gamma_2,e_{O,2}\rangle$ such that $T_1 \simeq_{e_P,e_O} T_2$. Using a symmetric reasoning, we get that $\mathbf{Tr}\langle K_1[\hat{n}],\gamma_1,e_{O,1}\rangle \simeq_{e_P,e_O} \mathbf{Tr}\langle K_2[\hat{n}],\gamma_2,e_{O,2}\rangle$. So using the induction hypothesis on terms, we get that

$$\mathrm{S}_E[\![\tau]\!]_{e_{O,1}}(K_1[\hat{n}]) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(K_2[\hat{n}])$$

and since this is true for all $n \in \mathbb{Z}$, we get $\mathrm{S}_K[\![\sigma,\tau]\!]_{e_{O,1}}(K_1) \simeq_{e_O} \mathrm{S}_K[\![\sigma,\tau]\!]_{e_{O,2}}(K_2)$.

— If σ is functional, then taking x_i a fresh player name pointer not in $e_{P,i}$, we define $e^1_P = e_P \cdot (x_1,x_2,\sigma)$. Then taking $T_1 \in \mathbf{Tr}\langle K_1[x_1],\gamma^1_1,e_{O,1}\rangle$ we have $\langle\bar{x_1}\rangle \cdot T_1 \in \mathbf{Tr}\langle K_i,\gamma_i,e_{O,i}\rangle$. Thus there exists $T_2 \in \mathbf{Tr}\langle K_2[x_2],\gamma^1_2,e_{O,2}\rangle$ such that $T_1 \simeq_{e^1_P,e_O} T_2$. Using a symmetric reasoning, we get that $\mathbf{Tr}\langle K_1[x_1],\gamma^1_1,e_{O,1}\rangle \simeq_{e^1_P,e_O} \mathbf{Tr}\langle K_2[x_2],\gamma^1_2,e_{O,2}\rangle$. So using the induction hypothesis on terms, we get that $\mathrm{S}_E[\![\tau]\!]_{e^1_{O,1}}(K_1[x_2]) \simeq_{e^1_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(K_2[x_2])$, *i.e.* $\mathrm{S}_K[\![\sigma,\tau]\!]_{e_{O,1}}(K_1) \simeq_{e_O} \mathrm{S}_K[\![\sigma,\tau]\!]_{e_{O,2}}(K_2)$. ∎

Proof (Full Abstraction on Terms) Suppose that both $M_i \mapsto v_i$, then $\mathrm{Tr}\langle M_i,\gamma_i,e_{O,i}\rangle = \mathrm{Tr}\langle v_i,\gamma_i,e_{O,i}\rangle$ and $\mathrm{S}_E[\![\tau]\!]_{e_{O,1}}(M_i) = \mathrm{S}_V[\![\tau]\!]_{e_{O,1}}(v_i)$ and we conclude using the induction hypothesis on values.

If both $M_i \mapsto K_i[f_i\,u_i]$, then $\mathrm{Tr}\langle M_i,\gamma_i,e_{O,i}\rangle = (\bar{f_i}\,\langle\widetilde{u_i}\rangle)\cdot\mathrm{Tr}\langle K_i,\gamma^1_i,e_{O,i}\rangle$ with $\widetilde{u_i} = u_i$ and $\gamma^1_i = \gamma_i$ if $\sigma = \mathrm{Int}$, otherwise $\widetilde{u_i}$ is equal to some fresh player name pointer x_i and $\gamma^1_i = \gamma_i \cdot (x_i \hookrightarrow u_i)$. Moreover $\mathrm{S}_E[\![\tau]\!]_{e_{O,i}}(M_i) = (f_i, \mathrm{S}_V[\![\tau]\!]_{e_{O,i}}(u_1)) \cdot \mathrm{S}_K[\![\sigma',\tau]\!]_{e_{O,i}}(K_i)$ And we conclude easily using the induction hypothesis on contexts. ∎

Finally, using the full abstraction of trace semantics, we get a complete characterization of contextual equivalence in term of equivalence of the callback structure of terms.

Corollary 4: *Let (M_1,M_2) a pair of pure terms of RefML such that $\Gamma_g,\Gamma_f \vdash M_1,M_2 : \tau$. Then $\cdot;\Gamma_g,\Gamma_f \vdash M_1 \simeq_{ctx} M_2 : \tau$ iff $\mathrm{S}_E[\![\tau]\!]_{\Gamma_f}(\gamma(M_1)) \simeq_{\widetilde{\Gamma_f}} \mathrm{S}_E[\![\tau]\!]_{\Gamma_f}(\gamma(M_2))$ for all substitutions $\gamma : \Gamma_g \to \mathrm{Val}$.*

$$
\begin{array}{lll}
(K[M], \mathbb{C}) & \mapsto_s & (K[M'], \mathbb{C}) \text{ when } (M, \varnothing) \mapsto (M', \varnothing) \\
(K[x == x'], \mathbb{C}) & \mapsto_s & (K[\mathbf{true}], \mathbb{C} \wedge (x = x')) \quad \text{or} \quad (K[\mathbf{false}], \mathbb{C} \wedge (x \neq x')) \\
(K[x == \widehat{n}], \mathbb{C}) & \mapsto_s & (K[\mathbf{true}], \mathbb{C} \wedge (x = n)) \quad \text{or} \quad (K[\mathbf{false}], \mathbb{C} \wedge (x \neq n)) \\
(K[\widehat{n} == x], \mathbb{C}) & \mapsto_s & (K[\mathbf{true}], \mathbb{C} \wedge (x = n)) \quad \text{or} \quad (K[\mathbf{false}], \mathbb{C} \wedge (x \neq n)) \\
(K[x + \widehat{n}], \mathbb{C}) & \mapsto_s & (K[z], \mathbb{C} \wedge (z = x + n)) \text{ with } z \text{ fresh in } \mathbb{C} \\
(K[\widehat{n} + x], \mathbb{C}) & \mapsto_s & (K[z], \mathbb{C} \wedge (z = x + n)) \text{ with } z \text{ fresh in } \mathbb{C} \\
(K[x + x'], \mathbb{C}) & \mapsto_s & (K[z], \mathbb{C} \wedge (z = x + x')) \text{ with } z \text{ fresh in } \mathbb{C} \\
(K[\texttt{if } b \texttt{ then } M_1 \texttt{ else } M_2], \mathbb{C}) & \mapsto_s & (K[M_1], \mathbb{C} \wedge (b = 0)) \quad \text{or} \quad (K[M_2], \mathbb{C} \wedge (b = 1))
\end{array}
$$

Figure 5.5: Definition of Symbolic Execution

Notice that the same definition of $\mathrm{S}_E[\![\tau]\!](M)$ can be used to deal with pure terms of GroundML, and the same proof of full abstraction goes through. This means that the notion of contextual equivalence of RefML and GroundML is the same on pure terms, which is to our knowledge a new result.

Finally, to decide the equivalence of two pure terms, we simply have to compute their callback structures and then check if they are equivalent—which is always possible when the type Int is bounded.

5.5.2 Symbolic Execution and Decidability of the unbounded fragment

$\mathrm{S}_E[\![\tau]\!](M)$ is in general an infinite object when we do not bound the type Int. However, due to the fact that terms cannot have infinite branching, it is possible to represent $\mathrm{S}_E[\![\tau]\!](M)$ in a finite way.

To do so, we introduce a *symbolic execution* in Figure 5.5, which reason logically on integer variables with arithmetic constraints. Those constraints are collected in a set $\mathbb{C}$, so that the reduction is defined on pairs $(M, \mathbb{C})$. A more general symbolic execution, defined for GroundML, will be presented in Chapter 7.

Then, this symbolic execution is used to define the *symbolic callback structure* of terms in Figure 5.6. Notice that $\mathbb{S}_E[\![\tau]\!]_{\mathcal{I}}(M)$ is now defined as a set.

The equivalence of two symbolic callback structures is then defined as a logical formula of Presburger Arithmetic.

Definition 37. *Two symbolic callback structures S_1, S_2 of $\mathcal{S}truct_n$ are* symbolically equivalent *for a span e_O, written $S_1 \simeq^n_{e_O} S_2$, as*

— $\left\langle \bar{t_1} \right\rangle \simeq^n_{e_O} \left\langle \bar{t_2} \right\rangle \stackrel{def}{=} t_1 = t_2$,

— $((\bar{f}_1 \langle \dagger \rangle, S_1) \cdot S'_1) \simeq^{n+1}_{e_O} ((\bar{f}_2 \langle \dagger \rangle, S_2) \cdot S'_2) \stackrel{def}{=}$
$\begin{cases} (S'_1 \simeq^n_{e_O} S'_2 \wedge S''_1 \simeq^n_{e'_O} S''_2) & \text{if } (f_1, f_2) \in e_O, \\ \text{False} & \text{otherwise.} \end{cases}$

$$
\begin{array}{lcl}
\mathbb{S}_V[\![\mathrm{Int}]\!]_{\mathcal{I}}(t) & \stackrel{def}{=} & \left\langle \bar{t} \right\rangle \\
\mathbb{S}_V[\![\mathrm{Int} \to \sigma]\!]_{\mathcal{I}}(v) & \stackrel{def}{=} & (\left\langle \bar{\dagger} \right\rangle, \dagger \langle z \rangle, \mathbb{S}_E[\![\sigma]\!]_{\mathcal{I}}(v\ z)) \\
\mathbb{S}_V[\![\tau \to \sigma]\!]_{\mathcal{I}}(v) & \stackrel{def}{=} & (\left\langle \bar{\dagger} \right\rangle, \dagger \langle y \rangle, \mathbb{S}_E[\![\sigma]\!]_{\mathcal{I}\cdot(y,\tau)}(v\ y)) \text{ with } y \notin \mathcal{I} \\
\mathbb{S}_K[\![\mathrm{Int}, \tau]\!]_{\mathcal{I}}(K) & \stackrel{def}{=} & (\langle z \rangle, \mathbb{S}_E[\![\tau]\!]_{\mathcal{I}}(K[z])) \\
\mathbb{S}_K[\![\sigma, \tau]\!]_{\mathcal{I}}(K) & \stackrel{def}{=} & (\langle x \rangle, \mathrm{S}_E[\![\tau]\!]_{\mathcal{I}\cdot(x,\sigma)}(K[x])) \\
\mathbb{S}_E[\![\tau]\!]_{\mathcal{I}}(M) & \stackrel{def}{=} & \{((\bar{f}\ \langle\dagger\rangle, \mathrm{S}_V[\![\sigma]\!]_{\mathcal{I}}(v)), \mathbb{C}) \cdot \mathrm{S}_K[\![\sigma', \tau]\!]_{\mathcal{I}}(K) \mid (M, \varnothing) \mapsto_s^* (K[f\ v], \mathbb{C}) \\
 & & \text{and } (f, \sigma \to \sigma') \in \mathcal{I}\} \cup \{(\mathrm{S}_V[\![\tau]\!]_{\mathcal{I}}(v), \mathbb{C}) \mid (M, \varnothing) \mapsto_s^* (v, \mathbb{C})\}
\end{array}
$$

Figure 5.6: Definitions of the symbolic callback structure

— $\{(S_{j_1}, \mathbb{C}_{j_1}) \mid j_1 \in J_1\} \simeq_{e_O}^{n+1} \{(S_{j_2}, \mathbb{C}_{j_2}) \mid j_2 \in J_2\} \stackrel{def}{=} \bigwedge_{(j_1,j_2)\in J_1\times J_2} (\mathbb{C}_{j_1} \wedge \mathbb{C}_{j_2}) \Rightarrow (S_{j_1} \simeq_{e_O}^{n} S_{j_2})$
— $(\langle x_1 \rangle, S_1) \simeq_{e_O}^{n} (\langle x_2 \rangle, S_1) \stackrel{def}{=} \forall x_1, x_2.(x_1 = x_2) \implies S_1 \simeq_{e_O}^{n} S_2$
— $(\dagger \langle x_1 \rangle \cdot S_1) \simeq_{e_O}^{n} (\dagger \langle x_2 \rangle \cdot S_2)$ *if* $S_1 \simeq_{e_O\cdot(x_1,x_2)}^{n} S_2$.

To get the decidability result, we have to prove that this symbolic equivalence coincide with the equivalence on callback structures introduce in the previous section.

Theorem 26: *Let M_1, M_2 two ground-closed terms of type τ and e_O a span on their free functional variables. Then $\mathrm{S}_E[\![\tau]\!]_{e_{O,1}}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_{O,2}}(M_2)$ iff $\mathbb{S}_E[\![\tau]\!]_{e_{O,1}}(M_1) \simeq_{e_O} \mathbb{S}_E[\![\tau]\!]_{e_{O,2}}(M_2)$.*

To prove this theorem, we must relate the symbolic execution of Figure 5.5 with the operational reduction of Figure 3.8.

Lemma 21: *Let M a term such that $M \mapsto^* M'$. Then there exists a set of Presburger constraint $\mathbb{C}$ and a substitution $\alpha : \mathrm{FV}(C) \to \mathrm{Int}$ such that*
— *$\alpha(\mathbb{C})$ is true,*
— *$(M, \varnothing) \mapsto_s^* (M'', \mathbb{C})$,*
— *$\alpha(M'') = M'$.*

Lemma 22: *Let M a term such that $(M, \varnothing) \mapsto_s^* (M'', \mathbb{C})$ and there exists a substitution $\alpha : \mathrm{FV}(C) \to \mathrm{Int}$ such that $\alpha(\mathbb{C})$ is true. Then there exists M' such that $M \mapsto_s^* M'$ and $\alpha(M'') = M'$.*

Then, it is clear that free logical variables of the formula $\mathbb{S}_E[\![\tau]\!]_{e_O,1}(M_1) \simeq_{e_O} \mathbb{S}_E[\![\tau]\!]_{e_O,2}(M_2)$ are the free logical variables of M_1 and M_2. So we can show that substitutions of logical variables go throught the symbolic equivalence, via the following three lemmas.

Lemma 23: $\alpha(\mathbb{S}_V[\![\tau]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathbb{S}_V[\![\tau]\!]_{e_O,2}(v_2))$ *iff* $\mathbb{S}_V[\![\sigma]\!]_{e_O,1}(\alpha(v_1)) \simeq_{e_O} \mathbb{S}_V[\![\tau]\!]_{e_O,2}(\alpha(v_2))$

Lemma 24: $\alpha(\mathbb{S}_K[\![\sigma,\tau]\!]_{e_O,1}(K_1) \simeq_{e_O} \mathbb{S}_K[\![\sigma,\tau]\!]_{e_O,2}(K_2))$
iff $\mathbb{S}_K[\![\sigma,\tau]\!]_{e_O,1}(\alpha(K_1)) \simeq_{e_O} \mathbb{S}_K[\![\sigma,\tau]\!]_{e_O,2}(\alpha(K_2))$

Lemma 25: $\alpha(\mathbb{S}_E[\![\tau]\!]_{e_O,1}(M_1) \simeq_{e_O} \mathbb{S}_E[\![\tau]\!]_{e_O,2}(M_2))$ *iff* $\mathbb{S}_E[\![\tau]\!]_{e_O,1}(\alpha(M_1)) \simeq_{e_O} \mathbb{S}_E[\![\tau]\!]_{e_O,2}(\alpha(M_2))$

Finally, using these lemmas (which will be proved in a more general setting in Chapter 7), we can prove Theorem 26 and the two following theorems on contexts and values, by mutual induction.

Theorem 27: *Let v_1, v_2 two ground-closed terms of type τ and e_O a span on their free functional variables. Then* $\mathrm{S}_V[\![\tau]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathrm{S}_V[\![\tau]\!]_{e_O,2}(v_2)$ *iff* $\mathbb{S}_E[\![\tau]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathbb{S}_E[\![\tau]\!]_{e_O,2}(v_2)$.

Theorem 28: *Let K_1, K_2 two ground-closed contexts of type τ and e_O a span on their free functional variables. Then* $\mathrm{S}_K[\![\sigma,\tau]\!]_{e_O,1}(K_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_O,2}(K_2)$ *iff* $\mathbb{S}_E[\![\tau]\!]_{e_O,1}(K_1) \simeq_{e_O} \mathbb{S}_K[\![\sigma,\tau]\!]_{e_O,2}(K_2)$.

Proof (Theorem 26) Let M_1, M_2 such that $\mathrm{S}_E[\![\tau]\!]_{e_O,1}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_O,2}(M_2)$. Let $M_i'', \mathbb{C}_i$ such that $(M_i, \varnothing) \mapsto_s^* (M_i'', \mathbb{C}_i)$. and α_1, α_2 two substitutions such that $\alpha_i(\mathbb{C}_i)$ is valid. From Lemma 22, we get the existence of M_i' such that $M_i \mapsto^* M_i'$ and $M_i' = \alpha_i(M_i'')$. So from $\mathrm{S}_E[\![\tau]\!]_{e_O,1}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_O,2}(M_2)$ we get that

— either both $M_i' = K_i[f_i\ v_i]$ with $(f_1, f_2) \in e_O$, $\mathrm{S}_V[\![\sigma]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_O,2}(v_2)$ and $\mathrm{S}_K[\![\sigma',\tau]\!]_{e_O,1}(K_1) \simeq_{e_O} \mathrm{S}_K[\![\sigma',\tau]\!]_{e_O,2}(K_2)$. Then, $M_i'' = K_i'[f_i v_i']$ with $K_i = \alpha_i(K_i')$ and $v_i = \alpha_i(v_i')$. Thus, using lemmas 24 and 23 and the induction hypothesis, we get that $\alpha(\mathbb{S}_V[\![\sigma]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_O,2}(v_2))$ and $\alpha(\mathrm{S}_K[\![\sigma',\tau]\!]_{e_O,1}(K_1) \simeq_{e_O} \mathrm{S}_K[\![\sigma',\tau]\!]_{e_O,2}(K_2))$.

— Or both $M_i' = v_i$ with $\mathrm{S}_V[\![\sigma]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_O,2}(v_2)$. Then, $M_i'' = v_i'$ with $v_i = \alpha_i(v_i')$. Thus, using lemma 23 and the induction hypothesis, we get that $\alpha(\mathbb{S}_V[\![\sigma]\!]_{e_O,1}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_O,2}(v_2))$.

Reciprocally, suppose that $\mathbb{S}_E[\![\tau]\!]_{e_O,1}(M_1) \simeq_{e_O} \mathrm{S}_E[\![\tau]\!]_{e_O,2}(M_2)$ and that $M_i \mapsto^* M_i'$. Then using Lemma 21, we get the existence of two sets of Presburger constraint $\mathbb{C}_i$ and two substitution $\alpha_i : \mathrm{FV}(C_i) \to \mathrm{Int}$ such that

— $\alpha_i(\mathbb{C}_i)$ is true,
— $(M_i, \varnothing) \mapsto_s^* (M_i'', \mathbb{C})$,
— $\alpha(M_i'') = M_i'$.

Then from the fact that both $\alpha_i(\mathbb{C}_i)$ are true, we get that

— either both $M_i'' = K_i[f_i\ v_i]$, there exists σ, σ' *s.t.* $(f_1, f_2) \in e_{O,\sigma\to\sigma'}$ and $\alpha(\mathbb{S}_V[\![\sigma]\!]_{e_{O,1}}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_{O,2}}(v_2))$ and $\alpha(\mathbb{S}_K[\![\sigma', \tau]\!]_{e_{O,1}}(K_1) \simeq_{e_O} \mathbb{S}_K[\![\sigma', \tau]\!]_{e_{O,2}}(K_2))$. Then using lemmas 24 and 23 and the induction hypothesis, we get that $\mathrm{S}_V[\![\sigma]\!]_{e_{O,1}}(\alpha_1(v_1)) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_{O,2}}(\alpha_2(v_2))$ and $\mathrm{S}_K[\![\sigma', \tau]\!]_{e_{O,1}}(\alpha_1(K_1)) \simeq_{e_O} \mathrm{S}_K[\![\sigma', \tau]\!]_{e_{O,2}}(\alpha_2(K_2))$. We conclude using the fact that $M_i' = \alpha_i(K_i)[f_i\ \alpha_i(v_i)]$.

— Or both $M_i'' = v_i$ and $\alpha(\mathbb{S}_V[\![\sigma]\!]_{e_{O,1}}(v_1) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_{O,2}}(v_2))$. Then using lemma 23, we get that $\mathrm{S}_V[\![\sigma]\!]_{e_{O,1}}(\alpha_1(v_1)) \simeq_{e_O} \mathrm{S}_E[\![\sigma]\!]_{e_{O,2}}(\alpha_2(v_2))$ and we conclude using the fact that $M_i' = \alpha_i(v_i)$. ∎

5.6 Discussion and Future Works

In this chapter, we have introduced a fully-abstract trace semantics for both RefML and GroundML. We have shown a precise correspondence with the game model of Murawski and Tzevelekos [MT11b]. As a preview of the next two chapters, we have study the pure fragment of RefML to get a simple decidability result on the contextual equivalence. In the following, we are going to study the structure of traces to perform more precise surgery.

Via the relation between trace and games semantics, we have seen that we can equip the trace semantics with a categorical structure, namely a closed-Freyd category. It would be interesting to extend this trace semantics to a polymorphic language, where the mechanism to represent the arenas an action belongs to in terms of tags would become crucial, as in [Lai10].

In a recent work [LS14], Levy and Staton have proposed a general framework to prove such a correspondence between a game and a trace model. It would be intersting to see if we could adapt our correspondence proof in their setting.

5.6.1 Integer References

It would be interesting to extend the characterization of fragments of RefML in trace semantics by dealing with the restriction to integer references, namely RedML, as in [MT13]. We believe that our interactive reduction defined for GroundML can be restricted to give a fully abstract model of RedML, using similar restrictions on the use of name pointers in locations.

5.6.2 Control Operators

As we have said, the well-bracketing condition is hard-wired in the definition of justified traces. To remove it, we would need to specify which question an answer is answering. One possibility to do that would be to use the recent work of Gabbay and Ghica [GG12], which uses nominal sets to represent strategies. In fact, our work is halfway to them: the

way they name questions, and the freshness condition they impose, seems similar to our use of name-pointers and the nominal reasoning we perform on them. It should thus be possible to use their work to also give fresh name to answers actions.

This would allow the study of languages where we need semantically to remove the well-bracketing condition, namely languages with control operators like `call/cc` or exceptions.

5.6.3 Algorithmic Game Semantics

Algorithmic Game Semantics [GM00, HMO11, MT12] is a way to give characterization of contextual equivalence in term of equality of languages recognizable by various forms of automaton. However, those techniques rely on restrictions on types to be able to encode the pointer structure of plays inside the considered languages, so that they are still recognizable. Thus, our result, even if it concerns only the pure fragment, seems out of reach of algorithmic game semantics, since we do not have any restriction on the type of terms. Moreover, we have worked inside RefML, where the absence of visibility condition on strategies makes the encoding of the pointer structure harder.

6

Concrete Logical Relations

In the Chapter 5, we have introduced trace semantics, a fully abstract model of RefML, as a variation of the standard game semantics model. However, these two models are hardly usable as such to reason on the equivalence of terms of RefML. The main reason is that, in these models, denotations of terms—*i.e.* strategies—are infinite objects for which reasoning on equality is hard.

From an other side, there have been a large amount of works to develop operational techniques and model to prove equivalence of programs written in languages with references. Two of the mains ones are *bisimulations* and *logical relations*. Environmental bisimulations [SKS11, Sum09] extend the usual bisimulations defined for first-order languages to higher order languages. To do so, they are defined as set of relations to represent

the evolution of the information disclosed by the term to the context. Coinduction is then used to reason on the resulting equivalence. This use of coinduction is particularly useful to reason on recursion in terms, avoiding any use of step-indexed methods.

Kripke logical relations, which has been presented in the introduction of this thesis, are an other approach to prove contextual equivalence of terms. In a call-by-value setting, they are defined as relations $\mathcal{V}[\![\tau]\!]\,w$ and $\mathcal{E}[\![\tau]\!]\,w$ on values and terms, defined by a mutual induction on the type τ. They use a notion of world w to constrain the heaps used to reduce terms M_1, M_2 to values in the definition of $\mathcal{E}[\![\tau]\!]\,w$. In a recent work [HDNV12], Hur et al. have proposed a way to marry Kripke logical relations and environmental bisimulations into *parametric bisimulations*.

However, all these works are either not complete *w.r.t.* contextual equivalence, or rely on a kind of closure which introduces an infinite quantification over contexts. This is especially the case of logical relations, which usually use a biorthogonal definition to achieve completeness: $\mathcal{E}[\![\tau]\!] \stackrel{def}{=} \{(M_1, M_2) \mid \forall (K_1, K_2) \in \mathcal{K}[\![\tau]\!].(K_1[M_1], K_2[M_2]) \in \mathcal{O}$ and $\mathcal{K}[\![\tau]\!] \stackrel{def}{=} \{(v_1, v_2) \mid \forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!].(K_1[v_1], K_2[v_2]) \in \mathcal{O}\}$ where $\mathcal{O}$ is the set of pairs of equi-diverging terms. One could imagine relate directly a direct and a birthogonal definition of a logical relation. However, this seems out of reach for languages like RefML. More precisely, consider two terms $(M_1, M_2) \in \mathcal{E}[\![\tau]\!]\,w$ where $\mathcal{E}[\![\tau]\!]\,w$ is defined by biorthogonality, and let try to unwind this definition to a direct-style: let h_1, h_2 two heaps satisfying the constraints induced by w, and suppose that both $(M_1, h_1) \mapsto (v_1, h_1')$ and $(M_2, h_2) \mapsto (v_2, h_2')$. Then it is really hard to build a world w' future of w s.t. $(v_1, v_2) \in \mathcal{E}[\![\tau]\!]\,w$ and h_1', h_2' satisfying constraints induced by w'. One need to rely on the discriminating power of contexts in $\mathcal{K}[\![\tau]\!]\,w$, but trying to do that, one comes to study the precise interaction between terms and contexts, which lead to study game and trace semantics.

Here, we propose the first proof of completeness of Kripke logical relations, defined without any kind of closure, for divergent-free terms of GroundML with contexts in RefML. To do so, we heavily rely on the trace semantics introduce in the previous chapter. In fact, we refine this model into Kripke trace semantics.

To achieve this result, we have tweaked the usual definition of Kripke logical relations in multiple ways, to bring it forward to trace semantics. Firstly, our logical relations are defined on terms with free functional variables[1] (*i.e.* opponent name pointers), which are related with a relational environment e which corresponds to a span, as introduced in Section 3.3. So our logical relations $\mathcal{E}[\![\tau]\!]_e$ are indexed by this span, such that

— two λ-abstractions $\lambda x_1.M_1, \lambda x_2.M_2$ are in $\mathcal{V}[\![\tau \to \sigma]\!]_e$, where τ is not a ground type, when $(M_1, M_2) \in \mathcal{E}[\![\sigma]\!]_{e\cdot(x_1,x_2,\tau)}$,

— two (functional) variables (x_1, x_2) are in $\mathcal{V}[\![\tau \to \sigma]\!]_e$ when they are in e.

This is particularly useful to reason abstractly on callbacks.

1. To our knowledge, logical relations are always defined on closed terms. This could have maybe justified to choose a different name for our technique.

Then, we deal globally on the constraints on disclosed locations. Indeed, let consider the two following terms:

$$\begin{array}{rcl} M_1 & = & \texttt{let x = ref0 in } \lambda\texttt{f.fx;x} := \texttt{1} \\ M_2 & = & \texttt{let x = ref0 in } \lambda\texttt{f.fx;x} := \texttt{2} \end{array}$$

They are not equivalent because the location linked to x, which is disclosed via the callback `fx`, does not store the same value at the end of the execution of M_1 and M_2. Usually, such invariants on disclosed locations are imposed via the world w (just like invariants on private locations) and the definition of $\mathcal{V}[\![\text{ref}\,\tau]\!]\,w$. Here, we choose a different approach, and impose globally these equivalences of stored values.

To do so, we need to keep track of disclosed locations. This is done using a notion of span $\mathcal{D}$ between disclosed locations, as in the work of Stark [Sta98]. So worlds contain such a span $\mathcal{D}$, which evolves during the execution of a term, being extended with new disclosed locations. For example, let consider the following two terms:

$$\begin{array}{rcl} M_1 & = & \texttt{let x = ref0 in let y = ref0 in } \lambda\texttt{f.fx;fy} \\ M_2 & = & \texttt{let x = ref0 in } \lambda\texttt{f.fx;fx} \end{array}$$

They are not equivalent because M_1 discloses two different locations via the two callbacks of `f`, while M_2 disclosed the same. Trying to prove this equivalence using our logical relations, this would amount to build a span which contain both (l_x, l_x) and (l_x, l_y) (where l_x, l_y represent locations linked to x, y) which is not possible by definition of spans as partial bijections. This idea of using spans to relate disclosed locations is already present in the work of Dreyer et al. [DNB10] but somehow hidden. Indeed, in their approach, every state of their STSs encodes a partial bijection on locations, which can be used for the same purpose. However, it is used locally in the definition of $\mathcal{V}[\![\text{ref}\,\tau]\!]\,w$, rather than imposed globally in the definition of the constraints deduced from w on heaps, written $(h_1, h_2) : w$.

To achieve model-checking results for our logical relations, which are explained in Chapter 7, we have to constrain the notion of future worlds induces by the transition system. In previous works [PS98, ADR09, DNB10], a future world w' of w can be extended with new invariants (or new STSs of invariants) on the part of heaps which is disjoint from the heaps described by w. This follows the idea that worlds were partitioned into *islands*, each one corresponding to a part of the heap disjoint from the others. Here, we forbid this usage. So the transition system stays fixed from the beginning. This means that it has to foreseen the creation of future locations made by the term: we call *omniscient* such transitions systems which are entirely defined from the beginning. So we make a clear distinction between worlds w, which are simple invariants, and the transition system $\mathcal{A}$, which constraints the evolution of those worlds. Then, the definition of our logical relations $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]\,w$ depends on both a transition system $\mathcal{A}$, which is fixed in the whole inductive definition of the logical relations, and a world w which evolve inside the definition.

We choose to work with *Labeled Transition Systems* (LTSs), rather than State Transition Systems, to represent the evolution of worlds. This allows transition systems to be kept finite in many examples, as justified in Chapter 7. They are mainly used to represent a constraint on the heaps as a pair of pre- and post-conditions between the heaps taken as input and the heaps we get as output.

As we will see in the definition of the equivalence $\Sigma; \Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$ defined using our logical relations, two terms are equivalent if there exists an LTS $\mathcal{A}$ constraining worlds such that (M_1, M_2) is in $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]\, w$ for w "initial" (*i.e.* for which all the locations appearing in M_1, M_2 are disclosed). In order to prove completeness, we must be able to build such a LTS when two terms M_1, M_2 have equal denotations from trace semantics.

So we define a finer notion, *Kripke trace semantics*, which uses worlds to constraint heaps which are used to generate traces. This is particularly useful to control the disclosure of locations, since the definition of $[\Sigma; \Gamma \vdash M : \tau]$ suppose that all locations are disclosed, which is not preserved by reduction. However, this use of worlds to refine trace semantics is independent of any LTS. This means that we have to characterize worlds w and LTS $\mathcal{A}$ which have the adequate properties to enforce, from the fact that M_1, M_2 have their trace denotations equal in w, that $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]\, w$. Such worlds are called *adequate*, which definition can be seen in a first approximation as the "dual" of logical relations. More precisely, we define the set of adequate worlds for two terms M_1, M_2 via a predicate on worlds, written $\overline{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!](M_1, M_2)$. Basically, this predicate imposes that there exists exactly all the necessarily future worlds for w (*w.r.t.* $\mathcal{A}$) to describe the possible evolutions of the heaps we get from the interactive reduction of M_1, M_2. Then, from two terms M_1, M_2, we build a special LTS $\mathcal{A}$, called the *exhaustive LTS*, for which the initial world is indeed in $\overline{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!](M_1, M_2)$. It is build by following all the traces generated by the interactive reduction, adding a new state for each action of such traces.

Finally, we call our logical relations *concrete*, to make a clear distinction with the temporal logical relations defined in the next chapter.

Plan of the Chapter In Section 6.1, we introduce the notion of LTS and worlds. Then, concrete logical relations for the fragment of divergent-free terms of GroundML in RefML are defined in Section 6.2. In Section 6.3, we introduce Kripke trace semantics, a refinement of the previous chapter where worlds are used to constrain heaps in the definition of interactive reduction. This is used in Section 6.4 to prove the soundness of concrete logical relations *w.r.t.* trace semantics.

To tackle the problem of completeness, we introduce the notion of *adequate worlds* in Section 6.5, which is defined dually to logical relations to constrain the shape of LTSs to follow the control flow of terms. This notion is central in the proof of completeness of Section 6.6, where we use an *exhaustive LTS*, defined using interactive reduction, which is indeed adequate.

In Section 6.7.1, we briefly sketch how to define concrete logical relations for full RefML, using a coinductive definition which can be formalized via guarded recursive

$$\begin{array}{lcll} \mathrm{Trans}_S & \stackrel{def}{=} & S^2 \to \mathcal{P}(\mathrm{Heap}^4 \times \mathrm{Span}^2) & (\delta) \\ \mathrm{LTS}_S & \stackrel{def}{=} & \{(\delta_{\mathbf{priv}}, \delta_{\mathbf{pub}}) \mid \delta_{priv}, \delta_{pub} \in \mathrm{Trans}_S, \delta_{\mathbf{pub}} \subseteq \delta_{\mathbf{priv}}\} & (\mathcal{A}) \\ \mathrm{World}_S & \stackrel{def}{=} & S \times \mathrm{Heap}^2 \times \mathrm{Span} & (w) \end{array}$$

Figure 6.1: Definition of Worlds

types. Interestingly, working with a fixed notion of LTS, the usual circularity between the definition of worlds and logical relations disappears. We also sketch in Section 6.7.2 the definition of concrete logical relations for GroundML, where the notion of *backtracking* between states introduced in [DNB10], used to relax the notion of future world, appears.

6.1 LTSs and Worlds

As explained in the introduction, we use a fixed notion of *transition system*, which cannot be extended, to control the evolution of worlds. By evolution, we mean the notion of future world, written $w' \sqsupseteq w$, which represents the evolution of heap-invariants. So in our settings worlds are just simple invariants on heaps. We use labels over transitions rather than states. Such labels control the evolution between two invariants.

6.1.1 First Definitions

The definitions of LTSs and worlds is given in Figure 6.1. We write $\mathcal{P}(X)$ for the powerset of X. Labeled-Transition Systems are defined abstractly over a subset S of a set of states State. This set of states can simply be taken as natural numbers. They are formed by two functions respectively for private and public transitions. Private transitions represent transitions that only terms can take, while public ones can be taken by both terms and contexts. This explains the condition $\delta_{\mathbf{pub}} \subseteq \delta_{\mathbf{priv}}$.

A (public or private) transition associates to a pair of states a predicate on the two heaps and a span taken as input, and the two heaps and the span produced as output, so that, as we said, it can define pre- and post-conditions on them. We do not specify what is the representation of states (*i.e.* elements of State) are, because it does not really matter. In the model checking implementation in the next chapter (Section 7.5), we take it to be the set of integers.

Worlds are simply triples formed by two heaps and a span on locations. The two heaps are the invariants on the private part of the heap, while the span defines the partial bijection between the disclosed part of the heap.

6.1.2 Constraining heaps with worlds

Worlds specify heaps really precisely, since there is no freedom on the private part of the heap, while on the public part, the span is used to induce a nominal equivalence, as shown in the following definition.

Definition 38. *A triple* $(h_1, h_2, \mathcal{D})$ satisfies *the constraints of a world* $w = (h_1^w, h_2^w, \mathcal{D}^w)$, *written* $(h_1, h_2, \mathcal{D}) : w$, *when* $\mathcal{D} = \mathcal{D}^w$, $h_i = h_i^{priv} \cdot h_i^{discl}$ *with* $\mathrm{dom}(h_i^{discl}) = \mathcal{D}_i$ *and* $h_i^{priv} \subseteq h_i^w$, *and* $h_1^{discl} \sim_{\mathcal{D}} h_2^{discl}$.

As we have seen in the previous chapter, one can characterize the nominal the equivalence $h_1 \sim_{\mathcal{D}} h_2$ induced by a span on two heaps as:

$$h_1 \sim_{\mathcal{D}} h_2 \stackrel{def}{=} \forall (l_1, l_2) \in \mathcal{D}_{\mathrm{Int}}.\ h_1(l_1) = h_2(l_2) \ \wedge \forall (l_1, l_2) \in \mathcal{D}_{\mathrm{ref}\,\iota}.\ (h_1(l_1), h_2(l_2)) \in \mathcal{D}_\iota$$

From a triple $(h_1, h_2, \mathcal{D})$ satisfying the constraints of a world w, one can build new heaps h_1', h_2' which also satisfies the constraints of w by changing the values stored in the disclosed part in a related way:

Lemma 26: *Let* $(h_1, h_2, \mathcal{D}) : w$, *then for all pair of closed heaps* h_1^{pub}, h_2^{pub} *such that* $\mathrm{dom}(h_i^{pub}) = \mathcal{D}_i$ *and* $h_1^{pub} \sim_{\mathcal{D}} h_2^{pub}$, *we have* $(h_{1|\overline{\mathcal{D}_1}} \cdot h_1^{pub}, h_{2|\overline{\mathcal{D}_2}} \cdot h_2^{pub}, \mathcal{D}_2) : w$.

6.1.3 Evolution of Worlds

Transitions of an LTS $\mathcal{A}$ are used to define private and public notions of future worlds.

Definition 39. *Let* $\mathcal{A}$ *a LTS and* $w_1 = (s', h_1', h_2', \mathcal{S}')$, $w_2 = (s, h_1, h_2, \mathcal{S})$ *two worlds. We say that* w_2 *is a future* (w.r.t. $\mathcal{A}$) *of* w_1, *written* w_2 ($\mathcal{A}\sqsupseteq$) w_1 *if either* $w_1 = w_2$ *or* $(h_1, h_2, h_1', h_2', \mathcal{S}, \mathcal{S}') \in \delta_{\mathbf{priv}}(s, s')$. *Public futures (noted with* $\sqsupseteq_{\mathbf{pub}}$*) are defined similarly using* $\delta_{\mathbf{pub}}$, *and transitive closure of these two relations are noted* $\sqsupseteq^*$ *and* $\sqsupseteq^*_{\mathbf{pub}}$.

We often omit the prefix $\mathcal{A}$ when it is clear from context. It is important to remark that the current heaps h_1, h_2 of a world specified only the private part of the heap. This is because it is not possible to expect any property on public parts of the heaps, except that they are equivalent (a property that is imposed by the definition of future worlds). Moreover, it is possible for worlds to "garbage collect" part of the private heaps, i.e. h_i'

do not necessarily extend h_i. These properties are useful to build *finite* LTSs for terms which create locations under a λ-abstraction.

Because contexts may create fresh disclosed locations during the execution, we also introduce another notion of (public) future $\sqsupseteq^{\mathcal{F}}_{\mathbf{pub}}$ which forces the existence of a public state creating, for each pair of disclosed locations in the current world storing values of type $\mathrm{ref}\,\iota$, of fresh disclosed locations of $\mathrm{Loc}_{\iota'}$ for each strict subtype of $\mathrm{ref}\,\iota$. One need this special notion of future because the LTS $\mathcal{A}$ is fixed, so that it is normally not possible to extend the world with these new locations coming from the context, that $\mathcal{A}$ has not foreseen.

This is used to model the indirect disclosure (i.e. via the heap) of locations by the context. For example, if the span $w.\mathcal{S}$ of the current world contains a pair of locations $(l_1, l_2) \in \mathrm{Loc}^2_{\mathrm{ref}\,\mathrm{ref}\,\iota}$, then the context can store in $h_i(l_i)$ some fresh locations $l'_i \in \mathrm{Loc}_{\mathrm{ref}\,\iota}$ that themselves store in $h_i(l'_i)$ some fresh locations $l''_i \in \mathrm{Loc}_\iota$. These locations will be automatically disclosed when the control flow returns to the term.

We index this notion of public future with a type τ, written $\sqsupseteq^{\mathcal{F}\tau}_{\mathbf{pub}}$, to force as well the existence of a public state creating fresh disclosed locations of type τ (together with locations on which it may point to in the case of references of references), when τ is equal to some $\mathrm{ref}\,\iota$. This is used to represent the direct disclosure of locations by contexts (i.e. via a callback or a reduction to a value).

To formalize it, we define the set $\mathbf{NewLoc}(\iota)$ of set of pairs of locations for all subtypes of ι : $\mathbf{NewLoc}(\iota) \overset{def}{=} \{\{(l^{\iota_1}_1, l^{\iota_1}_2), \ldots, (l^{\iota_n}_1, l^{\iota_n}_2)\} \mid l^{\iota_j}_i \in \mathrm{Loc}_{\iota_j}, \iota_j \text{ ranges over } \mathbf{subtype}(\iota)\}$. We define $\mathbf{subtype}(\iota)$ as

— $\mathbf{subtype}(\mathrm{Unit}) = \varnothing$, $\mathbf{subtype}(\mathrm{Bool}) = \varnothing$, $\mathbf{subtype}(\mathrm{Int}) = \varnothing$,
— and $\mathbf{subtype}(\mathrm{ref}\,\iota) = \{\mathrm{ref}\,\iota\} \cup \mathbf{subtype}(\iota)$.

Definition 40. $(s', h'_1, h'_2, \mathcal{S}') \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} (s, h_1, h_2, \mathcal{S})$ *when* $h'_i = h_i$ *and there exists a span* $\mathcal{S}''$ *such that* $\mathcal{S}' = \mathcal{S} \uplus \mathcal{S}''$, *and* $\mathcal{S}'' = \biguplus_{(l_1,l_2)\in\mathcal{S}_\iota} \mathcal{D}_\iota$, *with* $\mathcal{D}_\iota \in \mathbf{NewLoc}(\iota)$. *We refine this definition as* $\sqsupseteq^{\mathcal{F}\iota}_{\mathbf{pub}}$ *when* $\mathcal{S}' = \mathcal{S} \uplus \mathcal{S}'' \uplus \mathcal{D}'_\iota$ *with* $\mathcal{D}'_\iota \in \mathbf{NewLoc}(\iota)$.

Notice that $\sqsupseteq^{\mathcal{F}\tau}_{\mathbf{pub}}$ is equivalent to $\sqsupseteq^{\mathcal{F}}_{\mathbf{pub}}$ when τ is not equal to some $\mathrm{ref}\,\iota$.

6.2 Concrete Logical Relations on Traces

Trace semantics, just like usual game semantics, generates infinite objects which are hard to reason on. This is due to the fact that, in presence of references, we cannot check the equivalence of two terms by simply executing them only one time, as we have done in the section 5.5 of the previous chapter, since their executions depend on a heap that evolves. Kripke logical relations allow such a "one shot" reasoning by checking the equivalence of terms on every possible pair of heaps, using worlds to get this information.

Operational semantics defined in Section 3.2.1 does not take into account disclosure of locations, so we refine it into *elementary* interactive reduction $\to$ on simple configurations $\langle M, h, D\rangle$ given by: if $(M, h) \mapsto (M', h')$ with $M' = v$ or $M' = K[f\, v]$ then $\langle M, h, D\rangle \to \langle M', h', \mathrm{discl}(v, h', D)\rangle$.

The definition of concrete logical relations is given in Figure 6.2.

Its main features is to guarantee that:

— resulting values (*i.e.* player answers) are related,

— callbacks (*i.e.* player questions) of the two terms are synchronized— they interrogate equivalent pointers,

— values given to those callbacks are related,

— there exists a future world such that the heaps given in the entry of those callbacks satisfies its invariant.

Notice the use of injection ρ_i in the definition of $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e\,(w, w_0)$ to change name of locations created during the reduction, so that they correspond to the ones in w. Indeed, worlds we have defined in Section 6.1 have fixed names for every locations upfront, fixed by the LTS $\mathcal{A}$. Unfortunately, those names may not correspond to the one created during the operational reduction. To circumvent this problem, we use the injections ρ_i to perform the reallocation.

In the definition, a relational environment e is used to keep track of the arguments given by the contexts to the terms, via a λ-abstraction or via the returned value of a callback. Indeed, compared to usual logical relations, when τ is of functional type, the definition of $\mathcal{V}_{\mathcal{A}}[\![\tau \to \sigma]\!]_e\, w$ do not quantify over logically related values v_1, v_2 of type τ, but rather use fresh variables y_1, y_2 (*i.e.* name pointers), remembering in e that they are related. This relational environment is in fact a span on variable of functional types, or equivalently on opponent name pointers. This way to reason relationally on open terms is in fact really close to *normal form* (or *open*) bisimulation [LL07, LL08], where the same variable is given as arguments to the values we want to relate.

We treat λ-abstractions and variables of functional types in an uniform way. This is necessary to deal with η-equivalence. Indeed, one can prove that $(f, \lambda x.fx) \in \mathcal{V}_{\mathcal{A}}[\![\sigma \to \tau]\!]_{\underbrace{(f, f, \sigma \to \tau)}_{e}}\, w$:

— if σ is atomic, then we simply have to prove that $\forall w_1 \sqsupseteq^* w.\exists w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$, for all $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_e\, w_2$ $(fv_1, (\lambda x.fx)v_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e\, w_2$, which is straightforward from the fact that $((\lambda x.fx)v_2, h) \mapsto (fv_2, h)$ with any $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$,

— if σ is functional, we have to prove that $\forall w_1 \sqsupseteq^* w.\exists w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$, $(fy_1, (\lambda x.fx)y_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_{e\cdot(y_1,y_2,\sigma)}\, w_2$, which amounts to prove that $(y_1, y_2) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_{e\cdot(y_1,y_2,\sigma)}\, w_2$, which is proved in the following lemma.

Lemma 27: *Let e a span on name pointers, and $(x_1, x_2, \sigma \to \tau) \in e$. Then $(x_1, x_2) \in \mathcal{V}_{\mathcal{A}}[\![\sigma \to \tau]\!]_e\, w$ for any world w.*

$$
\begin{array}{lcl}
\mathcal{V}_{\mathcal{A}}\left[\!\left[\mathrm{Unit}\right]\!\right]_e w & \stackrel{def}{=} & \{((),())\} \\
\mathcal{V}_{\mathcal{A}}\left[\!\left[\mathrm{Bool}\right]\!\right]_e w & \stackrel{def}{=} & \{(\mathbf{true},\mathbf{true}),(\mathbf{false},\mathbf{false})\} \\
\mathcal{V}_{\mathcal{A}}\left[\!\left[\mathrm{Int}\right]\!\right]_e w & \stackrel{def}{=} & \{(n,n) \mid n \in \mathrm{Int}\} \\
\mathcal{V}_{\mathcal{A}}\left[\!\left[\mathrm{ref}\,\tau\right]\!\right]_e w & \stackrel{def}{=} & \{(l_1,l_2) \mid (l_1,l_2) \in w.\mathcal{S}_\tau\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{V}_{\mathcal{A}}\left[\!\left[\iota \to \sigma\right]\!\right]_e w & \stackrel{def}{=} & \{(u_1,u_2) \mid \forall w' \sqsupseteq^* w.\exists w'' \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w'.\forall (v_1,v_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\iota\right]\!\right]_e w''. \\
& & \qquad (u_1\ v_1, u_2\ v_2) \in \mathcal{E}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_e (w'',w'')\} \\
\mathcal{V}_{\mathcal{A}}\left[\!\left[\tau \to \sigma\right]\!\right]_e w & \stackrel{def}{=} & \{((u_1,u_2) \mid \forall w' \sqsupseteq^* w.\exists w'' \sqsupseteq^{\mathcal{F}_\tau}_{\mathbf{pub}} w'. \\
& & \qquad (u_1\ y_1, u_2\ y_2) \in \mathcal{E}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_{e\cdot(y_1,y_2,\tau)} (w'',w'')\} \\
& & \qquad\qquad (y_1,y_2 \text{ fresh}, \tau \text{ not ground})
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{K}_{\mathcal{A}}\left[\!\left[\iota,\sigma\right]\!\right]_e (w,w_0) & \stackrel{def}{=} & \{(K_1,K_2) \mid \exists w' \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w.\forall (v_1,v_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\iota\right]\!\right]_e w'. \\
& & \qquad (K_1[v_1], K_2[v_2]) \in \mathcal{E}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_e (w',w_0)\} \\
\mathcal{K}_{\mathcal{A}}\left[\!\left[\tau,\sigma\right]\!\right]_e (w,w_0) & \stackrel{def}{=} & \{(K_1,K_2) \mid \exists w' \sqsupseteq^{\mathcal{F}_\tau}_{\mathbf{pub}} w.(K_1[x_1], K_2[x_2]) \in \mathcal{E}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_{e\cdot(x_1,x_2,\tau)} (w',w_0)\} \\
& & \qquad\qquad (x_2,x_2 \text{ fresh}, \tau \text{ not ground})
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{E}_{\mathcal{A}}\left[\!\left[\tau\right]\!\right]_e (w,w_0) & \stackrel{def}{=} & \Big\{(M_1,M_2) \mid \forall (h_1,h_2,\mathcal{D}) \in (h_1,h_2,\mathcal{D}) : w.h_i \in \mathbf{Cl}(\nu_L(M_i) \cup \mathcal{D}_i) \Rightarrow \\
& & \quad \Big(\exists w' \sqsupseteq w.\exists (h_1',h_2',\mathcal{D}') : w'.\exists \rho_i : (\nu_L(h_i')\backslash\nu_L(h_i)) \hookrightarrow (\mathrm{Loc}\backslash\nu_L(h_i)). \\
& & \qquad \langle M_i,h_i,\mathcal{D}_i\rangle \to^* \rho_i \cdot \langle E_i,h_i',\mathcal{D}_i'\rangle \wedge \\
& & \qquad \Big(E_i = v_i \wedge (v_1,v_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\tau\right]\!\right]_e w' \wedge w' \sqsupseteq^*_{\mathbf{pub}} w_0\Big) \\
& & \qquad \vee\Big(E_i = K_i[f_i\ v_i] \wedge (f_1,f_2) \in e_{\sigma\to\sigma'} \wedge (v_1,v_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_e w' \\
& & \qquad\quad \wedge\forall w'' \sqsupseteq^*_{\mathbf{pub}} w'.(K_1,K_2) \in \mathcal{K}_{\mathcal{A}}\left[\!\left[\sigma',\tau\right]\!\right]_e (w'',w_0)\Big)\Big)\Big\}
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{G}_{\mathcal{A}}\left[\!\left[(x_1,x_2,\tau)\cdot e_P\right]\!\right]_e w & \stackrel{def}{=} & \{((x_1 \mapsto v_1)\cdot\gamma_1, (x_2 \mapsto v_2)\cdot\gamma_2) \mid (v_1,v_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\tau\right]\!\right]_e w \\
& & \qquad \wedge(\gamma_1,\gamma_2) \in \mathcal{G}_{\mathcal{A}}\left[\!\left[e_P\right]\!\right]_e w\}
\end{array}
$$

$$
\begin{array}{l}
\Sigma;\Gamma_g,\Gamma_f \vdash M_1 \simeq_{clog} M_2 : \tau \stackrel{def}{=} \exists S \subseteq \mathrm{Int}, \exists \mathcal{A} \in \mathrm{LTS}_S, \exists s \in S, \forall h \in \mathbf{Cl}(\Sigma, \nu_L(\mathrm{codom}(\gamma_g))), \\
\qquad \forall \gamma_g : \Gamma_g \to \mathrm{Val}.(\gamma_g(M_1),\gamma_g(M_2)) \in \mathcal{E}_{\mathcal{A}}\left[\!\left[\tau\right]\!\right]_{\widetilde{\Gamma_f}} (w,w)
\end{array}
$$

where $w = (s,h,h,\widetilde{\nu_L(h)})$ and Γ_f (resp. Γ_g) contains only variables of functional (resp. ground) types.

Figure 6.2: Definition of Concrete Logical Relations for RefML

PROOF By induction on σ. Let $w_1 \sqsupseteq^{*}_{\mathbf{pub}} w$, we can consider any $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w$ (there is always such w_2). Then:

— if σ is of atomic type, we must prove that for all $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_e w_2$, $(x_1\, v_1, x_2\, v_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e w_2$, which is direct.

— Otherwise, we must prove that $(x_1\, y_1, x_2\, y_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_{e\cdot(y_1,y_2,\sigma)} w_2$. To do so, we just have to prove that $(y_1, y_2) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_{e\cdot(y_1,y_2,\sigma)} w_2$, which comes from the induction hypothesis. ∎

Notice that even if we use a logical relation $\mathcal{K}_{\mathcal{A}} [\![\sigma, \tau]\!]$ on context, our definition do not use any biorthogonal technique.

Then, concrete logical relations allow us to consider only *ground callbacks* of the interactive reduction, that is player questions which appears at depth zero. In fact, they are defined by an induction on types and on the number of ground callbacks of the terms considered. This is well-defined because only divergent-free terms of GroundML are considered here.

The relation $\mathcal{G}_{\mathcal{A}} [\![e_P]\!]_e w$ on substitutions will be particularly useful to relate player environments in trace semantics.

Finally, we define the "full" logical relation $\Sigma; \Gamma_f, \Gamma_g \vdash M_1 \simeq_{clog} M_2 : \tau$ for terms with open ground variables. Notice that we could have used $(\gamma_{g,1}, \gamma_{g,2}) \in \mathcal{G}_{\mathcal{A}} \left[\!\!\left[\widetilde{\Gamma_g}\right]\!\!\right]_{\varepsilon} w$ in this definition rather than a single substitution $\gamma_g : \Gamma_g \to \mathrm{Val}$ since for such world $w = (s, h, h, \widetilde{\nu_L(h)})$, we would have necessarily $\gamma_{g,1} = \gamma_{g,2}$.

6.3 Kripke Trace Semantics

In order to relate our concrete logical relations to trace semantics, we introduce *Kripke trace semantics*, that is a refinement of trace semantics where worlds are used to constrain the heaps used, so that there is a distinction between private and disclosed locations. As we said in the previous chapter, trace semantics does not allow for compositional reasoning because the starting heap in the definition is unconstrained—all the locations are disclosed. This suggests that trace semantics could also benefit from a refinement using worlds.

Worlds are used to parametrize the notion of equality induced by trace semantics on two terms, by constraining the heaps appearing in traces. Moreover, we want to keep track of functions values the terms have disclosed to contexts. This is particularly important when these functional values can modify the heap, like setters.

So we refine the equivalence $\simeq^{\mathcal{D}}_{e_P,e_O}$ on set of traces using worlds. This refined equivalence keeps track in γ_1, γ_2 of the functions the terms have disclosed to contexts.

Definition 41. *Given two terms M_1 and M_2, a world w, two spans e_P, e_O on player and opponent name pointers and two substitutions $\gamma_i : e_{P,i} \to \mathrm{Val}$, we define $[M_1] \simeq^{\gamma,w}_{e_P,e_O}$*

$[M_2]$, *if for all* $(h_1, h_2, \mathcal{D}) : w$ *s.t.* $h_i \in \mathbf{Cl}(\nu_L(M_i, \mathrm{codom}(\gamma_i)) \cup \mathcal{D}_i)$,

$$[\langle M_1, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle] \simeq^{\mathcal{D}}_{e_P, e_O} [\langle M_2, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle]\,.$$

Notice that the definition of $[M_1] \simeq^{\gamma,w}_{e_P,e_O} [M_2]$ does not depend on any LTS $\mathcal{A}$, a fortiori not on the current state of w. Indeed, it just uses the current heaps and spans of w, but not its transition structure induces by $\mathcal{A}$. Then, when working with terms where all locations have been disclosed, one can relate this definition to the usual equivalence on trace.

Theorem 29: *Let* M_1, M_2 *two terms such that* $\Sigma; \Gamma_g, \Gamma_f \vdash M_1, M_2 : \tau$. *Then*

$$[\Sigma; \Gamma_g, \Gamma_f \vdash M_1 : \tau] = [\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$$

iff for all $\gamma_g : \Gamma_g \to \mathrm{Val}$, *for all* $h \in \mathbf{Cl}(\Sigma \cup \nu_L(\mathrm{codom}(\gamma_g)))$, $[\gamma_g(M_1)] \simeq^{\varepsilon,w}_{\varepsilon,\widetilde{\Gamma}} [\gamma_g(M_2)]$ *where* $w = (\_, h, h, \widetilde{\nu_L(h)})$.

Proof Straightforward from Theorem 11. ∎

We now show basic facts about the equivalence between Kripke trace semantics and concrete logical relations. Firstly, we can prove that two ground values logically related in w have their trace semantics equivalent in w.

Theorem 30: *Let* v_1, v_2 *two closed values of ground type* ι, e_O *a span on opponent name pointers and* w *a world. If* $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\iota]\!]_{e_O} w$ *then* $[v_1] \simeq^{\varepsilon,w}_{\varepsilon,\varepsilon} [v_2]$.

Proof Unwinding the definition of $[v_1] \simeq^{\varepsilon,w}_{\varepsilon,e_O} [v_2]$, we have to prove that for all $(h_1, h_2, \mathcal{D}) : w$ s.t. $h_i \in \mathbf{Cl}(\nu_L(v_i))$, $(\langle \bar{v_1}\rangle, h_{1|\mathcal{D}'_1}) \sim^{\mathcal{D}'}_{\cdot,e_O} (\langle \bar{v_1}\rangle, h_{2|\mathcal{D}'_2})$ where $\mathcal{D}' \sqsupseteq \mathcal{D}$ with $\mathcal{D}'_i = \mathrm{discl}(v_i, h_i, \mathcal{D}_i)$.

If $\iota = \mathrm{Int}$, then $\mathcal{D}' = \mathcal{D}$ and to prove the equivalence of actions amounts to prove $v_1 = v_2$ which is direct from $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\mathrm{Int}]\!]_{e_O} w$.

If $\iota = \mathrm{ref}\,\iota'$, then from $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\mathrm{ref}\,\iota']\!]_{e_O} w$, $(v_1, v_2) \in \mathcal{D}_w$, the span on locations of w. And from $(h_1, h_2, \mathcal{D}) : w$ $\mathcal{D} \subseteq \mathcal{D}_w$, and $h_i \in \mathbf{Cl}(\nu_L(v_i))$ gives us that indeed $v_i \in \mathcal{D}$. So $\mathcal{D}'_i = \mathcal{D}_i$, *i.e.* $v_1 \sim_{\mathcal{D}'} v_2$ and the definition of $(h_1, h_2, \mathcal{D}) : w$ gives us that $h_{1|\mathcal{D}_1} \sim_{\mathcal{D}} h_{2|\mathcal{D}_2}$. ∎

Notice that the reciprocal property,

$$\text{if } [v_1] \simeq^{\varepsilon,w}_{\varepsilon,\varepsilon} [v_2] \text{ then } (v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\iota]\!]_{e_O} w$$

is in general false for $\iota = \mathrm{ref}\,\iota'$. Indeed, there is no reason for (v_1, v_2) to be in $\mathcal{D}_w$ in such a case. This is one of the reason to justify the introduction of *adequate worlds* in Section 6.5. Then, under the hypothesis of w is adequate, one can prove the wanted property (cf. Theorem 38).

We now want to relate the equivalence of terms M_1, M_2 to the equivalence of their (elementary interactive) reductions E_1, E_2, which are either values v_1, v_2 or callbacks $K_1[f_1\ v_1], K_2[f_2\ v_2]$. To do so, we first relate the semantics of M_i to E_i.

Lemma 28: *Let M a term, such that $\langle M, h, D\rangle \to^* \langle E, h', D'\rangle$, then $[\langle M, \gamma, \mathcal{I}, h, D\rangle] = [\langle E, \gamma, \mathcal{I}, h', D'\rangle]$*

Proof E is either equal to v or $K[f\ v]$. The proof is done by case analysis on τ, the type of v.

— If τ is of ground type, then $[\langle M, \gamma, \mathcal{I}, h, D\rangle]$ is equal to $(a, h'_{|D'}) \cdot [\langle \Diamond, \gamma, \mathcal{I}, h', D'\rangle]$, where a is either $\langle \bar{v}\rangle$ or $\bar{f}\,\langle v\rangle$. But $[\langle E, \gamma, \mathcal{I}, h', D'\rangle]$ is also equal to $(a, h'_{|D'}) \cdot [\langle \Diamond, \gamma, \mathcal{I}, h', D'\rangle]$ since the disclosure of locations has already happened.

— Otherwise, we have $[\langle M, \gamma, \mathcal{I}, h, D\rangle]$ equal to

$$\bigcup_a \{(a, h'_{|D'}) \cdot [\langle \Diamond, \gamma \cdot (x \hookrightarrow v), \mathcal{I}, h', D'\rangle]\}$$

where a ranges either over $\langle \bar{x}\rangle$ with $x \in \mathbb{P}\backslash\mathrm{dom}(\gamma)$ if $E = v$, or over $\bar{f}\,\langle x\rangle$ with $x \in \mathbb{P}\backslash\mathrm{dom}(\gamma)$ if $E = K[f\ v]$. And again $[\langle v, \gamma, \mathcal{I}, h', D'\rangle]$ is also equal to

$$\bigcup_a \{(a, h'_{|D'}) \cdot [\langle \Diamond, \gamma(x \hookrightarrow v), \mathcal{I}, h', D'\rangle]\}.$$

■

Then, we state two lemmas to reason on equivalence of trace semantics. The first one is about renaming of locations.

Lemma 29 (Renaming of locations): *Let $e_P, e_O, \mathcal{D}$ three spans respectively on player and opponent name pointers and on locations, and γ_1, γ_2 two value substitutions defined respectively on $e_{P,1}$ and $e_{P,2}$. Suppose $\left[\langle E_1 \cdot \overrightarrow{K_1}, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle\right] \simeq^{\mathcal{D}}_{e_P, e_O} \left[\langle E_2 \cdot \overrightarrow{K_2}, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle\right]$ and let two injections $\rho_i : \mathrm{dom}(h_i) \hookrightarrow \mathrm{Loc}$, then, writing $\mathcal{D}'$ for $(\rho_1 \times \rho_2) \cdot \mathcal{D}$, we have $\left[\rho_1 \cdot \langle E_1 \cdot \overrightarrow{K_1}, \gamma_1, \mathcal{I}_1, h'_1, \mathcal{D}_1\rangle\right] \simeq^{\mathcal{D}'}_{e_P, e_O} \left[\rho_2 \cdot \langle E_2 \cdot \overrightarrow{K_2}, \gamma_2, \mathcal{I}_2, h'_2, \mathcal{D}_2\rangle\right]$*

Proof Straightforward using Lemma 31. ■

The second one is about garbage-collecting unused location. It is particularly useful because the definition of $[M_1] \simeq^{\gamma, w}_{e_P, e_O} [M_2]$ works only with heaps in $\mathbf{Cl}(\nu_L(M_i, \gamma_i))$ while the usual equivalence on trace semantics is defined on any heaps.

Lemma 30 (Garbage Collect): *Let $e_P, e_O, \mathcal{D}$ three spans respectively on player and opponent name pointers and on locations, and γ_1, γ_2 two value substitutions defined respectively on $e_{P,1}$ and $e_{P,2}$. Suppose h_1, h_2 are two heaps such that we can decompose h_i into $h'_i \cdot h^g_i$ with $h'_i \in \mathbf{Cl}(\nu_L(E_i, \overrightarrow{K_1}, \mathrm{codom}(\gamma_i)) \cup \mathcal{D}_i)$, Then*

$$\left[\langle E_1 \cdot \overrightarrow{K_1}, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle\right] \simeq^{\mathcal{D}}_{e_P,e_O} \left[\langle E_2 \cdot \overrightarrow{K_2}, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle\right]$$

if and only if

$$\left[\langle E_1 \cdot \overrightarrow{K_1} \cdot \overrightarrow{K_1}, \gamma_1, \mathcal{I}_1, h'_1, \mathcal{D}_1\rangle\right] \simeq^{\mathcal{D}}_{e_P,e_O} \left[\langle E_2 \cdot \overrightarrow{K_2}, \gamma_2, \mathcal{I}_2, h'_2, \mathcal{D}_2\rangle\right].$$

Proof Straightforward from the fact that the locations from h^g_i are never disclosed, so cannot appear in traces. ∎

Now, we prove the wanted theorem

Theorem 31: *Let (M_1, M_2) a pair of terms, e_O a span on their name pointers and w a world. Let also consider e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions on its projections. Suppose that for all $(h_1, h_2, \mathcal{D}) : w$ s.t. $h_i \in \mathbf{Cl}(\nu_L(M_i, \gamma_i) \cup \mathcal{D}_i)$, there exists E_1, E_2, a world $w' \sqsupseteq w$, two heaps and a span $(h'_1, h'_2, \mathcal{D}') : w'$ and two injections $\rho_i : \mathrm{dom}(h'_i) \backslash \mathrm{dom}(h_i) \hookrightarrow \mathrm{Loc}$ such that*

- *$\langle M_i, h_i, \mathcal{D}_i\rangle \to^* \rho_i \cdot \langle E_i, h'_i, \mathcal{D}'_i\rangle$,*
- *either both $E_i = v_i$ and $[v_1] \simeq^{\gamma,w'}_{e_P,e_O} [v_2]$,*
- *or both $E_i = K_i[f_i\ v_i]$ and $(f_1, f_2) \in e_O$, $[v_1] \simeq^{\gamma',w'}_{e'_P,e_O} [v_2]$ and $[K_1] \simeq^{\gamma',w'}_{e'_P,e_O} [K_2]$, where $\gamma'_i = \gamma$ and $e'_P = e_P$ if σ, the type of v_i, is atomic, otherwise $\gamma'_i = \gamma_i \cdot (x_i \hookrightarrow v_i)$ and $e'_P = e_P \cdot (x_1, x_2, \sigma)$ for x_1, x_2 fresh in e_P.*

Then $[M_1] \simeq^{\gamma,w}_{e_P,e_O} [M_2]$.

Proof Suppose both $E_i = v_i$, from lemma 28, we get that

$$[\langle M_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle] = [\rho_i \cdot \langle E_i, \gamma_i, e_{O,i}, h'_i, \mathcal{D}'_i\rangle].$$

Let us decompose h'_i into $h''_i \cdot h^g_i$ with $h''_i \in \mathbf{Cl}(\nu_L(E_i, \mathrm{codom}(\gamma_i)) \cup \mathcal{D}_i)$ Then, from $[v_1] \simeq^{\gamma,w'}_{e_P,e_O} [v_2]$ we get that

$$[\langle v_1, \gamma_1, e_{O,1}, h'_1, \mathcal{D}'_1\rangle] \simeq^{\mathcal{D}'}_{e_P,e_O} [\langle v_2, \gamma_2, e_{O,2}, h'_2, \mathcal{D}'_2\rangle]$$

and we conclude using lemmas on garbage collections (lemma 30) and on renaming of locations (lemma 29).

Otherwise, both $E_i = K_i[f_i\ v_i]$ and we reason by case analysis of σ.

— If σ is of ground type, then $[\langle K_i[f_i\ v_i], \gamma_i, e_{O,i}, h'_i, \mathcal{D}'_i\rangle]$ is equal to

$$(\rho_i \cdot (\bar{f}_i \langle v_i \rangle, h'_{i|\mathcal{D}'_i})) \cdot [\rho_i \cdot \langle K_i[\bullet], \gamma_i, e_{O,i}, h'_i, \mathcal{D}'_i\rangle]$$

and from $[v_1] \simeq^{\gamma,w}_{e_P,e_O} [v_2]$ and $(f_1, f_2) \in e_O$ we get that $(\rho_1 \cdot (\bar{f}_1 \langle v_1 \rangle, h'_{1|\mathcal{D}'_1})) \sim^{\mathcal{D}''}_{e_P,e_O} (\rho_1 \cdot (\bar{f}_2 \langle v_1 \rangle, h'_{2|\mathcal{D}'_2}))$ where $\mathcal{D}'' = (\rho_1 \times \rho_2)(\mathcal{D}')$. Then we conclude using $[K_1] \simeq^{\gamma,w'}_{e_P,e_O} [K_2]$ with both lemma 30 and 29.

— Otherwise, $[\langle M_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle]$ is equal to

$$\bigcup_{x_i} (\bar{f}_i \langle x_i \rangle, \rho_i \cdot h'_{i|\mathcal{D}'_i} \cdot [\rho_i \cdot \langle K_i[\bullet], \gamma'_i, e_{O,i}, h'_i, \mathcal{D}'_i\rangle]$$

with x_i ranges over $\mathbb{P}\backslash\mathrm{dom}(\gamma_i)$. Then writing $\mathcal{D}''$ for $(\rho_1 \times \rho_2)(\mathcal{D}')$, we have $(\bar{f}_1 \langle x_1 \rangle, \rho_i \cdot h'_{1|\mathcal{D}'_1}) \sim^{\mathcal{D}''}_{e'_P,e_O} (\bar{f}_2 \langle x_1 \rangle, \rho_2(h'_{2|\mathcal{D}'_2}))$. Then we conclude again using $[K_1] \simeq^{\gamma,w'}_{e'_P,e_O} [K_2]$ with both lemma 30 and 29. ∎

Finally, we state two lemmas on trace semantics.

Lemma 31: *Let C, C' two configurations and U a trace s.t. $C \xrightarrow{U} C'$ and $\rho : X \hookrightarrow \mathrm{Loc}$ with $X \subseteq \nu_L(C)$ an injection. Then there exists ρ' which extends ρ s.t. $\rho \cdot C \xrightarrow{\rho' \cdot U} \rho' \cdot C'$*

Lemma 32 (Disclosed Equivalence): *Let $e_P, e_O, \mathcal{D}$ three spans respectively on player and opponent name pointers and on locations, and γ_1, γ_2 two value substitutions defined respectively on $e_{P,1}$ and $e_{P,2}$. Suppose*

$$\left[\langle K_1[\bullet] \cdot \overrightarrow{K_1}, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle\right] \simeq^{\mathcal{D}}_{e_P,e_O} \left[\langle K_2[\bullet] \cdot \overrightarrow{K_2}, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle\right]$$

and let h'_1, h'_2 such that $h'_{i|\overline{\mathcal{D}_i}} = h_{i|\overline{\mathcal{D}_i}}$ and $h'_{1|D_1} \sim^{\mathcal{D}}_{e_P,e_O} h'_{2|D_2}$. Then

$$\left[\langle K_1[\bullet] \cdot \overrightarrow{K_1}, \gamma_1, \mathcal{I}_1, h'_1, \mathcal{D}_1\rangle\right] \simeq^{\mathcal{D}}_{e_P,e_O} \left[\langle K_2[\bullet] \cdot \overrightarrow{K_2}, \gamma_2, \mathcal{I}_2, h'_2, \mathcal{D}_2\rangle\right].$$

Proof Straightforward from the fact that the interactive reduction on opponent configurations choose arbitrary values on the disclosed part of the heap. ∎

6.4 From Concrete Logical Relations to Trace Semantics

Soundness of logical relations *w.r.t.* contextual equivalence is usually proved via the so-called *fundamental property*, which states that from for a term M s.t. $\Sigma; \Gamma \vdash M : \tau$, we have $\Sigma; \Gamma \vdash M \simeq_{clog} M : \tau$ (*i.e.* $(M, M) \in \mathcal{E}[\![\tau]\!]$). This property is proved via an induction on the typing judgment of M, using *compatibility lemmas* which show that

the logical relations follow the structure of the typing proof. However, such approach cannot work here, because of the existential quantification over LTS in the definition of $\Sigma;\Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$. Indeed, performing the induction on the typing proof, the application rule $M_1M_2 : \tau$ is problematic, since we get two LTS $\mathcal{A}_1, \mathcal{A}_2$ which are a priori not related.

To get around this problem, we rather link the logical relation to Kripke trace semantics, and use its soundness to conclude. So the goal of this section is to prove the soundness of our concrete logical relations *w.r.t.* Kripke trace semantics, that is:

$$\text{if } (M_1, M_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_{e_O}(w, w_0) \text{ then } [M_1] \simeq_{e_P,e_O}^{(\gamma_1,\gamma_2),w} [M_2].$$

for a span e_P on player name pointers and γ_1, γ_2 logically related (see Theorem 34 for the exact statement). In the previous section, we have seen that Kripke trace semantics follows the same reduction properties than the definition of $\mathcal{E}[\![\tau]\!]$. But this is the "easy" part, since we have only consider "internal" reduction without any actions. It is the soundness for values and contexts which are the hardest one to prove, since they induce actions in trace semantics.

To prove them, we perform some "surgery" on traces to show that Kripke trace semantics follows the inductive structure of logical relations. It is similar to the proof done in Section 5.5, but complicated by the presence of heaps.

Firstly, we prove that the disclosure process performed by opponent actions, namely the "creation" of disclosed locations, corresponds to the notion of future world $w_1 \sqsupseteq_{\mathbf{pub}}^{\mathcal{F}\tau} w$.

Lemma 33: *Let e_P, e_O two spans on name pointers, w a world, γ_i two environments on $e_{P,i}$ and K_1, K_2 two applicative contexts. Taking $(h_1, h_2, \mathcal{D}) : w$ with $h_i \in \mathbf{Cl}(\nu_L(K_i, \gamma_i))$, suppose that*

$$\langle K_i[\bullet], \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle \xrightarrow{\langle u_i\rangle, h'_{i|D'_i}} \langle K_i[u_i], \gamma_i, e'_{O,i}, h'_i, D'_i\rangle$$

s.t. there exists a span $\mathcal{D}'$ extending $\mathcal{D}$ with $\mathcal{D}'_i = D_i$ and $h'_1 \sim_{\mathcal{D}'} h'_1$, $u_1 \sim_{\mathcal{D}'} u_2$ if u_1, u_2 are locations. then there exists $w_1 \sqsupseteq_{\mathbf{pub}}^{\mathcal{F}\tau} w$, where τ is the type of u_i, s.t. $(h'_1, h'_2, \mathcal{D}') : w_1$.

Proof From the rules **O-AnsG** and **O-Ans**, simplified due to the absence of higher-order references in h_1, h_2, we get that $h'_{i|\overline{\mathcal{D}'_i}} = h_{i|\overline{\mathcal{D}_i}}$ and $\mathcal{D}'_i = \text{discl}(u_i, h'_i, \mathcal{D}_i)$. Let consider the subheap h'^n_i of h'_i whose domain is disjoint from h_i. Then, from $h'_{i|\overline{\mathcal{D}'_i}} = h_{i|\overline{\mathcal{D}_i}}$ we get that $\text{dom}(h'^n_i) = \mathcal{D}'_i \backslash \mathcal{D}_i$. So let $(l_1, l_2, \iota) \in \mathcal{D}' \backslash \mathcal{D}$, from $\mathcal{D}'_i = \text{discl}(u_i, h'_i, \mathcal{D}_i)$ we get that there exists an $n > 0$ such that $l_i \in h^n(\{u_i\} \cup \mathcal{D}_i)$. Suppose this n is maximal, *i.e.* either $h'_i(l_i) \in \{u_i\} \cup \mathcal{D}_i$ or $h'_i(l_i)$ is equal to $()$, a boolean or an integer. This means that there exists $l^0_i \in \{u_i\} \cup \mathcal{D}_i$ and $l^{k+1}_i = h^n(l^k_i)$ for all $k \in \{1, \ldots, n-1\}$, with $l^n_i = l_i$. Then, l^0_i is of type $\iota' = \underbrace{\text{ref} \ldots \text{ref}}_{n \text{ times}} \iota$ (with ι the type of l_i). But then, ι is in $\mathbf{subtype}(\iota')$, so there exists $\mathcal{D}^1 \in \mathbf{NewLoc}(\iota')$ disjoint from $\mathcal{D}$ with $(l^k_1, l^k_2) \in \mathcal{D}^1$ for all $k \in \{1, \ldots, n\}$. Iterating the process over $\mathcal{D}'_i \backslash (\mathcal{D}_i \cup \{(l^k_1, l^k_2) \mid k \in \{1, \ldots, n\}\})$, we get the existence of such disjoints spans $\mathcal{D}^1, \ldots, \mathcal{D}^m$. Thus, one can write $(\mathcal{D}'_i \backslash \mathcal{D}_i)$ as a subset of $\biguplus_{j \in \{1,\ldots,m\}} \mathcal{D}^j$.

Finally, from $(h_1, h_2, \mathcal{D}) : w$, we get that $w = (h_1^w, h_2^w, \mathcal{D}^w)$ with $h_{i|\overline{\mathcal{D}_i}} \subseteq h_i^w$ and $\mathcal{D} = \mathcal{D}^w$. So one can define w_1 as $(h_1^w, h_2^w, \mathcal{D}^w \uplus (\biguplus_{j\in\{1,\dots,m\}} \mathcal{D}^j))$. ∎

Lemma 34: *Let e_P, e_O two spans on name pointers, w a world, γ_i two environments on $e_{P,i}$ and K_i two applicative contexts. Taking $(h_1, h_2, \mathcal{D}) : w$ with $h_i \in \mathbf{Cl}(\nu_L(K_i, \gamma_i))$, suppose that*

$$\langle \Diamond, \gamma_i, e_{O,i}, h_i, D_i\rangle \xrightarrow{f_i\langle u_i\rangle, h'_{i|D'_i}} \langle v_i\ u_i, \gamma_1, e'_{O,1}, h'_1, D'_1\rangle$$

where $\gamma_i(f_i) = v_i$ of type $\tau \to \sigma$ s.t. there exists a span $\mathcal{D}'$ extending $\mathcal{D}$ with $\mathcal{D}'_i = D_i$ and $h'_1 \sim_{\mathcal{D}'} h'_1$, $u_1 \sim_{\mathcal{D}'} u_2$ if u_1, u_2 are locations. Then there exists $w_1 \sqsupseteq^{\mathcal{F}\tau}_{\mathbf{pub}} w$, s.t. $(h'_1, h'_2, \mathcal{D}') : w_1$.

Proof The proof is similar to the previous lemma. ∎

Then, from an opponent action a_1,we show how to build an equivalent opponent action a_2, in the two following lemmas.

Lemma 35: *Let $e_0, e_O, \mathcal{D}$ three spans respectively on player, opponent name pointers and locations. Let γ_1, γ_2 two functional substitutions s.t. $\mathrm{dom}(\gamma_i) = e_{P,i}$. Let $f_1 \in e_{P,1}$ with $\gamma(f_1) = v_1$, s.t.*

$$\langle \Diamond, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle \xrightarrow{f_1\langle u_1\rangle, h^1_{1|\mathcal{D}^1_1}} \langle v_1\ u_1, \gamma_1, e^1_{O,1}, h^1_1, \mathcal{D}^1_1\rangle$$

Then there exists:

— *a span on opponent name pointers $e^1_O \sqsupseteq e_O$,*

— *a span on locations $\mathcal{D}^1 \sqsupseteq \mathcal{D}$,*

— *an action $(f_2\,\langle u_2\rangle, h^1_{2|\mathcal{D}^1_2})$,*

s.t. $(f_1\,\langle u_1\rangle, h^1_{1|\mathcal{D}^1_1}) \sim^{\mathcal{D}^1}_{e_P, e^1_O} (f_2\,\langle u_2\rangle, h^1_{2|\mathcal{D}^1_2})$ and

$$\langle \Diamond, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle \xrightarrow{f_2\langle u_2\rangle, h^1_{2|\mathcal{D}^1_2}} \langle v_2\ u_2, \gamma_1, e^1_{O,2}, h^1_2, \mathcal{D}^1_2\rangle.$$

Proof Firstly, there exists σ, τ and f_2 s.t. $(f_1, f_2) \in e_{O,\sigma\to\tau}$. Then, we reason by case analysis on σ to build u_2.

— If σ is equal to Unit, Bool or Int, we simply take $u_2 = u_1$.

— If $\sigma = \mathrm{ref}\,\iota$, then if $u_1 \in \mathcal{D}_1$, we take as u_2 the corresponding location u_2 s.t. $(u_1, u_2, \iota) \in \mathcal{D}$. Otherwise, we take as l_2 any locations not in h_2.

— If σ is of functional type, then we take as u_2 any fresh opponent name pointer not in $e_{O,2}$. ∎

Lemma 36: *Let $e_0, e_O, \mathcal{D}$ three spans respectively on player, opponent name pointers and locations. Let γ_1, γ_2 two functional substitutions s.t. $\mathrm{dom}(\gamma_i) = e_{P,i}$. Suppose that*

$$\langle K_1[\bullet], \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle \xrightarrow{\langle u_1\rangle, h^1_{1|\mathcal{D}^1_1}} \langle K_1[u_1], \gamma_1, e^1_{O,1}, h^1_1, \mathcal{D}^1_1\rangle$$

Then there exists:

— *a span on opponent name pointers* $e_O^1 \sqsupseteq e_O$,
— *a span on locations* $\mathcal{D}^1 \sqsupseteq \mathcal{D}$,
— *an action* $(\langle x_2 \rangle\, u_2, h^1_{2|\mathcal{D}_2^1})$,

s.t. $(\langle u_1 \rangle, h^1_{1|\mathcal{D}_1^1}) \sim^{\mathcal{D}^1}_{e_P, e_O^1} (\langle u_2 \rangle, h^1_{2|\mathcal{D}_2^1})$ *and*

$$\langle K_2[\bullet], \gamma_2, e_{O,2}, h_2, \mathcal{D}_2 \rangle \xrightarrow{\langle u_2 \rangle, h^1_{2|\mathcal{D}_2^1}} \langle K_2[u_2], \gamma_1, e^1_{O,2}, h^1_2, \mathcal{D}^1_2 \rangle.$$

Proof It follows the same pattern of the proof of the previous lemma. ∎

Then we state a technical unwinding lemma, proving that for two terms and two heaps logically-related at w, there exists a future world w' s.t. they interactively reduce to two values related in w' and two heaps validating w'. This result is non-trivial since the interactive reduction can perform nested calls of terms, which is not directly taken into account with logical relations.

Lemma 37: *Let* M_1, M_2 *two ground-closed terms and* e_O *a span on its (opponent) name pointers. Suppose that* $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_{e_O} (w, w_0)$*, and let* e_P *a span on player name pointers with* $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$ *and* $(h_1, h_2, \mathcal{D}) : w$ *s.t.* $h_i \in \mathbf{Cl}(\nu_L(M_i, \gamma_i) \cup \mathcal{D}_i)$*. Consider* $T_i \in [\langle M_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle]$ *such that* $T_1 \simeq^{\mathcal{D}}_{e_P, e_O} T_2$ *with* $T_i = U_i \cdot (\langle \bar{v_i} \rangle, h^f_{i|\mathcal{D}_i^f})$ *and*

$$\begin{array}{lll} \langle M_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle & \xrightarrow{U_i} & \langle v_i, \gamma'_i, e^f_{O,i}, h'_i, \mathcal{D}'_i \rangle \\ & \xrightarrow{\langle \bar{v_i} \rangle, h^f_{i|\mathcal{D}_i^f}} & \langle \Diamond, \gamma^f_i, e^f_{O,i}, h^f_i, \mathcal{D}^f_i \rangle \end{array}$$

with $\gamma_i^f = \gamma'_i \cdot (x_i \hookrightarrow v_i)$ *if* v_i *is of functional type, otherwise* $\gamma_i^f = \gamma'_i$*. Then there exists*
— *two injections* $\rho_i^f : (\nu_L(h_i^f) \setminus \nu_L(h_i)) \hookrightarrow (\mathrm{Loc} \setminus \nu_L(h_i))$,
— *a world* w_f *s.t.* $w_f \sqsupseteq w$ *and* $w_f \sqsupseteq_{\mathbf{pub}} w_0$,
— *a span on player name pointers* e_P^f *which extends* e_P *and whose projection are equal to* $\mathrm{dom}(\gamma_i^f)$,
— *a span on opponent name pointers* e_O^f *which extends* e_O *and whose projection are equal to* $e^f_{O,i}$,

such that $(\rho_1^f \cdot h_1^f, \rho_2^f \cdot h_2^f, \rho_1^f \times \rho_2^f \cdot \mathcal{D}^f) : w_f$, $(\rho_1^f \cdot v_1, \rho_2^f \cdot v_2) \in \mathcal{V}_{\mathcal{A}} [\![\tau]\!]_{e_O^f} w_f$ *and* $(\rho_1^f \cdot \gamma_1^f, \rho_2^f \cdot \gamma_2^f) \in \mathcal{G}_{\mathcal{A}} \left[\!\!\left[e_P^f \right]\!\!\right]_{e_O^f} w_f$.

Proof From $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_{e_O} (w, w_0)$, we get the existence of
— two terms E_i,
— a world $w_1 \sqsupseteq w$,
— two heaps and a span $(h_1^1, h_2^1, \mathcal{D}^1) : w_1$,
— two injections $\rho_i : (\nu_L(h_i^1) \setminus \nu_L(h_i)) \hookrightarrow (\mathrm{Loc} \setminus \nu_L(h_i))$,

such that $\langle M_i, h_i, \mathcal{D}_i \rangle \to^* \langle E_i, \rho_i \cdot h_i^1, \rho_i \cdot \mathcal{D}_i^1 \rangle$.

Either both $E_i = v_i$, i.e. $U_i = \varepsilon$, $\rho_i \cdot h_i^1 = h_i^f$, $e_{O,i}^f = e_{O,i}$. Then $w_1 \sqsupseteq_{\mathbf{pub}} w_0$ and $(\rho_1^{-1} \cdot v_1, \rho_2^{-1} \cdot v_2) \in \mathcal{V}_{\mathcal{A}} [\![\tau]\!]_e w_1$. So we directly get the wanted properties taking w_f as w_1, ρ_i^f as ρ_i^{-1}, $e_P^f = e_P$ and $e_O^f = e_O$.

Otherwise $E_i = K_i[f_i\, u_i^1]$ s.t.
- $(f_1, f_2) \in e_{O,\sigma\to\sigma'}$ for two types σ, σ',
- $(\rho_1^{-1} \cdot u_1^1, \rho_2^{-1} \cdot u_2^1) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_{e_O} w_1$,
- for all $w_2 \sqsupseteq^*_{\mathbf{pub}} w_1$, $(\rho_1^{-1} \cdot K_1, \rho_2^{-1} \cdot K_2) \in \mathcal{K}_{\mathcal{A}} [\![\sigma', \tau]\!]_{e_O} (w_2, w_0)$.

Then, we can decompose U_i into $(\bar{f}_i \left\langle \widehat{u_i^1} \right\rangle, \rho_i^1 \cdot h^1_{i|\mathcal{D}_i^1}) \cdot U_i^1 \cdot U_i^2$ with

$$\langle K_i[f_i\, u_i^1], \gamma_i, e_{O,i}, \rho_i^1 \cdot h_i^1, \rho_i^1 \cdot \mathcal{D}_i^1 \rangle \xrightarrow[U_i^1 \longrightarrow]{\bar{f}_i \left\langle \widehat{u_i^1} \right\rangle, \rho_i^1 \cdot (h^1_{i|\mathcal{D}_i^1})} \begin{array}{l} \langle K_i[\bullet], \gamma_i^1, e_{O,i}, \rho_i^1 \cdot h_i^1, \rho_i^1 \cdot \mathcal{D}_i^1 \rangle \\ \langle K_i[\bullet], \gamma_i^2, e_{O,i}^1, h_i^2, \mathcal{D}_i^2 \rangle \end{array}$$

because $\mathrm{discl}(u_i^1, h_i^1, \mathcal{D}_i^1) = D_i^1$ since $u_i^1 \in D_i^1$ if u_i^1 is a location. Then, using lemma 31, we get the existence of ρ_i^1 extending ρ_i^{-1} such that

$$\langle \rho_i^{-1} \cdot K_i[f_i\, u_i^1], \gamma_i, e_{O,i}, h^1_{i|\mathcal{D}_i^1}, \rangle \xrightarrow[\rho_i^1 \cdot U_i^1]{\bar{f}_i \left\langle \rho_i^{-1} \cdot \widehat{u_i^1} \right\rangle, \widetilde{h^1_{i|\mathcal{D}_i^1}}} \begin{array}{l} \langle \rho_i^{-1} \cdot K_i[\bullet], \rho_i^{-1} \cdot \gamma_i^1, e_{O,i}, h_i^1, \mathcal{D}_i^1 \rangle \\ \langle \rho_i^1 \cdot K_i[\bullet], \rho_i^1 \cdot \gamma_i^2, e_{O,i}^1, \rho_i^1 \cdot h_i^2, \rho_i^1 \cdot \mathcal{D}_i^2 \rangle \end{array}$$

If σ is atomic, then $\widehat{u_i^1} = u_i^1$ and $\rho_i^{-1} \cdot \gamma_i^1 = \gamma_i$. Otherwise, $\widehat{u_i^1}$ is equal to some fresh player name pointer y_i and $\rho_i^{-1} \cdot \gamma_i^1 = \gamma_i \cdot (y_i \hookrightarrow \rho_i^{-1} \cdot u_i^1)$, and defining $e_P^1 = e_P \cdot (y_1, y_2)$, we have $(\rho_1^{-1} \cdot \gamma_1^1, \rho_2^{-1} \cdot \gamma_2^1) \in \mathcal{G}_{\mathcal{A}} [\![e_P^1]\!]_{e_O} w_1$ using world monotonicity of functional values. In both cases, we have $(\rho_i^1 \cdot U_i^1) \in [\langle \Diamond, \rho_i^1 \cdot \gamma_i^1, e_{O,i}, h_i^1, \mathcal{D}_i^1 \rangle]$ and $(\rho_i^1 \cdot U_1^1) \simeq^{\mathcal{D}^1}_{e_P^1, e_O} (\rho_i^1 \cdot U_2^1)$. So using the induction hypothesis of the second lemma on the pair of traces $(\rho_i^1 \cdot U_1^1, \rho_i^1 \cdot U_2^1)$ we get the existence of
- two injections $\rho_i^2 : (\nu_L(\rho_i^{-1} \cdot h_i^2) \setminus \nu_L(h_i^1)) \hookrightarrow (\mathrm{Loc} \setminus \nu_L(h_i^1))$,
- a world $w_2 \sqsupseteq^*_{\mathbf{pub}} w_1$ such that $(\rho_1^2 \cdot h_1^2, \rho_2^2 \cdot h_2^2, (\rho_1^2 \times \rho_2^2) \cdot \mathcal{D}^2) : w_2$
- a span on player name pointers e_P^2 which extends e_P^1 and whose projection are equal to $\mathrm{dom}(\gamma_i^2)$,
- a span on opponent name pointers e_O^1 which extends e_O and whose projection are equal to $e_{O,i}^1$,

such that, defining $\widetilde{\rho_i}$ as the injection $\rho_i^2 \circ \rho_i^1 : (\nu_L(h_i^2) \setminus \nu_L(h_i)) \hookrightarrow (\mathrm{Loc} \setminus \nu_L(h_i))$, we have $(\widetilde{\rho_1} \cdot \gamma_1^2, \widetilde{\rho_2} \cdot \gamma_2^2) \in \mathcal{G}_{\mathcal{A}} [\![e_P^2]\!]_{e_O^1} w_2$. Moreover, $(\widetilde{\rho_1} \cdot h_1^2, \widetilde{\rho_2} \cdot h_2^2, (\widetilde{\rho_1} \times \widetilde{\rho_2}) \cdot \mathcal{D}^2) : w_2$.

Then, we can decompose $\widetilde{\rho_i} \cdot U_i^2$ into $(\langle u_i^2 \rangle, h^3_{i|\mathcal{D}_i^3}) \cdot U_i^3$ such that

$$\langle \widetilde{\rho_i} \cdot K_i[\bullet], \widetilde{\rho_i} \cdot \gamma_i^2, e_{O,i}^1, \widetilde{\rho_i} \cdot h_i^2, \widetilde{\rho_i} \cdot \mathcal{D}_i^2 \rangle \xrightarrow{\langle u_i^2 \rangle, h^3_{i|\mathcal{D}_i^3}} \langle (\widetilde{\rho_i} \cdot K_i)[u_i^2], \widetilde{\rho_i} \cdot \gamma_i^2, e_{O,i}^2, h_i^3, \mathcal{D}_i^3 \rangle$$

And from the fact that $\widetilde{\rho_1} \cdot U_1^2 \simeq^{\mathcal{D}'_3}_{e_P^2, e_O^1} \widetilde{\rho_2} \cdot U_2^2$, we get the existence of $e_O^2 \sqsupseteq e_O^1$ and $\mathcal{D}^3 \sqsupseteq \mathcal{D}^2$ such that $(\langle u_1^2 \rangle, h^3_{1|\mathcal{D}_1^3}) \sim^{\mathcal{D}^3}_{e_P^2, e_O^2} (\langle u_2^2 \rangle, h^3_{2|\mathcal{D}_2^3})$. So using lemma 33, we get the existence of

$w_3 \sqsupseteq_{\mathbf{pub}}^{\mathcal{F}\sigma'} w_2$ such that $(\widetilde{\rho_1} \cdot h_1^3, \widetilde{\rho_2} \cdot h_2^3, (\widetilde{\rho_1} \times \widetilde{\rho_2}) \cdot \mathcal{D}^3) : w_3$.

Then, $\widetilde{\rho_i} \cdot K_i = \rho_i^{-1} \cdot K_i$ and from the world motononicity of contexts, $(\widetilde{\rho_1} \cdot K_1, \widetilde{\rho_2} \cdot K_2) \in \mathcal{K}_{\mathcal{A}} [\![\sigma', \tau]\!]_{e_O^1} (w_3, w_0)$. And we can conclude using the induction hypothesis on the first lemma. ∎

Lemma 38: *Let e_P, e_O two spans respectively on player and opponent name pointers, $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$ and $(h_1, h_2, \mathcal{D}) : w$ s.t. $h_i \in \mathbf{Cl}(\nu_L(M_i, \gamma_i) \cup \mathcal{D}_i)$. Then for all $T_i \in [\langle \lozenge, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle]$ such that $T_1 \simeq^{\mathcal{D}}_{e_P, e_O} T_2$ and $\langle \lozenge, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle \xrightarrow{T_i} \langle \lozenge, \gamma_i^f, e_{O,i}^f, h_i^f, \mathcal{D}_i^f \rangle$ there exists*

- *two injections $\rho_i : (\nu_L(h_i') \backslash \nu_L(h_i)) \hookrightarrow \mathrm{Loc}$,*
- *a world $w_f \sqsupseteq^*_{\mathbf{pub}} w$ such that $(\rho_1(h_1^f), \rho_2(h_2^f), (\rho_1 \times \rho_2)(\mathcal{D}^f)) : w_f$,*
- *a span e_O^f which extends e_O and whose projections are $e_{O,i}^f$,*
- *and a span e_P^f which extends e_P such that $(\rho_1(\gamma_1^f), \rho_2(\gamma_2^f)) \in \mathcal{G}_{\mathcal{A}} \left[\!\!\left[e_P^f \right]\!\!\right]_{e_O^f} w_f$,*

Proof If T_i is non-empty, then we have $T_i = (f_i \langle u_i \rangle, h^1_{i|\mathcal{D}_i^1}) \cdot U_i^1 \cdot U_i^2$ and

$$\begin{array}{lll} \langle \lozenge, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle & \xrightarrow{f_i\langle u_i \rangle, h^1_{i|\mathcal{D}_i^1}} & \langle v_i\ u_i, \gamma_i, e_{O,i}^1, h_i^1, \mathcal{D}_i^1 \rangle \\ & \xrightarrow{U_i^1} & \langle \lozenge, \gamma_i^2, e_{O,i}^2, h_i^2, \mathcal{D}_i^2 \rangle \\ & \xrightarrow{U_i^2} & \langle \lozenge, \gamma_i^f, e_{O,i}^f, h_i^f, \mathcal{D}_i^f \rangle \end{array}$$

where $\gamma_i(f_i) = v_i$, with $U_i^1 \in \left[\langle v_i\ u_i, \gamma_i^1, e_{O,i}^1, h_i^1, \mathcal{D}_i^1 \rangle\right]$ and $U_i^2 \in \left[\langle \lozenge, \gamma_i^2, e_{O,i}^2, h_i^2, \mathcal{D}_i^2 \rangle\right]$. Then, from the equivalence of T_1 and T_2, $(f_1, f_2) \in e_{P,\sigma\to\tau}$, so $(\gamma_1(f_1), \gamma_2(f_2)) \in \mathcal{V}_{\mathcal{A}} [\![\sigma \to \tau]\!]_{e_O} w$.

Using lemma 33, we get the existence of $w_1 \sqsupseteq_{\mathbf{pub}}^{\mathcal{F}\sigma} w$ such that $(h_1^1, h_2^1, \mathcal{D}^1) : w_1$, and moreover $(v_1\ u_1, v_2\ u_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_{e_O^1} (w_1, w_1)$. So using the induction hypothesis on the first lemma, we get the existence of

- two injections $\rho_i^2 : (\nu_L(h_i^2) \backslash \nu_L(h_i^1)) \hookrightarrow \mathrm{Loc}$,
- a world $w_2 \sqsupseteq_{\mathbf{pub}} w_1$ such that $(\rho_1^2(h_1^2), \rho_2^2(h_2^2), \widetilde{\mathcal{D}^2}) : w_2$,
- a span on opponent name pointers e_O^2 which extends e_O^1 and whose projections are $e_{O,i}^2$,
- a span on opponent name pointers e_P^2 which extends e_P^1 such that $(\rho_1^2(\gamma_1^2), \rho_2^2(\gamma_2^2)) \in \mathcal{G}_{\mathcal{A}} [\![e_P^2]\!]_{e_O^2} w_2$.

Then, from lemma 31,there exists two injections ρ_i^3 extending ρ_i^2 s.t.

$$\langle \lozenge, \rho_i^2(\gamma_i^2), e_{O,i}^2, \rho_i^2(h_i^2), \rho_i^2(\mathcal{D}_i^f) \rangle \xrightarrow{U_i^2} \langle \lozenge, \rho_i^3(\gamma_i^f), e_{O,i}^f, \rho_i^3(h_i^f), \rho_i^3(\mathcal{D}_i^f) \rangle$$

Then using the induction hypothesis on the second lemma, we get the existence of two injections $\rho_i' : (\rho_i^3(\nu_L(h_i^f)) \backslash \rho_i^2(\nu_L(h_i^2))) \hookrightarrow \mathrm{Loc}$ so defining the injection $\rho_i^f : (\nu_L(h_i^f) \backslash \nu_L(h_i))) \hookrightarrow \mathrm{Loc}$ as $\rho_i' \circ \rho_i^3$ on $\nu_L(h_i^f) \backslash \nu_L(h_i^2)$ and as ρ_i^2 on $\nu_L(h_i^2) \backslash \nu_L(h_i^1)$ and as the identity on $\nu_L(h_i^1) \backslash \nu_L(h_i)$, we get the existence of

- a world $w_f \sqsupseteq_{\mathbf{pub}} w_2$ such that $(\rho_1^f(h_1^f), \rho_2^f(h_2^f), (\rho_1^f \times \rho_2^f)(\mathcal{D}^f)) : w_f$,

— a span on opponent name pointers e_O^f which extends e_O^2 and whose projections are $e_{O,i}^f$,
— a span on opponent name pointers e_P^f which extends e_P^2 such that $(\rho_1^f(\gamma_1^f), \rho_2^f(\gamma_2^f)) \in \mathcal{G}_{\mathcal{A}} \left[\!\left[e_P^f \right]\!\right]_{e_O^f} w_f$.

and we conclude using the fact that $w_f \sqsupseteq^*_{\mathbf{pub}} w_2 \sqsupseteq_{\mathbf{pub}} w_1 \sqsupseteq_{\mathbf{pub}} w$. ∎

Finally, using this unwinding of the logical relation, we can show the wanted theorems for values, terms and contexts. One also need an other lemma which show the adequacy of the equivalence of substitutions $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$.

Lemma 39: *Let e_p, e_O two spans respectively on player and opponent name pointers. and $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$. Then $[\cdot] \simeq^{(\gamma_1,\gamma_2),w}_{e_P,e_O} [\cdot]$.*

Theorem 32: *Let v_1, v_2 two values, e_O a span on its opponent names pointers, and w a worlds such that $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\tau]\!]_{e_O} w$. Then taking e_P a span on player name pointers, for all $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$, $[v_1] \simeq^{(\gamma_1,\gamma_2),w}_{e_P,e_O} [v_2]$.*

Theorem 33: *Let K_1, K_2 two contexts, e_O a span on its opponent names pointers, and w, w_0 two worlds such that $(M_1, M_2) \in \mathcal{K}_{\mathcal{A}} [\![\sigma, \tau]\!]_\tau e_O(w, w_0)$. Then taking e_P a span on player name pointers, for all $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$, $[K_1] \simeq^{(\gamma_1,\gamma_2),w}_{e_P,e_O} [K_2]$.*

Theorem 34: *Let M_1, M_2 two terms, e_O a span on its opponent names pointers, and w, w_0 two worlds such that $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_{e_O} (w, w_0)$. Then taking e_P a span on player name pointers, for all $(\gamma_1, \gamma_2) \in \mathcal{G}_{\mathcal{A}} [\![e_P]\!]_{e_O} w$, $[M_1] \simeq^{(\gamma_1,\gamma_2),w}_{e_P,e_O} [M_2]$.*

Proof The lemma and the three theorems are proved by a mutual induction on types and on the length of ground callbacks.

Proof ((Lemma 39)) Let $(h_1, h_2, \mathcal{D}) : w$ s.t. $h_i \in \mathbf{Cl}(\nu_L(\gamma_i) \cup \mathcal{D}_i)$, and $T_1 \in [\langle \Diamond, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1 \rangle]$. We must build $T_2 \in [\langle \Diamond, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2 \rangle]$ such that $T_1 \simeq^{\mathcal{D}}_{e_P,e_O} T_2$. We can write T_1 as

$$(f_1 \left\langle u_1^1 \right\rangle, h^1_{1|\mathcal{D}_1^1}) \cdot U_1^1 \cdot (\left\langle \bar{u_1^2} \right\rangle, h^3_{1|\mathcal{D}_1^3}) \cdot U_1^2$$

with $f_1 \in \mathrm{dom}(\gamma_1)$ and

$$\begin{array}{lll} \langle \Diamond, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1 \rangle & \xrightarrow{f_1 \langle u_1^1 \rangle, h^1_{1|\mathcal{D}_1^1}} & \langle v_1\ u_1^1, \gamma_1, e^1_{O,1}, h_1^1, \mathcal{D}_1^1 \rangle \\ & \xrightarrow{U_1^1} & \langle \widetilde{u_1^2}, \gamma_1^1, e^2_{O,1}, h_1^2, \mathcal{D}_1^2 \rangle \\ & \xrightarrow{\langle \bar{u_1^2} \rangle, h^3_{i|\mathcal{D}_i^3}} & \langle \Diamond, \gamma_1^2, e^2_{O,1}, h_1^3, \mathcal{D}_1^3 \rangle \end{array}$$

Let f_2 such that $(f_1, f_2) \in e_P$, i.e. there exists two types σ, σ' s.t. $(f_1, f_2) \in e_{P,\sigma \to \sigma'}$. From lemma 35, we get the existence of:

— a span on opponent name pointers $e_O^1 \sqsupseteq e_O$,

— a span on locations $\mathcal{D}^1 \sqsupseteq \mathcal{D}$,
— an action $(f_2\,\langle u_2^1\rangle, h^1_{2|\mathcal{D}_2^1})$,

such that $(f_1\,\langle u_1^1\rangle, h^1_{1|\mathcal{D}_1^1}) \sim^{\mathcal{D}^1}_{e_P,e_O^1} (f_2\,\langle u_2^1\rangle, h^1_{2|\mathcal{D}_2^1})$. Then, using lemma 34, we get a world $w_1 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w$ such that $(h_1^1, h_2^1, \mathcal{D}^1) : w_1$. Then, the first move of T_2 is defined as $(f_2\,\langle u_2^1\rangle, h^1_{2|\mathcal{D}_2^1})$.

From $(f_1\,\langle u_1^1\rangle, h^1_{1|\mathcal{D}_1^1}) \sim^{\mathcal{D}^1}_{e_P,e_O^1} (f_2\,\langle u_2^1\rangle, h^1_{2|\mathcal{D}_2^1})$, we get that $(u_1^1, u_2^1) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_{e_O^1} w_1$, so

$$(v_1\ u_1^1, v_2\ u_2^1) \in \mathcal{E}_{\mathcal{A}}[\![\sigma']\!]_{e_O^1}(w_1, w_1).$$

Applying the induction hypothesis of Theorem 34 (*i.e.* on terms) with lemma 30, we first get the existence of

$$U_2^1 \cdot \left(\left\langle \bar{u_2^2}\right\rangle, h^3_{1|\mathcal{D}_2^3}\right) \in \left[\langle v_2\ u_2^1, \gamma_2, e^1_{O,2}, h_2^1, \mathcal{D}_2^1\rangle\right]$$

such that $U_2^1 \simeq^{\mathcal{D}^2}_{e_P,e_O^2} U_2^1$ and $(\left\langle \bar{u_1^2}\right\rangle, h^3_{1|\mathcal{D}_1^3}) \sim^{\mathcal{D}^3}_{e_P^2,e_O^2} (\left\langle \bar{u_2^2}\right\rangle, h^3_{2|\mathcal{D}_2^3})$. Then, applying lemma 37, we get the existence of

— two injections $\rho_i : (\nu_L(h_i^3) \backslash \nu_L(h_i^1)) \hookrightarrow (\mathrm{Loc} \backslash \nu_L(h_i^1))$,
— a world $w_3 \sqsupseteq w_1$
— a span e_P^2 which extends e_P and whose projections are equal to $\mathrm{dom}(\gamma_i^2)$,
— a span e_O^2 which extends e_O and whose projections are equal to $e^2_{O,i}$, ∎

such that $(\rho_1(h_1^3), \rho_2(h_2^3), (\rho_1 \times \rho_2)(\mathcal{D}^3)) : w_3$, $(\rho_1(\gamma_1^2), \rho_2(\gamma_2^2)) \in \mathcal{G}_{\mathcal{A}}[\![e_P^2]\!]_{e_O^2} w_3$ and $(\rho_1(u_1^2), \rho_2(u_2^2)) \in \mathcal{V}_{\mathcal{A}}[\![\sigma']\!]_{e_O^2} w_3$.

Thus, we can use the induction hypothesis to get $U_2^2 \in \left[\langle \Diamond, \gamma_2^2, e^2_{O,2}, h_2^3, \mathcal{D}_2^3\rangle\right]$ such that $\rho_1(U_1^2) \simeq^{\widetilde{\mathcal{D}^3}}_{e_P^1,e_O^2} \rho_2(U_2^2)$ where $\widetilde{\mathcal{D}^3} = (\rho_1 \times \rho_2)(\mathcal{D}^3)$. So, using the renaming of locations (lemma 29), we get that $U_1^2 \simeq^{\mathcal{D}^3}_{e_P^1,e_O^2} U_2^2$. And we conclude taking T_2 as $(f_2\,\langle u_2^1\rangle, h^1_{2|\mathcal{D}_2^1}) \cdot U_2^1 \cdot \left\langle \bar{u_2^2}\right\rangle h^3_{1|\mathcal{D}_2^3}, \cdot U_2^2$

Proof (Soundness on Values) If τ is of atomic type, then the result has already been proven in Theorem 30. Otherwise, taking $(h_1, h_2, \mathcal{D}) : w$ s.t. $h_i \in \mathbf{Cl}(\nu_L(v_i) \cup \mathcal{D}_i)$, we have $[\langle v_i, \gamma_i, e_i, h_i, \mathcal{D}_i\rangle]$ equal to $(\langle \bar{x_i}\rangle, h_{i|\mathcal{D}_i}) \cdot [\langle \Diamond, \gamma_i', e_i, h_i, \mathcal{D}_i\rangle]$ where $\gamma_i' = \gamma_i \cdot (x_i \hookrightarrow v_i)$. And we conclude using lemma 39. ∎

Proof (Soundness on Contexts) Let $(h_1, h_2, \mathcal{D}) : w$ with $h_i \in \mathbf{Cl}(\nu_L(K_i, \gamma_i) \cup \mathcal{D}_i)$, and $T_1 \in [\langle K_1, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle]$, we must build $T_2 \in [\langle K_2, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle]$ such that $T_1 \simeq^{\mathcal{D}}_{e_P,e_O} T_2$. We have $T_1 = U_1^1 \cdot (\langle u_1\rangle, h^2_{1|\mathcal{D}_1^2}) \cdot U_1^2$ with $U_1^1 \in [\langle \Diamond, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle]$ and

$$\begin{array}{lll} \langle K_1, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle & \xrightarrow{U_1^1} & \langle K_1, \gamma_1^1, e^1_{O,1}, h_1^1, \mathcal{D}_1^1\rangle \\ & \xrightarrow{\langle u_1\rangle, h^2_{1|\mathcal{D}_1^2}} & \langle K_1[u_1], \gamma_1^1, e^2_{O,1}, h_1^2, \mathcal{D}_1^2\rangle \end{array}$$

Using lemma 39, we get the existence of $U_2^1 \in [\langle \Diamond, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2\rangle]$ s.t. $U_1^1 \simeq^{\mathcal{D}}_{e_P,e_O} U_2^1$ and $\langle K_2, \gamma_2, e_2, h_2, \mathcal{D}_2\rangle \xrightarrow{U_2^1} \langle K_2, \gamma_2^1, e^1_{O,2}, h_2^1, \mathcal{D}_2^1\rangle$. Then using lemma 38, we get the existence of

— two injections $\rho_i : (\nu_L(h_i^3) \backslash \nu_L(h_i^1)) \hookrightarrow (\mathrm{Loc} \backslash \nu_L(h_i^1))$,
— a world $w_1 \sqsupseteq w$
— a span e_P^1 which extends e_P and whose projections are equal to $\mathrm{dom}(\gamma_i^1)$,
— a span e_O^1 which extends e_O and whose projections are equal to $e_{O,i}^1$,
such that $(\rho_1(h_1^1), \rho_2(h_2^1), (\rho_1 \times \rho_2)(\mathcal{D}^1)) : w_1$, and $(\rho_1(\gamma_1^1), \rho_2(\gamma_2^1)) \in \mathcal{G}_{\mathcal{A}} [\![e_P^1]\!]_{e_O^1} w_1$.

From lemma 36, we get the existence of:
— a span on opponent name pointers $e_O^2 \sqsupseteq e_O^1$,
— a span on locations $\mathcal{D}^2 \sqsupseteq \mathcal{D}$,
— an action $(\langle u_2 \rangle, h^2_{2|\mathcal{D}_2^1})$,
such that $(\langle u_1 \rangle, h^2_{1|\mathcal{D}_1^1}) \sim^{\mathcal{D}^2}_{e_P, e_O^2} (\langle u_2 \rangle, h^2_{2|\mathcal{D}_2^1})$. Then, using lemma 34, we get a world $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$ such that $(h_1^2, h_2^2, \mathcal{D}^2) : w_2$. Then, from the equivalence between the two opponent actions, we get that $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_{e_O^2} w_2$.

Thus using the monotonicity of contexts, we have $(K_1[u_1], K_2[u_2]) \in \mathcal{K}_{\mathcal{A}} [\![\sigma, \tau]\!]_{e_O^2} (w_2, w_2)$, so using the induction hypothesis on terms, we get the existence of U_2^2 such that $\rho_1'(U_1^2) \simeq^{\mathcal{D}_3'}_{e_P^1, e_O^2} \rho_1'(U_2^2)$. And we conclude using the renaming of locations (lemma 29). ∎

Proof (Soundness on Terms) Straightforward using the main theorem on Kripke trace semantics (theorem 31). ∎

Then, we can deduce the soundness of concrete logical relations *w.r.t.* trace semantics

Corollary 5: *Let M_1, M_2 two terms such that $\Sigma; \Gamma_g, \Gamma_f \vdash M_1, M_2 : \tau$. Then $\Sigma; \Gamma_g, \Gamma_f \vdash M_1 \simeq_{clog} M_2 : \tau$ implies $[\Sigma; \Gamma_g, \Gamma_f \vdash M_1 : \tau] = [\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$.*

Proof From $\Sigma; \Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$ we get the existence of an LTS $\mathcal{A}$ and a state s of $\mathcal{A}$ such that for all $\gamma_g : \Gamma_g \to \mathrm{Val}$ and $h \in \mathbf{Cl}(\Sigma, \nu_L(\mathrm{codom}(\gamma_g)))$, defining w as $(s, h, h, \widetilde{\nu_L(h)})$, we get $(\gamma_{g,1}(M_1), \gamma_{g,2}(M_2)) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_{\widetilde{\Gamma_f}} (w, w)$. So Theorem 34 gives us that $[M_1] \simeq^{(\varepsilon,\varepsilon),w}_{\varepsilon,\widetilde{\Gamma_f}} [M_2]$. We conclude using Theorem 29. ∎

6.5 Adequate Worlds

Completeness can no longer be proven abstractly using biorthogonality. That is, for any two terms that are contextually equivalent, we need to construct explicitly a LTS for which they are logically related. To make the proof more abstract and less dependent on the LTS construction, we define a notion of adequate world *w.r.t.* an LTS $\mathcal{A}$ that expresses all we need to know about evolution of worlds to prove that two terms are related. The definition, given in Figure 6.3, is a dual version of concrete logical relations. Indeed, instead of characterizing terms that are in relation for a given world, it characterizes worlds that respects the execution of two given terms M_1, M_2. Namely, it imposes:

$$
\begin{array}{lcl}
\overline{\mathcal{V}}_{\mathcal{A}}[\![\mathrm{Unit}]\!]^{\gamma}_{e_P,e_O}(u_1,u_2) & \stackrel{def}{=} & \mathrm{World} \\
\overline{\mathcal{V}}_{\mathcal{A}}[\![\mathrm{Bool}]\!]^{\gamma}_{e_P,e_O}(u_1,u_2) & \stackrel{def}{=} & \mathrm{World} \\
\overline{\mathcal{V}}_{\mathcal{A}}[\![\mathrm{Int}]\!]^{\gamma}_{e_P,e_O}(u_1,u_2) & \stackrel{def}{=} & \mathrm{World} \\
\overline{\mathcal{V}}_{\mathcal{A}}[\![\mathrm{ref}\,\iota]\!]^{\gamma}_{e_P,e_O}(u_1,u_2) & \stackrel{def}{=} & \{w \mid (u_1,u_2) \in w.\mathcal{S}_\iota\}
\end{array}
$$

$$
\begin{array}{l}
\overline{\mathcal{V}}_{\mathcal{A}}[\![\iota \to \tau]\!]^{\gamma}_{e_P,e_O}(v_1,v_2) \stackrel{def}{=} \\
\quad \{w \mid \forall w_1 \sqsupseteq^* w.\forall(h_1^1,h_2^1,\mathcal{D}^1) : w_1.\exists(h_1,h_2,\mathcal{D}) : w.\exists e_P^1 \sqsupseteq e_P.\exists e_O^1 \sqsupseteq e_O.\exists T_1,T_2. \\
\quad\; T_1 \simeq^{\mathcal{D}^1}_{e_P^1,e_O^1} T_2 \wedge \langle \Diamond, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle \xrightarrow{T_i} \langle \overrightarrow{K_i}, \gamma_i^1, e_{O,i}^1, h_i^1, \mathcal{D}_i^1\rangle \\
\quad\; \wedge \exists w_2 \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w_1.\forall(u_1,u_2) \in \mathcal{V}_{\mathcal{A}}[\![\iota]\!]_{e_O} w_2.(w_2,w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_P^1,e_O^1}(\gamma_1(v_1)\ u_1, \gamma_2(v_2)\ u_2)\}
\end{array}
$$

$$
\begin{array}{l}
\overline{\mathcal{V}}_{\mathcal{A}}[\![\sigma \to \tau]\!]^{\gamma}_{e_P,e_O}(v_1,v_2) \stackrel{def}{=} \\
\quad \{w \mid \forall w_1 \sqsupseteq^* w.\forall(h_1^1,h_2^1,\mathcal{D}^1) : w_1.\exists(h_1,h_2,\mathcal{D}) : w.\exists e_P^1 \sqsupseteq e_P.\exists e_O^1 \sqsupseteq e_O.\exists T_1,T_2. \\
\quad\; T_1 \simeq^{\mathcal{D}^1}_{e_P,e_O^1} T_2 \wedge \langle \Diamond, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle \xrightarrow{T_i} \langle \overrightarrow{K_i}, \gamma_i^1, e_{O,i}^1, h_i^1, \mathcal{D}_i^1\rangle \\
\quad\; \wedge \exists w_2 \sqsupseteq^{\mathcal{F}_\sigma}_{\mathbf{pub}} w_1.(w_2,w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_P^1,e_O^1\cdot(y_1,y_2)}(\gamma_1(v_1)\ y_1, \gamma_2(v_2)\ y_2)\}
\end{array}
$$

$$
\begin{array}{l}
\overline{\mathcal{K}}_{\mathcal{A}}[\![\iota,\tau]\!]^{\gamma}_{e_P,e_O}(K_1,K_2) \stackrel{def}{=} \\
\quad \{(w,w_0) \mid \forall w_1 \sqsupseteq^*_{\mathbf{pub}} w.\forall(h_1^1,h_2^1,\mathcal{D}^1) : w_1.\exists(h_1,h_2,\mathcal{D}) : w.\exists e_P^1 \sqsupseteq e_P.\exists e_O^1 \sqsupseteq e_O. \\
\quad\; \exists T_1,T_2.T_1 \simeq^{\mathcal{D}^1}_{e_P^1,e_O^1} T_2 \wedge\ \langle K_i[\bullet], \gamma_i, e_i, h_i, \mathcal{D}_i\rangle \xrightarrow{T_i} \langle K_i[\bullet], \gamma_i^1, e_i^1, h_i^1, \mathcal{D}_i^1\rangle \\
\quad\; \wedge \exists w_2 \sqsupseteq^{\mathcal{F}_\sigma}_{\mathbf{pub}} w_1.\forall(v_1,v_2) \in \mathcal{V}_{\mathcal{A}}[\![\iota]\!]_{e_O} w_2.(w_2,w_0) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_p^1,e_O^1}(K_1[v_1],K_2[v_2])\}
\end{array}
$$

$$
\begin{array}{l}
\overline{\mathcal{K}}_{\mathcal{A}}[\![\sigma,\tau]\!]^{\gamma}_{e_P,e_O}(K_1,K_2) \stackrel{def}{=} \\
\quad \{(w,w_0) \mid \forall w_1 \sqsupseteq^*_{\mathbf{pub}} w.\forall(h_1^1,h_2^1,\mathcal{D}^1) : w_1.\exists(h_1,h_2,\mathcal{D}) : w.\exists e_P^1 \sqsupseteq e_P.\exists e_O^1 \sqsupseteq e_O. \\
\quad\; \exists T_1,T_2.T_1 \simeq^{\mathcal{D}}_{e_P,e_O} T_2 \wedge \langle K_i[\bullet], \gamma_i, e_i, h_i, \mathcal{D}_i\rangle \xrightarrow{T_i} \langle K_i[\bullet], \gamma_i^1, e_{O,i}^1, h_i^1, \mathcal{D}_i^1\rangle \\
\quad\; \wedge \exists w_2 \sqsupseteq^{\mathcal{F}_\sigma}_{\mathbf{pub}} w_1.(w_2,w_0) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_p^1,e_O^1\cdot(y_1,y_2)}(K_1[y_1],K_2[y_2])\}
\end{array}
$$

$$
\begin{array}{l}
\bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1,M_2) \stackrel{def}{=} \\
\quad \Big\{(w,w_0) \mid \forall(h_1,h_2,\mathcal{D}) : w.\forall e_P^1 \sqsupseteq e_P.\forall \mathcal{D}^1 \sqsupseteq \mathcal{D}.\forall c_1 \sim^{\mathcal{D}^1}_{e_P^1,e_O} c_2. \\
\quad\; \langle M_i,\gamma_i,e_i,h_i,\mathcal{D}_i\rangle \xrightarrow{c_i} \langle E_i,\gamma_i^1,e_{O,i},h_i^1,\mathcal{D}_i^1\rangle \Rightarrow \Big(\exists w_1 \sqsupseteq w.(h_1^1,h_2^1,\mathcal{D}^1) : w_1 \wedge \\
\qquad \Big((c_i = (\langle \bar{v_i}\rangle, h^1_{i|\mathcal{D}_i^1})) \wedge w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_P^1,e_O}(v_1,v_2) \wedge w_1 \sqsupseteq_{\mathbf{pub}} w_0\Big) \text{ or} \\
\qquad \Big((c_i = (\bar{f}_i\,\langle v_i\rangle, h^1_{i|\mathcal{D}_i^1})) \wedge (f_1,f_2) \in e_{O,\sigma\to\sigma'} \wedge w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\sigma]\!]^{\gamma^1}_{e_P^1,e_O}(v_1,v_2) \\
\qquad\quad \wedge (w_1,w_0) \in \overline{\mathcal{K}}_{\mathcal{A}}[\![\sigma',\tau]\!]^{\gamma^1}_{e_P^1,e_O}(E_1,E_2)\Big)\Big)\Big\}
\end{array}
$$

Figure 6.3: Definition of adequate worlds

— the existence of future worlds for all possible heaps that can be obtained by reducing M_1, M_2 from two heaps in the current world (see the definition of $\bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1, M_2)$).
— that every future world corresponds to a reduction, either public in the definition of $\bar{\mathcal{K}}_{\mathcal{A}}[\![\tau, \sigma]\!]^{\gamma}_{e_P,e_O}(K_1, K_2)$, or private in the definition of $\bar{\mathcal{V}}_{\mathcal{A}}[\![\tau \to \sigma]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$.

Its definition is parametrized by two spans e_P, e_O on player and opponent name pointers, and by a couple of substitutions, written γ, whose two components are written γ_1 and γ_2 and are of domain respectively $e_{P,1}$ and e_{P_2}. They are used to control the functional values disclosed by the term.

Notice that, from $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e w$, it is not true that $w \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\varepsilon}_{\varepsilon,e}(M_1, M_2)$. Indeed, one do not need w to be adequate to prove that $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e w$. This notion is only necessary when working on completeness. It is also not true that $w \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\varepsilon}_{\varepsilon,e}(M_1, M_2)$ implies $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e w$, because we do not impose equality on ground values, since it is not needed: when working with contextually equivalent terms, this is automatic.

To show that, for any pair of contextually equivalent terms M_1, M_2,, there exists an LTS $\mathcal{A}$ and an adequate world *w.r.t.* $\mathcal{A}$, we build *coinductively* an *exhaustive LTS*, *i.e.* an infinite transition system where each state represents a configuration which can appear in the interactive reduction *of the terms*, and where transitions constraint heaps to be in accordance with the reduction. To build it, we first define some basic operations on transition system:

— Add a private transition r to an transition system $\mathcal{A} = (\delta_{\mathbf{priv}}, \delta_{\mathbf{pub}})$:

$$\mathcal{A} \overset{\mathbf{priv}}{\oplus} r \overset{def}{=} (\delta_{\mathbf{priv}} \cup \{r\}, \delta_{\mathbf{pub}})$$

— Add a set R of public transition to an transition system $\mathcal{A} = (\delta_{\mathbf{priv}}, \delta_{\mathbf{pub}})$:

$$A \overset{\mathbf{pub}}{\oplus} R \overset{def}{=} (\delta_{\mathbf{priv}} \cup R, \delta_{\mathbf{pub}} \cup R)$$

— Union of two transition systems $\mathcal{A}_1, \mathcal{A}_2$ where $\mathcal{A}_i = (\delta_{i,\mathbf{priv}}, \delta_{i,\mathbf{pub}})$:

$$\mathcal{A}_1 \sqcup \mathcal{A}_2 \overset{def}{=} (\delta_{1,\mathbf{priv}} \cdot \delta_{2,\mathbf{priv}}, \delta_{1,\mathbf{pub}} \cdot \delta_{2,\mathbf{pub}})$$

Then, we define operations to construct transitions :

— $\mathbf{TransPriv}(s, s', h_1, h_2, h'_1, h'_2, \mathcal{D}, \mathcal{D}')$ is defined as the private transition

$$(s, s') \hookrightarrow \{(h_{1|\mathcal{D}_1}, h_{2|\mathcal{D}_2}, h'_{1|\mathcal{D}'_1}, h'_{2|\mathcal{D}'_2}, \mathcal{D}, \mathcal{D}')\}$$

— $\mathbf{TransPub}(c_1, c_2)(s, s', h_1, h_2, h'_1, h'_2, \mathcal{D}, \mathcal{D}')$ is defined as the singleton formed by the public transition

$$(s, s') \hookrightarrow \{(h_{1|\mathcal{D}_1}, h_{2|\mathcal{D}_2}, h'_{1|\mathcal{D}'_1}, h'_{2|\mathcal{D}'_2}, \mathcal{D}, \mathcal{D}')\}$$

if the a_i are either player or opponent answers or opponent questions, otherwise, it is defined as the empty set.

Then, we define the exhaustive LTS $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ and $\mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ associated respectively to two opponent and player configurations C_1, C_2. They use a list L of worlds corresponding to the public transitions that must be added to the LTS once a value is reached in the interactive reduction.

— $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ is defined as

$$\bigsqcup_j \Big(\mathbf{LtsK}^{L^j,s^j}_{e_P^j,e_O,\mathcal{D}'}(C_1^j,C_2^j) \overset{\mathbf{priv}}{\oplus} \mathbf{TransPriv}(s,s^j,h_1,h_2,h_1^j,h_2^j,\mathcal{D},\mathcal{D}^j) \overset{\mathbf{pub}}{\oplus} \mathbf{TransPub}(c_1,c_2)(s_0,s^j,h_1^0,h_2^0,h_1^j,h_2^j,\mathcal{D}^0,\mathcal{D}^j) \Big)$$

for all pair of players actions (c_1^j, c_2^j) s.t. the exists a span on player name pointers $e_P^j \sqsupseteq e_P$ and a span on locations $\mathcal{D}^j \sqsupseteq \mathcal{D}$ with $c_1^j \sim^{\mathcal{D}^j}_{e_P^j,e_O} c_2^j$ and there exists $\widetilde{C_i}$ with $\widetilde{h_{1|\mathcal{D}_1}} \sim_{\mathcal{D}} \widetilde{h_{2|\mathcal{D}_2}}$, $\widetilde{h_{i|\overline{\mathcal{D}_i}}} = h_{i|\overline{\mathcal{D}_i}}$ and $\widetilde{C_i} \xrightarrow{c_i^j} C_i^j$, where we write h_i (resp. $\widetilde{h_i}$) for the heap of C_i (resp. $\widetilde{C_i}$). If the c_i are (player) questions, then we suppose that $L^j = L$ (and by definition of **TransPub** we do not add any public transition). Otherwise, we suppose that $L^j = (s^0, h_1^0, h_2^0, \mathcal{D}^0) \cdot L'$.

— $\mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ is defined as

$$\bigsqcup_j \Big(\mathbf{LtsE}^{L^j,s^j}_{e_P,e_O^j,\mathcal{D}^j}(C_1^j,C_2^j) \overset{\mathbf{pub}}{\oplus} \mathbf{TransPub}(c_1^j,c_2^j)(s,s^j,h_1,h_2,h_1',h_2',\mathcal{D},\mathcal{D}^j) \Big)$$

for all pair of opponent actions (c_1^j, c_2^j) s.t. the exists a span on opponent name pointers $e_O^j \sqsupseteq e_O$ and a span on locations $\mathcal{D}^j \sqsupseteq \mathcal{D}$ with $c_1^j \sim^{\mathcal{D}^j}_{e_P,e_O^j} c_2^j$ and $C_i \xrightarrow{c_i^j} C_i^j$. We suppose that the s^j are fresh and write h_i (resp. h_i^j) for the heap of C_i (resp. C_i^j). If the c_i^j are (opponent) questions, then $L^j = (s, h_1^j, h_2^j, \mathcal{D}^j) \cdot L$. Otherwise, $L^j = L$.

The definition of $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ needs to be generalized to consider any configurations $\widetilde{C_1}, \widetilde{C_2}$ whose private part of the heaps are equal to the one of h_1, h_2, but whose disclosed part can be changed, as soon as they are still nominally equivalent. This is due to the fact that the definition of $\overline{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1,M_2)$ quantifies over $(h_1,h_2,\mathcal{D}) : w$, which do not constraint the disclosed part of h_1, h_2 beside being nominally equivalent. An other solution would have been to merge in the definition of $\mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ the states s^j for all the configurations C_i^j whose heaps only differ on the disclosed part (but whose $\mathcal{D}_i^j$ are equal). However, doing so, the definition of $\mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$ become in a way primitive over the definition of $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1,C_2)$, which would complicate the reasoning.

The exhaustive LTS $\mathcal{A} = (\delta_{\mathbf{priv}}, \delta_{\mathbf{pub}})$ has a tree-structure, i.e. there does not exist two states s, s' s.t. $s' \geq_{\mathcal{A}} s$ and $s \geq_{\mathcal{A}} s'$, where we define $s' \geq_{\mathcal{A}} s$ as $\delta^*_{\mathbf{priv}}(s,s')$ non empty. Thus, for any state s of $\mathcal{A}$ we can define the sub-LTS $\mathcal{A}_{\geq s}$ whose transition functions are

restricted to states $s' \geq_{\mathcal{A}} s$. Then, one can relate the properties of two LTS $\mathcal{A}, \mathcal{A}'$ about worlds w, when $\mathcal{A}_{\geq s} = \mathcal{A}'_{\geq s}$ and the state of w is greater than s.

Lemma 40: *Let $w = (s, h_1, h_2, \mathcal{D})$ a world s.t. $w \in \overline{\mathcal{V}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$. Let $\mathcal{A}'$ an LTS s.t. $\mathcal{A}_{\geq s} = \mathcal{A}'_{\geq s}$, then $w \in \overline{\mathcal{V}}_{\mathcal{A}'}[\![\tau]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$.*

Lemma 41: *Let $w = (s, h_1, h_2, \mathcal{D})$ and $w_0 = (s_0, h_1^0, h_2^0, \mathcal{D}^0)$ two worlds s.t. $(w, w_0) \in \overline{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1, M_2)$. Let $\mathcal{A}'$ an LTS extending $\mathcal{A}$ s.t. $\mathcal{A}_{\geq s} = \mathcal{A}'_{\geq s}$ and $\{s' \mid \mathcal{A}.\delta_{\mathbf{pub}}(s_0, s') \wedge s' \geq_{\mathcal{A}} s\} = \{s' \mid \mathcal{A}'.\delta_{\mathbf{pub}}(s_0, s') \wedge s' \geq_{\mathcal{A}'} s\}$, then $(w, w_0) \in \overline{\mathcal{E}}_{\mathcal{A}'}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1, M_2)$.*

Proof Straightforward from the fact that $\mathcal{A}$ and $\mathcal{A}'$ impose the same futures for w. ∎

Before proving that the exhaustive LTS is adequate, we need some lemmas which ensure the existence of equivalent traces from the existence of a succession of (public for the first lemma, private for the second) transitions in an LTS.

Lemma 42: *Let C_1, C_2 two interactive configurations such that $C_i = \langle \Diamond, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle$. We consider the exhaustive LTS $\mathcal{A} = \mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ and $s', h'_1, h'_2, \mathcal{D}'$ s.t.*

$$(h_1, h_2, h'_1, h'_2, \mathcal{D}, \mathcal{D}') \in \mathcal{A}.\delta^*_{\mathbf{pub}}(s, s')$$

Then there exists two traces T_1, T_2, two spans $e'_P \sqsupseteq e_P$, $e'_O \sqsupseteq e_O$ s.t. $T_1 \simeq^{\mathcal{D}'}_{e'_P,e'_O} T_2$ and $C_i \xrightarrow{T_i} \underbrace{\langle \Diamond, \gamma'_i, e'_O, h'_i, \mathcal{D}'_i \rangle}_{C'_i}$, and $\mathcal{A}'_{\geq s'} = \mathcal{A}_{\geq s'}$, where $\mathcal{A}' = \mathbf{LtsK}^{L,s'}_{e'_P,e'_O,\mathcal{D}'}(C'_1, C'_2)$.

Lemma 43: *Let C_1, C_2 two interactive configurations such that $C_i = \langle \Diamond, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle$. We consider the exhaustive LTS $\mathcal{A} = \mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ and $s', h'_1, h'_2, \mathcal{D}'$ s.t.*

$$(h_{1|\overline{\mathcal{D}_1}}, h_{2|\overline{\mathcal{D}_2}}, h'_{1|\overline{\mathcal{D}'_1}}, h'_{2|\overline{\mathcal{D}'_2}}, \mathcal{D}, \mathcal{D}') \in \mathcal{A}.\delta^*_{\mathbf{priv}}(s, s')$$

Then there exists two traces T_1, T_2 s.t. $T_1 \simeq^{\mathcal{D}'}_{e'_P,e'_O} T_2$ and $C_i \xrightarrow{T_i} C'_i$ where $C'_i = \langle \overrightarrow{K_i}, \gamma'_i, e'_O, h'_i, \mathcal{D}'_i \rangle$, and $\mathcal{A}'_{\geq s'} = \mathcal{A}_{\geq s'}$, where $\mathcal{A}' = \mathbf{LtsK}^{L^1,s'}_{e'_P,e'_O,\mathcal{D}'}(C'_1, C'_2)$.

Finally, we can prove that the exhaustive LTS is adequate.

Theorem 35 (Adequacy on Values): *Let C_1, C_2 two opponent interactive configurations such that $C_i = \langle \Diamond, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle$. Then defining $\mathcal{A}$ as $\mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ and w as $(s, h_{1|\overline{\mathcal{D}_1}}, h_{2|\overline{\mathcal{D}_2}}, \mathcal{D})$, for all $(f_1, f_2) \in e_{P,\sigma\to\tau}$, $w \in \overline{\mathcal{V}}_{\mathcal{A}}[\![\sigma \to \tau]\!]^{\gamma}_{e_P,e_O}(f_1, f_2)$.*

Theorem 36 (Adequacy on Contexts): *Let K_1, K_2 two contexts of type $\sigma \rightsquigarrow \tau$ and C_1, C_2 two interactive configurations such that $C_i = \langle K_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle$. Then defining $\mathcal{A}$ as $\mathbf{LtsK}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ where $L = (s_0, h_1^0, h_2^0, \mathcal{D}^0) \cdot L'$, $w = (s, h_{1|\overline{\mathcal{D}_1}}, h_{2|\overline{\mathcal{D}_2}}, \mathcal{D})$ and $w_0 = (s_0, h^0_{1|\mathcal{D}_1^0}, h^0_{2|\mathcal{D}_2^0}, \mathcal{D}^0)$, we have $(w, w_0) \in \overline{\mathcal{K}}_{\mathcal{A}}[\![\sigma, \tau]\!]^{\gamma}_{e_P,e_O}(K_1, K_2)$.*

Theorem 37 (Adequacy on Terms): *Let M_1, M_2 two terms of type τ and C_1, C_2 two interactive configurations with $C_i = \langle M_i, \gamma_i, e_i, h_i, \mathcal{D}_i\rangle$. Then defining $\mathcal{A}$ as $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ where $L = (s_0, h^0_1, h^0_2, \mathcal{D}^0) \cdot L'$, $w = (s, h_{1|\overline{\mathcal{D}_1}}, h_{2|\overline{\mathcal{D}_2}}, \mathcal{D})$ and $w_0 = (s_0, h^0_{1|\mathcal{D}^0_1}, h^0_{2|\mathcal{D}^0_2}, \mathcal{D}^0)$, we have $(w, w_0) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1, M_2)$.*

Proof The three theorem are proved by a mutual coinduction.

Proof (Adequacy on Values) Let $w_1 \sqsupseteq^* w$ with s_1 the state of w_1 and and $(h^1_1, h^1_2, \mathcal{D}^1) : w_1$. From lemma 43, there exists two traces T_1, T_2 s.t. $T_1 \simeq^{\mathcal{D}^1}_{e^1_P,e^1_O} T_2$, with $C_i \xrightarrow{T_i} C^1_i$ where $C^1_i = \langle \overrightarrow{K_i}, \gamma^1_i, e^1_O, h^1_i, \mathcal{D}^1_i\rangle$. Moreover, defining $\mathcal{A}^1$ as $\mathbf{LtsK}^{L,s_1}_{e^1_P,e^1_O,\mathcal{D}^1}(C^1_1, C^1_2)$, we have $\mathcal{A}^1_{\geq s_1} = \mathcal{A}_{\geq s_1}$.

Then, $C^1_i \xrightarrow{f_i\langle u_i\rangle, h^2_{i|D^2_i}} C^2_i$ where $C^2_i = \langle (v_i\ u_i) \cdot \overrightarrow{K_i}, \gamma^1_i, e^2_O, h^2_i, \mathcal{D}^2_i\rangle$ with $\gamma_(f_i) = v_i$. So by definition of $\mathcal{A}^1$, there exists a state s_2 s.t. $(h^1_{1|\overline{\mathcal{D}^1_1}}, h^1_{2|\overline{\mathcal{D}^1_2}}, h^2_{1|\overline{\mathcal{D}^2_1}}, h^2_{2|\overline{\mathcal{D}^2_2}}, \mathcal{D}^1, \mathcal{D}^2) \in \mathcal{A}^1.\delta_{pub}(s_1, s_2)$. Moreover, defining $\mathcal{A}^2$ as $\mathbf{LtsE}^{L,s_2}_{e^1_P,e^2_O,\mathcal{D}^2}(C^2_1, C^2_2)$, we have $\mathcal{A}^1_{\geq s_2} = \mathcal{A}^2_{\geq s_2}$. Then, defining w_2 as $(s_2, h^2_{1|D^2_1}, h^2_{2|D^2_2}, \mathcal{D}^2)$, the coinduction hypothesis gives us that

$$(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}^2}[\![\tau]\!]^{\gamma}_{e_P,e_O}(v_1\ u_1, v_2\ u_2)$$

So using lemma 41, we get that $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(v_1\ u_1, v_2\ u_2)$. ∎

Proof (Adequacy on Terms) Let $(\widetilde{h_1}, \widetilde{h_2}, \mathcal{D}) : w$ s.t. $\widetilde{h_1} \in \mathbf{Cl}(\nu_L(M_i, \gamma_i))$, then $\widetilde{h_{1|\mathcal{D}_1}} \sim_{\mathcal{D}} \widetilde{h_{2|\mathcal{D}_2}}$ and $\widetilde{h_{i|\overline{\mathcal{D}_i}}} = h_{i|\overline{\mathcal{D}_i}}$ so we define $\widetilde{C_i}$ as $\langle M_i, \gamma_i, e_{O,i}, \widetilde{h_i}, \mathcal{D}_i\rangle$. Suppose that there exists two player actions c_i, a span on player name pointers $e^1_P \sqsupseteq e_P$ and a span on locations $\mathcal{D}^1 \sqsupseteq \mathcal{D}$ s.t. $c_1 \sim^{\mathcal{D}^1}_{e^1_P,e_O} c_2$, and there exists two opponent configurations C^1_i s.t. $\widetilde{C_i} \xrightarrow{c_i} C^1_i$. Let write C^1_i as $\langle E_i, \gamma^1_i, e^1_{O,i}, h^1_i, \mathcal{D}^1_i\rangle$. Then by definition of $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$, there exists a fresh state s_1 such that, writing $\mathcal{A}^1$ for $\mathbf{LtsK}^{L',s_1}_{e^1_P,e_O,\mathcal{D}^1}(C^1_1, C^1_2)$, we have $(h_{1|\overline{\mathcal{D}_1}}, h_{2|\overline{\mathcal{D}_2}}, h^1_{1|\overline{\mathcal{D}^1_1}}, h^1_{2|\overline{\mathcal{D}^1_2}}, \mathcal{D}, \mathcal{D}^1) \in \delta_{priv}(s, s_1)$ and $\mathcal{A}^1_{\geq s_1} = \mathcal{A}_{\geq s_1}$. Let write w_1 for $(s_1, h^1_{1|\overline{\mathcal{D}^1_1}}, h^1_{2|\overline{\mathcal{D}^1_2}}, \mathcal{D}^1)$, so $w_1 \sqsupseteq w$. And from $c_1 \sim^{\mathcal{D}^1}_{e^1_P,e_O} c_2$, we get that $h^1_{1|\mathcal{D}_1} \sim_{\mathcal{D}^1} h^1_{2|\mathcal{D}_2}$, so $(h^1_1, h^1_2, \mathcal{D}^1) : w_1$.

— If both c_i are player answers, i.e. $c_i = (\langle \bar{v_i}\rangle, h^1_{i|D^1_i})$, then $C^1_i = \langle \Diamond, \gamma^1_i, e_{O,i}, h^1_i, \mathcal{D}^1_i\rangle$. If the v_i are ground values, then $\gamma^1_i = \gamma_i$ and either they are of type Unit, Bool or Int, in which case we directly have $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}^1}[\![\tau]\!]^{\gamma^1}_{e^1_P,e_O}(v_1, v_2)$, or they are of type ref ι. In such case, $(v_1, v_2) \in \mathcal{D}^1$ so we also get $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}^1}[\![\tau]\!]^{\gamma^1}_{e^1_P,e_O}(v_1, v_2)$. Otherwise, they are name pointers, and the coinduction hypothesis on Values applied to v_1, v_2 gives us that $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}^1}[\![\tau]\!]^{\gamma^1}_{e^1_P,e_O}(\gamma(v_1), \gamma(v_2))$. So using new lemma 40, we get that $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e^1_P,e_O}(v_1, v_2)$. Moreover, the definition of $\mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ gives us that $(h^0_{1|\overline{\mathcal{D}^0_1}}, h^0_{2|\overline{\mathcal{D}^0_2}}, h^1_{1|\overline{\mathcal{D}^1_1}}, h^1_{2|\overline{\mathcal{D}^1_2}}, \mathcal{D}^0, \mathcal{D}^1) \in \delta_{pub}(s_0, s_1)$. Thus we have $w_1 \sqsupseteq_{\mathbf{pub}} w_0$.

— If both c_i are player questions, i.e. $c_i = (\bar{f}_i \langle v_i \rangle, h^1_{i|D^1_i})$, then $C^1_i = \langle K_i[\bullet], \gamma^1_i, e_{O,i}, h^1_i, \mathcal{D}^1_i \rangle$. Then there exists σ, σ' such that $(f_1, f_2) \in e_{O,\sigma\to\sigma'}$. The coinduction hypothesis on Contexts gives us that $(w_1, w_0) \in \bar{\mathcal{K}}_{\mathcal{A}}[\![\sigma', \tau]\!]^{\gamma^1}_{e^1_P, e_O}(K_1, K_2)$. So, we just have to prove that $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\sigma]\!]^{\gamma^1}_{e^1_P, e_O}(v_1, v_2)$, which is done in the same way than when the c_i are answers. ∎

Proof (Adequacy on Contexts) Let $w_1 \sqsupseteq^*_{\mathbf{pub}} w$ and $(h_1, h_2, \mathcal{D}) : w_1$ s.t. $h_i \in \mathbf{Cl}(\nu_L(K_i))$, with s_1 the state of w_1. Then, from lemma 42, there exists two traces T_1, T_2, three spans $e^1_P \sqsupseteq e_P$, $e^1_O \sqsupseteq e_O$, and $\mathcal{D}^1 \sqsupseteq \mathcal{D}$, s.t. $T_1 \simeq^{\mathcal{D}^1}_{e^1_P, e^1_O} T_2$ and $C_i \xrightarrow{T_i} C^1_i$ where $C^1_i = \langle K_i, \gamma^1_i, e^1_O, h^1_i, \mathcal{D}^1_i \rangle$ with $\mathrm{dom}(\gamma^1_i) = e^1_{P,i}$. Moreover, writing $\mathcal{A}^1$ for $\mathbf{LtsK}^{L,s}_{e^1_P, e^1_O, \mathcal{D}^1}(C^1_1, C^1_2)$, we have $\mathcal{A}^1_{\geq s_1} = \mathcal{A}_{\geq s_1}$. Then, we reason by case analysis on σ.

If σ is functional, then, by construction of $\mathcal{A}^1$, there exists a state s_2 and a span $\mathcal{D}^2$ s.t. $(h^1_1, h^1_2, h^1_1, h^1_2, \mathcal{D}^1, \mathcal{D}^2) \in \mathcal{A}^1.\delta_{pub}(s_1, s_2)$, and

$$C^1_i \xrightarrow{\langle x_i \rangle, h^2_{i|D^2_i}} \underbrace{\langle K_i[x_i], \gamma^1_i, e^2_O, h^2_i, \mathcal{D}^2_i \rangle}_{C^2_i}$$

∎

where $e^2_O = e^1_O \cdot (x_1, x_2)$. Moreover, defining $\mathcal{A}^2$ as $\mathbf{LtsE}^{L,s}_{e^1_P, e^2_O, \mathcal{D}^2}(C^2_1, C^2_2)$, we have $\mathcal{A}^2_{\geq s_2} = \mathcal{A}^1_{\geq s_2}$. Then, defining w_2 as $(s_2, h^2_{1|D^2_1}, h^2_{2|D^2_2}, \mathcal{D}^2)$, the coinduction hypothesis gives us that $(w_2, w_0) \in \bar{\mathcal{E}}_{\mathcal{A}^2}[\![\tau]\!]^{\gamma^1}_{e^2_P, e^2_O}(K_1[x_1], K_2[x_2])$ Finally, $w_2 \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} w$ and, using lemma 41, $(w_2, w_0) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e^1_P, e^2_O}(K_1[x_1], K_2[x_2])$.

The case where σ is ground is done in the same way. ∎

6.6 From Trace Semantics to Concrete Logical Relations

Using this construction of an exhaustive LTS which is *adequate*, we can now prove the completeness of our concrete logical relations. We first prove it for ground values

Theorem 38: *Let v_1, v_2 two closed values of ground type ι, e_O a span on opponent name pointers and w a world s.t. $w \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\iota]\!]^{\gamma}_{e_P, e_O}(v_1, v_2)$. Then if $[v_1] \simeq^{\gamma, w}_{e_P, e_O} [v_2]$ we get that $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\iota]\!]_{e_O} w$.*

Proof If $\iota = \mathrm{Int}$, the theorem is direct since in both cases we have $v_1 = v_2$. If $\iota = \mathrm{ref}\, \iota'$, then from $w \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\iota]\!]^{\gamma}_{e_P, e_O}(v_1, v_2)$ we get that $(v_1, v_2) \in (w.\mathcal{S})_\iota$. ∎

To prove the completeness of concrete logical relations at higher types, we decompose traces to show that they respect the inductive structure induced by Kripke logical relations. To do so, we need invariants on heaps provided by adequate worlds. Indeed, the equivalence of Kripke trace semantics $\simeq^{\gamma,w}_{e_P,e_O}$ depends only on the current state of w, so it does not constraint the shape of any LTS, contrary to the definition of $\mathcal{E}_{\mathcal{A}}\left[\!\left[\tau\right]\!\right]_{e_O} w$ which depends on $\mathcal{A}$.

The proof of completeness relies on the following lemmas which state that if two functions are related at a given adequate world, then for any future, applying those functions to related values gives rise to related values. Note that this lemma is wrong when the world is not adequate.

Lemma 44: *Let (v_1, v_2) a pair of ground-closed values of type $\sigma \to \tau$, e_O a span on their name pointers and e_P a span on player name pointers with $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions. Taking $w \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\sigma \to \tau]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$, suppose that $[v_1] \simeq^{\gamma,w}_{e_P,e_O} [v_2]$. Then for all $w_1 \sqsupseteq^* w$ there exists two spans $e^1_P \sqsupseteq e_P$ and $e^1_O \sqsupseteq e_O$, and two substitutions γ^1_i whose domain is $e^1_{P,i}$ and which extend γ_i such that*

- *if σ is atomic, there exists $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$ such that for all $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_{e^1_O} w_2$, $[v_1\ u_1] \simeq^{\gamma^1,w_2}_{e^1_P,e^1_O} [v_2\ u_2]$ and $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e^1_P,e^1_O}(v_1\ u_1, v_2\ u_2)$.*
- *otherwise, there exists $w_2 \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} w_1$ such that $[v_1\ y_1] \simeq^{\gamma^1,w_1}_{e^1_P,e^2_O} [v_2\ y_2]$ and $(w_1, w_1) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e^1_P,e^2_O}(v_1\ y_1, v_2\ y_2)$, where $e^2_O = e^1_O \cdot (y_1, y_2)$.*

Proof Let $w_1 \sqsupseteq w$ and $(h^1_1, h^1_2, \mathcal{D}^1) : w_1$, then from $w \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\sigma \to \tau]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$, we get that there exists $(h_1, h_2, \mathcal{D}) : w$, two spans on name pointers $e^1_P \sqsupseteq e_P$,$e^1_O \sqsupseteq e_O$ and a pair of (incomplete) equivalent traces $U^1_1 \simeq^{\mathcal{D}^1}_{e^1_P,e^1_O} U^1_2$ such that

$$\begin{array}{rcl} \langle v_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle & \xrightarrow{\langle \bar{z_i}\rangle, h_{i|\mathcal{D}_i}} & \langle \Diamond, \gamma'_i, e_{O,i}, h_i, \mathcal{D}_i\rangle \\ & \xrightarrow{U^1_i} & \langle \overrightarrow{K_i}, \gamma^1_i, e^1_{O,i}, h^1_i, \mathcal{D}^1_i\rangle \end{array}$$

(since $h_{i|D_i}$ is closed, with $\gamma'_i = \gamma_i \cdot (z_i \hookrightarrow y_i)$. Then,

- If σ is of ground type, then there exists $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$ such that for all $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}}\left[\!\left[\sigma\right]\!\right]_{e^1_O} w_2$ $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\sigma]\!]^{\gamma^1}_{e^1_P,e^1_O}(v_1\ u_1, v_2\ u_2)$. Moreover, for all $(h^2_1, h^2_2, \mathcal{D}^2) : w_2$ there exists a reduction

$$\langle \overrightarrow{K_i}, \gamma^1_i, e^1_{O,i}, h^1_i, \mathcal{D}^1_i\rangle \xrightarrow{z_i\langle u_i\rangle, h^2_{i|\mathcal{D}^2_i}} \langle (v_i\ u_i) \cdot \overrightarrow{K_i}, \gamma^1_i, e^1_{O,i}, h^2_i, \mathcal{D}^2_i\rangle.$$

Then, we take $U^2_1 \in \left[\langle v_1\ u_1, \gamma^1_1, e^1_{O,1}, h^2_1, \mathcal{D}^2_1\rangle\right]$ s.t.

$$\langle v_1\ u_1 \cdot \overrightarrow{K_1}, \gamma^1_1, e^1_{O,1}, h^2_1, \mathcal{D}^2_1\rangle \xrightarrow{U^2_1} \langle \overrightarrow{K_1}, \gamma^2_1, e^2_{O,1}, h^3_1, \mathcal{D}^3_1\rangle$$

and taking $U^3_1 \in \left[\langle \overrightarrow{K_1}, \gamma^2_1, e^2_{O,1}, h^3_1, \mathcal{D}^3_1\rangle\right]$ we have

$$\underbrace{(\langle \bar{z_1}\rangle, h_{1|\mathcal{D}_1}) \cdot U^1_1 \cdot (z_1\langle u_1\rangle, h^2_{1|\mathcal{D}^2_1}) \cdot U^2_1 \cdot U^3_1}_{T_1} \in [\langle v_1, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1\rangle].$$

So from $[v_1] \simeq^{\gamma,w}_{e_O,e_P} [v_2]$, we get the existence of U_2 such that, defining T_2 as $(\langle \bar{z_2} \rangle, h_{2|\mathcal{D}_2}) \cdot U_2^1 \cdot (z_2 \langle u_2 \rangle, h^2_{2|\mathcal{D}_2^2}) \cdot U_2$, we have
— $T_1 \simeq^{\mathcal{D}}_{e_P,e_O} T_2$
— $T_2 \in [\langle v_2, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2 \rangle]$.
But then, $U_2 \in \left[\langle (v_2\ u_2) \cdot \overrightarrow{K_2}, \gamma_2^1, e^1_{O,2}, h_2^2, \mathcal{D}_2^2 \rangle\right]$, so we can decompose it in $U_2^2 \cdot U_2^3$ such that
— $\langle (v_2\ u_2) \cdot \overrightarrow{K_2}, \gamma_2^1, e^1_{O,2}, h_2^2, \mathcal{D}_2^2 \rangle \xrightarrow{U_2^2} \langle \overrightarrow{K_2}, \gamma_2^2, e_2^2, h_2^3, \mathcal{D}_1^3 \rangle$
— $U_2^2 \in \left[\langle v_2\ u_2, \gamma_2^1, e^1_{O,2}, h_2^2, \mathcal{D}_2^2 \rangle\right]$
— $U_2^3 \in \left[\langle \overrightarrow{K_2}, \gamma_2^2, e_2^2, h_2^3, \mathcal{D}_2^3 \rangle\right]$
Thus, $U_1^2 \simeq^{\mathcal{D}^2}_{e_P^1,e_O^1} U_2^2$, so

$$\left[\langle v_1\ u_1, \gamma_1^1, e^1_{O,1}, h_1^2, \mathcal{D}_1^2 \rangle\right] \simeq^{\mathcal{D}^2}_{e_P^1,e_O^1} \left[\langle v_2\ u_2, \gamma_2^1, e^1_{O,2}, h_2^2, \mathcal{D}_2^2 \rangle\right].$$

Using a symmetric reasoning, we prove the reverse inclusion, so

$$\left[\langle v_1\ u_1, \gamma_1^1, e^1_{O,1}, h_1^2, \mathcal{D}_1^2 \rangle\right] \simeq^{\mathcal{D}^2}_{e_P^1,e_O^1} \left[\langle v_2\ u_2, \gamma_2^1, e^1_{O,2}, h_2^2, \mathcal{D}_2^2 \rangle\right].$$

And we have prove this property for all $(h_1^2, h_2^2, \mathcal{D}^2) : w_2$, so $[v_1\ u_1] \simeq^{\gamma^1,w_2}_{e_P^1,e_O^1} [v_2\ u_2]$.
— Otherwise, there exists $w_2 \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} w_1$ such that, defining $e_O^2 = e_O^1 \cdot (y_1, y_2)$, $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\sigma]\!]^{\gamma^1}_{e_P^1,e_O^2}(v_1\ y_1, v_2\ y_2)$. Moreover, for all $(h_1^2, h_2^2, \mathcal{D}^2) : w_2$ there exists a reduction $\langle \overrightarrow{K_i}, \gamma_i^1, e^1_{O,i}, h_i^1, \mathcal{D}_i^1 \rangle \xrightarrow{z_i \langle y_i \rangle, h^2_{i|\mathcal{D}_i^2}} \langle (v_i\ y_i) \cdot \overrightarrow{K_i}, \gamma_i^1, e^2_{O,i}, h_i^2, \mathcal{D}_i^2 \rangle$.
Then, using the same reasoning as in the ground type case, we get that $[v_1\ y_1] \simeq^{\gamma^1,w_2}_{e_P^1,e_O^2} [v_2\ y_2]$. ∎

Lemma 45: *Let (K_1, K_2) a pair of ground-closed contexts, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions. We consider a world $w \in \bar{\mathcal{K}}_{\mathcal{A}}[\![\sigma, \tau]\!]^{\gamma}_{e_P,e_O}(K_1, K_2)$. Suppose that $[K_1] \simeq^{w,\gamma}_{e_P,e_O} [K_2]$. Then for all $w_1 \sqsupseteq^{*}_{\mathbf{pub}} w$ there exists two spans $e_P^1 \sqsupseteq e_P$ and $e_O^1 \sqsupseteq e_O$, and two substitutions γ_i^1 whose domain is $e^1_{P,i}$ and which extend γ_i such that*
— *if σ is atomic, then there exists $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$ such that for all $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_{e_O^1} w_2$, $[K_1[v_1]] \simeq^{\gamma^1,w_2}_{e_P^1,e_O^1} [K_2[v_2]]$ and $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_P^1,e_O^1}(K_1[v_1], K_2[v_2])$.*
— *otherwise, there exists $w_2 \sqsupseteq^{\mathcal{F}\sigma}_{\mathbf{pub}} w_1$ such that $[K_1[x_1]] \simeq^{\gamma^1,w_2}_{e_P^1,e_O^2} [K_2[x_2]]$ where $e_O^2 = e_O^1 \cdot (x_1, x_2)$, and $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma^1}_{e_P^1,e_O^2}(K_1[x_1], K_2[x_2])$.*

Proof Let $w_1 \sqsupseteq w$ and $(h_1^1, h_2^1, \mathcal{D}^1) : w_1$, then from $w \in \bar{\mathcal{K}}_{\mathcal{A}}[\![\sigma, \tau]\!]^{\gamma}_{e_P,e_O}(K_1, K_2)$, we get that there exists $(h_1, h_2, \mathcal{D}) : w$ and a pair of equivalent traces $U_1^1 \simeq^{\mathcal{D}^1}_{e_P^1,e_O^1} U_2^1$ in $[\langle K_i[\bullet], \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle]$, such that $\langle K_i[\bullet], \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle \xrightarrow{U_i^1} \langle K_i[\bullet], \gamma_i^1, e^1_{O,i}, h_i^1, \mathcal{D}_i^1 \rangle$.
Then we reason by case analysis on σ.

— If σ is of ground type, then there exists $w_2 \sqsupseteq_{\mathbf{pub}} \sigma w_1$ s.t. for all $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_{e_O^1} w_2$, $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}} [\![\tau]\!]^{K_1[v_1]}_{e_O^1, \gamma^1}(K_2[v_2],)$.
Moreover, for all $(h_1^2, h_2^2, \mathcal{D}^2) : w_2$, there exists a reduction

$$\langle K_i[\bullet], \gamma_i^1, e_{O,i}^1, h_i^1, \mathcal{D}_i^1 \rangle \xrightarrow{\langle v_i \rangle, h^2_{i|\mathcal{D}_i^2}} \langle K_i[v_i], \gamma_i^1, e_{O,i}^1, h_i^2, \mathcal{D}_i^2 \rangle$$

So we take $U_1^2 \in \left[\langle K_i[v_1], \gamma_i^1, e_{O,1}^1, h_1^2, \mathcal{D}_1^2 \rangle\right]$, then

$$\underbrace{U_1^1 \cdot (\langle v_1 \rangle, h^2_{1|\mathcal{D}_1^2}) \cdot U_1^2}_{T_1} \in [\langle K_1[\bullet], \gamma_1, e_{O,1}, h_1, \mathcal{D}_1 \rangle]$$

so from $[K_1] \simeq^{\gamma, w}_{e_P, e_O} [K_2]$, we get the existence of $T_2 \in [\langle K_i[\bullet], \gamma_2, e_{O,2}, h_2, \mathcal{D}_2 \rangle]$ s.t. $T_1 \simeq^{\mathcal{D}}_{e_P, e_O} T_2$. Then, due to the determinism of interactive reduction *w.r.t.* a trace, we can write T_2 as $U_2^1 \cdot (\langle v_2 \rangle, h^2_{2|\mathcal{D}_2^2}) \cdot U_2^2$ with $U_2^2 \in \left[\langle K_2[v_2], \gamma_2^1, e_{O,2}^2, h_2^2, \mathcal{D}_2^2 \rangle\right]$ and $U_1^2 \simeq^{\mathcal{D}^2}_{e_P^1, e_O^1} U_2^2$. Thus $\left[\langle K_1[v_1], \gamma_1^1, e_{O,1}^1, h_1^2, \mathcal{D}_1^2 \rangle\right] \precsim^{\mathcal{D}^2}_{e_O^1, e_P^1} \left[\langle K_2[v_2], \gamma_2^1, e_{O,2}^1, h_2^2, \mathcal{D}_2^2 \rangle\right]$. Using a symmetric reasoning, we get that

$$\left[\langle K_1[v_1], \gamma_1^1, e_{O,1}^1, h_1^2, \mathcal{D}_1^2 \rangle\right] \simeq^{\mathcal{D}^2}_{e_O^1, e_P^1} \left[\langle K_2[v_2], \gamma_2^1, e_{O,2}^1, h_2^2, \mathcal{D}_2^2 \rangle\right]$$

Since this equivalence is true for all $(h_1^2, h_2^2, \mathcal{D}^2) : w_2$, we get that $[K_1[v_1]] \simeq^{\gamma^1, w_2}_{e_P^1, e_O^1} [K_2[v_2]]$.

— Otherwise, there exists $w_2 \sqsupseteq_{\mathbf{pub}} \sigma w_1$ s.t. $(w_2, w_2) \in \bar{\mathcal{E}}_{\mathcal{A}} [\![\tau]\!]^{K_1[x_1]}_{e_O^2, \gamma^1}(K_2[x_2],)$, where $e_O^2 = e_O^1 \cdot (x_1, x_2)$. Moreover, for all $(h_1^2, h_2^2, \mathcal{D}^2) : w_2$, there exists a transition

$$\langle K_i[\bullet], \gamma_i^1, e_{O,i}^1, h_i^1, \mathcal{D}_i^1 \rangle \xrightarrow{\langle x_i \rangle, h^1_{i|\mathcal{D}_i^2}} \langle K_i[x_i], \gamma_i^1, e_{O,i}^2, h_i^2, \mathcal{D}_i^2 \rangle$$

Then, following the same reasoning as in the ground case, we get that $[K_1[x_1]] \simeq^{\gamma^1, w_2}_{e_P^1, e_O^2} [K_2[x_2]]$. ∎

Lemma 46: *Let $e_P, e_O, \mathcal{D}$ three spans respectively on player and opponent name pointers and on locations, and γ_1, γ_2 two value substitutions defined respectively on $e_{P,1}$ and $e_{P,2}$. Suppose $[\langle \Diamond, \gamma_1, e_{O,1}, h_1, \mathcal{D}_1 \rangle] \simeq^{\mathcal{D}}_{e_P, e_O} [\langle \Diamond, \gamma_2, e_{O,2}, h_2, \mathcal{D}_2 \rangle]$. Then, for all $(f_1, f_2) \in e_P$ we get that*

$$[\langle \gamma_1(f_1), \gamma_1, \mathcal{I}_1, h_1, \mathcal{D}_1 \rangle] \simeq^{\mathcal{D}}_{e_P, e_O} [\langle \gamma_2(f_2), \gamma_2, e_{O,2}, h_2, \mathcal{D}_2 \rangle]$$

Proof We simply use the fact that $[\langle \gamma_i(f_i), \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle] = (\langle \bar{x_i} \rangle, h_{i|\mathcal{D}_i}) \cdot [\langle \Diamond, \gamma_i', e_{O,i}, h_i, \mathcal{D}_i \rangle]$ with $\gamma_i' = \gamma_i \cdot (x_i \hookrightarrow \gamma_i(f_i))$ and the equivalence $[\langle \Diamond, \gamma_i', e_{O,i}, h_i, \mathcal{D}_i \rangle] \simeq^{\mathcal{D}}_{e_P', e_O} [\langle \Diamond, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i \rangle]$ where $e_P' = e_P \cdot (x_1, x_2)$, due to the fact that $\gamma_i(f_i)$ is by definition already in γ_i. ∎

Lemma 47: *Let (M_1, M_2) a pair of ground-closed terms, e_O a span on their name pointers e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions. We consider a world $(w, w_0) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1, M_2)$ such that $[M_1] \simeq^{\gamma,w}_{e_P,e_O} [M_2]$. Taking $(h_1, h_2, \mathcal{D}) : w$ with $h_i \in \mathbf{Cl}(\nu_L(M_i))$, suppose that $\langle M_i, h_i, \mathcal{D}_i\rangle \to^* \langle E_i, h_i^1, \mathcal{D}_i^1\rangle$. Then there exists $w_1 \sqsupseteq w$ with $(h_1^1, h_2^1, \mathcal{D}^1) : w_1$ such that*

- *either both $E_i = v_i$ with $[v_1] \simeq^{\gamma,w_1}_{e_P,e_O} [v_2]$ and $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$,*
- *or both $E_i = K_i[f_i\, v_i]$ with $(f_1, f_2) \in e_{O,\sigma\to\sigma'}$ and $[v_1] \simeq^{\gamma,w_1}_{e_P,e_O} [v_2]$, $[K_1] \simeq^{\gamma^1,w_1}_{e_P^1,e_O} [K_2]$ with $\gamma^1 = \gamma$ and $e_P^1 = e_P$ if σ is atomic, $\gamma_i^1 = \gamma_i \cdot (x_i \hookrightarrow v_i)$ and $e_P^1 = e_P \cdot (x_1, x_2)$ otherwise. Moreover, $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\sigma]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$ and $(w_1, w_0) \in \bar{\mathcal{K}}_{\mathcal{A}}[\![\sigma', \tau]\!]^{\gamma^1}_{e_P^1,e_O}(K_1, K_2)$,*

Proof The adequacy of w gives us the existence of w_1 such that $w_1 \sqsupseteq w$ and $(h_1^1, h_2^1, \mathcal{D}^1) : w_1$. If both $E_i = v_i$, the adequacy also gives us that $w_1 \sqsupseteq_{\mathbf{pub}} w_0$ and $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\tau]\!]^{e_O}_{\gamma,e_P}(v_1, v_2)$. Using lemma 28, we get that $[\langle M_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle] = [\langle v_i, \gamma_i, e_{O,i}, h_i^1, \mathcal{D}_i^1\rangle]$, so from $[M_1] \simeq^{\gamma,w}_{e_P,e_O} [M_2]$ we get that $[\langle v_1, \gamma_1, e_{O,1}, h_1^1, \rangle] \simeq^{\mathcal{D}^2}_{e_P,e_O} [\langle v_2, \gamma_2, e_{O,2}, h_2^1, \rangle]$.

Then, taking any $(h_1^2, h_2^2, \mathcal{D}^2) : w_1$ s.t. $h_i^2 \in \mathbf{Cl}(\nu_L(v_i, \gamma_i))$, we have $h^2_{i|\overline{\mathcal{D}_i^2}} = h^1_{i|\overline{\mathcal{D}_i^2}}$ and $h^2_{1|\mathcal{D}_1^2} \sim_{\mathcal{D}^2} h^2_{2|\mathcal{D}_2^2}$, so applying Lemma 32, we get that $[v_1] \simeq^{\gamma,w_1}_{e_P,e_O} [v_2]$.

Otherwise, if both $E_i = K_i[f_i\, v_i]$, the adequacy of w gives us $w_1 \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\sigma]\!]^{\gamma'}_{e_P,e_O}(v_1, v_2)$ and $w_1 \in \bar{\mathcal{K}}_{\mathcal{A}}[\![\sigma', \tau]\!]^{\gamma^1}_{e_P',e_O}(K_1, K_2)$. Moreover, $[\langle M_i, \gamma_i, e_{O,i}, h_i, \mathcal{D}_i\rangle]$ is equal to

$$(\bar{f}_i\, \langle u_i\rangle, h^1_{i|\mathcal{D}_i^1}) \cdot \left[\langle K_i[\bullet], \gamma_i^1, e_{O,i}, h_i^1, \mathcal{D}_i^1\rangle\right]$$

so from $[M_1] \simeq^{\gamma,w}_{e_P,e_O} [M_2]$ we get that

$$\left[\langle K_1[\bullet], \gamma_1^1, e_{O,1}, h_1^1, \mathcal{D}_1^1\rangle\right] \simeq^{\mathcal{D}^1}_{e_P,e_O} \left[\langle K_2[\bullet], \gamma_2^1, e_{O,2}, h_2^1, \mathcal{D}_2^1\rangle\right]$$

We also get that $(\bar{f}_1\, \langle u_1\rangle, h^1_{1|\mathcal{D}_1^1}) \sim^{\mathcal{D}^1}_{e_P^1,e_O} (\bar{f}_2\, \langle u_2\rangle, h^1_{2|\mathcal{D}_2^1})$ so $(f_1, f_2) \in e_{\sigma\to\sigma'}$ for some type σ, σ'.

Then, if σ is of ground type, we have $u_i = v_i$, $e_P^1 = e_P$ and $\gamma_i^1 = \gamma_i$. So from $(\bar{f}_1\, \langle v_1\rangle, h^1_{1|\mathcal{D}_1^1}) \sim^{\mathcal{D}^1}_{e_P,e_O} (\bar{f}_2\, \langle v_2\rangle, h^1_{2|\mathcal{D}_2^1})$ we get that

$$\left[\langle v_1, \gamma_1, e_{O,1}, h_1^1, \mathcal{D}_1^1\rangle\right] \simeq^{\mathcal{D}^1}_{e_P,e_O} \left[\langle v_2, \gamma_2, e_{O,2}, h_2^1, \mathcal{D}_2^1\rangle\right]$$

Otherwise, u_i is equal to some fresh pointer name x_i, $e_P^1 = e_P \cdot (x_1, x_2)$ and $\gamma_i^1 = \gamma_i \cdot (x_i \hookrightarrow v_i)$. So using lemma 46, we also get that $[\langle v_1, \gamma_1, e_{O,1}, h_1^1, \mathcal{D}_1^1\rangle] \simeq^{\mathcal{D}^1}_{e_P,e_O} [\langle v_2, \gamma_2, e_{O,2}, h_2^1, \mathcal{D}_2^1\rangle]$.

Finally, using the same reasoning as for the value case, we get that $[v_1] \simeq^{\gamma,w_1}_{e_P,e_O} [v_2]$ and $[K_1] \simeq^{\gamma^1,w_1}_{e_P^1,e_O} [K_2]$. ∎

Then, we can deduce that terms that are equivalent for a given adequate world are in concrete logical relations for that world.

Theorem 39 (Completeness for Values): *Let v_1, v_2 two ground-closed values, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions. We consider a world $w \in \bar{\mathcal{V}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(v_1, v_2)$. If $[v_1] \simeq^{\gamma,w}_{e_P,e_O} [v_2]$ then $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}\,[\![\tau]\!]_{e_O}\, w$.*

Theorem 40 (Completeness for Contexts): *Let K_1, K_2 two ground-closed contexts, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions. We consider $(w, w_0) \in \bar{\mathcal{K}}_{\mathcal{A}}[\![\sigma, \tau]\!]^{\gamma}_{e_P,e_O}(K_1, K_2)$. If $[K_1] \simeq^{\gamma,w}_{e_P,e_O} [K_2]$ then $(K_1, K_2) \in \mathcal{K}_{\mathcal{A}}\,[\![\sigma, \tau]\!]_{e_O}\,(w, w_0)$.*

Theorem 41 (Completeness for Terms): *Let M_1, M_2 two ground-closed terms, e_O a span on their name pointers, e_P a span on player name pointers and $\gamma_i : e_{P,i} \to \mathrm{Val}$ two substitutions. We consider $(w, w_0) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{\gamma}_{e_P,e_O}(M_1, M_2)$. If $[M_1] \simeq^{\gamma,w}_{e_P,e_O} [M_2]$ then $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}}\,[\![\tau]\!]_{e_O}\,(w, w_0)$.*

Proof The proof of the three theorem is done by a mutual induction on types and on the number of ground callbacks of terms considered. It follows directly from Lemma 44 for values, Lemma 45 for contexts and Lemma 47 for terms. ∎

Using the conjunction of this last theorem with the adequacy of exhaustive LTS, we get the completeness of concrete logical relations *w.r.t.* trace semantics.

Corollary 6 (Completeness of Concrete Logical Relations): *Let M_1, M_2 two terms such that $\Sigma; \Gamma \vdash M_1, M_2 : \tau$. Then $[\Sigma; \Gamma_g, \Gamma_f \vdash M_1 : \tau] = [\Sigma; \Gamma_g, \Gamma_f \vdash M_2 : \tau]$ implies $\Sigma; \Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$.*

Proof Let $\gamma_g : \Gamma_g \to \mathrm{Val}$ and $h \in \mathbf{Cl}(\Sigma, \nu_L(\mathrm{codom}(\gamma_g)))\}$. Then using Theorem 29, we get that $[\gamma_g(M_1)] \simeq^{(\varepsilon,\varepsilon),w}_{\varepsilon,\widetilde{\Gamma_f}} [\gamma_g(M_2)]$.

Moreover, defining C_i as $\langle \gamma_g(M_1), \varepsilon, \mathrm{dom}(\Gamma_f), h, D\rangle$ where $D = \mathrm{dom}(h)$, we build a LTS $\mathcal{A} = \mathbf{LtsE}^{L,s}_{e_P,e_O,\mathcal{D}}(C_1, C_2)$ with $L = w$. Then using Theorem 37, we get that $(w, w) \in \bar{\mathcal{E}}_{\mathcal{A}}[\![\tau]\!]^{(\varepsilon,\varepsilon)}_{\varepsilon,\widetilde{\Gamma_f}}(M_1, M_2)$. Finally we conclude from Theorem 41. ∎

6.7 Possible extensions

We have succeeded in building a concrete logical relation defined in a direct way which is sound and complete *w.r.t.* trace semantics. Thus using the correspondence between trace semantics and game semantics, and the fact that game semantics is fully abstract, we get the soundness and completeness of our logical relations *w.r.t.* contextual equivalence, which is the wanted result. This work has been performed for divergent-free terms of GroundML with contexts in RefML. But it seems possible to extend this work to two directions: to terms in RefML, or to contexts in GroundML.

6.7.1 Concrete Logical Relations for full RefML

Trying to define our concrete logical relations for the whole RefML, we would expect to encounter the usual circularity between the definition of worlds and logical relations that we have talked about in the introduction, and which was one of the motivations of our work in Chapter 4 in order to solve it.

But quite unexpectedly, this circularity disappears, since we use a fixed notion of LTS which does not evolve. Moreover, we represent higher-order functions stored in the disclosed part of heaps just with variables, as it is done in trace semantics or game semantics (cf. the end of Section 5.1.7), so that a posteriori it seems fair not to encounter the circularity since it does not appear neither in the game nor the trace model.

However, when working with term of RefML, one cannot reason anymore by induction on the number of ground callbacks of terms, since diverging terms like

$$\lambda \mathtt{f} : (\mathrm{Unit} \to \mathrm{Unit}).\ \mathtt{fix\ g(x)} : (\mathrm{Unit} \to \mathrm{Unit}).(\mathtt{f}();\mathtt{g}())$$

can perform an infinity of such callbacks.

Thus we would need a coinductive definition, which would probably be really closed to the work of Hur et al. in [HDNV12]. But we can also use guarded recursive types to define our logical relations in a coinductive way on the number of ground callback terms can perform, which can possibly be infinite. Let see briefly how to do it.

Firstly, we would need to generalize the definition of $(h_1, h_2, \mathcal{D}) : w$ in two different ways to deal with higher order references. The first one, $\mathbf{P}_e(w)$ only works on functional free heaps:

$$\mathbf{P}_e(w) \stackrel{def}{=} \Big\{(h_1, h_2, \mathcal{D}, e') \mid \mathcal{D} \subseteq \mathcal{D}^w, h_i = h_i^{priv} \cdot h_i^{discl} \text{ s.t. } \mathrm{dom}(h_i^{discl}) = \mathcal{D}_i \text{ and } h_i^{priv} \subseteq h_i^w, h_1^{discl} \sim_{e'}^{\mathcal{D}} h_2^{discl} \text{ and } e'_i = e_i \cup \nu_{\mathbb{P}}^O(h_i^{discl})\Big\}$$

where $w = (h_1^w, h_2^w, \mathcal{D}^w)$. It takes as a parameter a span on name pointers, that is extended to relate those name pointers stored in heaps. In this setting, the usual equivalence on the disclosed part of the heap is easily defined:

$$\begin{array}{rcl} h_1 \sim_{\mathcal{S},e} h_2 & \stackrel{def}{=} & \forall (l_1, l_2) \in \mathcal{S}_{\mathrm{Int}}.\ h_1(l_1) = h_2(l_2) \ \wedge \forall (l_1, l_2) \in \mathcal{S}_{\mathrm{ref}\,\iota}.\ (h_1(l_1), h_2(l_2)) \in \mathcal{S}_\iota \\ & & \forall (l_1, l_2) \in \mathcal{S}_{\tau \to \iota}.(h_1(l_1), h_2(l_2)) \in e \end{array}$$

The second one works on any kinds of heaps, so that it is in fact defined recursively with the logical relation:

$$\mathbf{Q}_e(w) \stackrel{def}{=} \Big\{(h_1, h_2, \mathcal{D}) \mid \mathcal{D} \subseteq \mathcal{D}^w, h_i = h_i^{priv} \cdot h_i^{discl} \text{ s.t. } \mathrm{dom}(h_i^{discl}) = \mathcal{D}_i \text{ and } h_i^{priv} \subseteq h_i^w, \forall (l_1, l_2, \tau) \in \mathcal{D}. \triangleright \Big((h_1(l_1), h_2(l_2)) \in \mathcal{V}_{\mathcal{A}}\, [\![\tau]\!]_e\, w\Big)\Big\}$$

Notice the use of $\triangleright$ to ensure the well-foundness of the definition.

Then, we keep the same definition for logical relations, but for two differences. Firstly, we insert a modality $\triangleright$ in front of every occurrence of $\mathcal{E}[\![\tau]\!]$ in the definition of $\mathcal{V}[\![\tau \to \sigma]\!]$ and $\mathcal{K}[\![\tau, \sigma]\!]$.

Then, one change the definition of $\mathcal{E}[\![\tau]\!]$ so that it uses the two predicates $\mathbf{P}_e(w)$ and $\mathbf{Q}_e(w)$ rather than $(h_1, h_2, \mathcal{D}) : w$.

$$\begin{aligned}\mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e(w, w_0) \stackrel{def}{=} \Big\{ & (M_1, M_2) \mid \forall (h_1, h_2, \mathcal{D}, e') \in \mathbf{P}_e(w).h_i \in \mathbf{Cl}(\nu_L(M_i)) \Rightarrow \\ & ((M_1, h_1) \Uparrow \wedge (M_2, h_2) \Uparrow) \vee \\ & \Big(\exists w' \sqsupseteq w.\exists (h_1', h_2', \mathcal{D}') \in \mathbf{Q}_e'(w').\exists \rho_i : (\nu_L(h_i')\backslash\nu_L(h_i)) \hookrightarrow (\mathrm{Loc}\backslash\nu_L(h_i)). \\ & \quad \langle M_i, h_i, \mathcal{D}_i\rangle \to^* \rho_i \cdot \langle E_i, h_i', \mathcal{D}_i'\rangle \wedge \\ & \quad \Big(E_i = v_i \wedge (v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\tau]\!]_{e'} w' \wedge w' \sqsupseteq^*_{\mathbf{pub}} w_0\Big) \\ & \quad \vee \Big(E_i = K_i[f_i\ v_i] \wedge (f_1, f_2) \in e'_{\sigma\to}\sigma' \wedge (v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_{e'} w' \\ & \qquad \wedge \forall w'' \sqsupseteq^*_{\mathbf{pub}} w'.(K_1, K_2) \in \mathcal{K}_{\mathcal{A}}[\![\sigma', \tau]\!]_{e'}(w'', w_0)\Big)\Big)\Big\}\end{aligned}$$

The idea is that, in input, we get heaps from contexts (*i.e.* Opponent), whose disclosed part is necessarily functional free. So we can simply use $\mathbf{P}_e(w)$. However, while terms are executed, they can modify the disclosed part and store functions in it, or simply create an higher-order reference which is then disclosed. So in exit, the disclosed part of the heap has no reason to be functional free. This means that we have to use $\mathbf{Q}_e(w)$.

Then, the logical relation is defined coinductively on the number of ground callbacks of terms. And this definition is justified by the presence of $\triangleright$ exactly where such ground callbacks are performed.

However, this definition is not complete. Indeed, it does not handle examples like a version of the "deferred divergence" presented in [DNB10], where terms are related while diverging at two different moment of the execution (*i.e.* the synchronization of callbacks has to be weaken). This weakness was already present in our definition of concrete logical relations for terms of GroundML, this explains why it is complete only for *divergence-free* terms.

A way to solve it is to use the idea of inconsistent worlds, as introduced in the same article to solve such equivalences. We left this idea for future work.

6.7.2 Concrete Logical Relations for GroundML

The other direction would be to restrict the power of contexts we have considered. Following this idea, Dreyer et al. [DNB12] give a characterization of such restriction via constraints on the shape of worlds and on the way we can reason on them. Restriction to GroundML then correspond to the possibility to backtrack in the world.

In Chapter 5, we have presented a fully-abstract trace semantics for GroundML, which uses the usual notion of visibility to restrict strategies, as defined in the "semantic cube" by Abramsky.

Then, one can imagine using it to define sound and complete concrete logical relations for GroundML which relates both notions. The idea is to slightly modify the definition of our concrete logical relations so that the way worlds evolve respects the scoped structure of the trace semantics for GroundML.

More precisely, let consider a term $M = \lambda \mathtt{f.let\ x\ =\ ref\ 0\ in\ fv_1; fv_2}$ with $\mathtt{v_1}, \mathtt{v_2}$ two functions which perform effects on x. In GroundML, the callbacks $\mathtt{fv_1}$ and $\mathtt{fv_2}$ are considered in the same way, that is the fact that $\mathtt{fv_1}$ has been performed first does not give him a special status. To integrate this idea, one would simply modify the definition of $\mathcal{K}[\![\tau]\!](w, w_0)$ so that it can backtrack to the initial world w_0.

One could then go further, and consider a language with only integer references, that is without the power of storing locations. Then, location disclosure is not monotonic. This is because the scope of disclosed locations is not global anymore. This should correspond to the availability restriction of names [MT13].

7

Temporal Logical Relations

Having complete direct-style logical relations, one can imagine being able to prove any equivalence with it. However, inspecting its proof of completeness, we see that it relies on a notion of *exhaustive LTS* which is clearly not computable.

However, it is in often possible to define worlds by inspecting terms, trying to find "smart" invariants on heaps. This opens the possibility to *model-check* the membership of two terms M_1, M_2 of type τ to the logical relation $\mathcal{E}[\![\tau]\!]$. This means that, providing a

LTS $\mathcal{A}$ representing the evolution of invariants on heaps for M_1 and M_2, we could *decide* if M_1, M_2 are indeed in $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]\, w$, for w the initial world.

In fact, many choices in the machinery used in Chapter 6 to define $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]\, w$ were chosen for this purpose, the most important one being to work with a fixed LTS $\mathcal{A}$. Indeed, this opens the possibility to work in a simple logic whose semantics is defined via a Kripke model. This idea was already followed by Dreyer et al. in [DNRB10], however their logic was quite complex, both because quantification over λ-terms is allowed, ans because of the low constraints on the variation of worlds in their setting.

Here, we choose to work in a logic which mixes *Presburger Arithmetic*, to define constraints on heaps, and *Temporal Logic*, to reason on the evolution in the LTS, *i.e.* on the control flow. Temporal Logic is of great importance in the field of model checking, to prove safety and liveness properties, after the seminal work of A. Pnuely [Pnu77]. To our knowledge, our framework presented in this chapter is the first one that uses temporal logic to reason on the control flow of an impure functional language, in order to prove equivalence of programs.

We then define *temporal logical relations* in this logic. To do so, we first introduce a notion of *symbolic execution* used to reduce terms with open ground variables. In practice, it simply perform an exhaustive reduction, generating predicates which constrain values of ground variables, such that these predicates are altogether satisfiable if and only if the reduction is indeed operationally possible. Using it, we can define temporal logical relations which correspond exactly to concrete logical relations.

In full generality, the problem of model-checking stays undecidable for our logic, because of two reasons:

— the quantifications over heaps, which can be seen as unbounded size lists,
— the mix between Presburger arithmetic and Temporal Logic.

However, under some reasonable hypothesis on the arity of the transition functions, which allow us to control the quantification over heaps and locations, and supposing that the transitive closure of the LTS is provided and can be stated in Presburger arithmetic, we get a decidable model-checking.

This work has given rise to an implementation in Haskell. This prototype computes the temporal logical relation $\mathbb{E}[\![\tau]\!]\,(M_1, M_2)$ associated to two terms, but is also able to automatically generate an LTS which can possibly validate this formula. Then, it performs a translation of the validity of the temporal formula *w.r.t.* to this LTS to SMT-LIB2, the common language of smt-solver. Using the Z3 solver (`http://z3.codeplex.com/`), we have thus been able to decide various examples from the literature. We left open the possibilities of theoretical results on this LTS generation. If we succeed in proving that the implementation generates an LTS which, under some hypothesis on terms, is *adequate* (as defined in Section 6.5), this would lead to decidability results for fragments of RefML and GroundML.

Such decidability results have already been obtained with *algorithmic game semantics*, which rely on the full abstraction results of game models. In theory, all program equiv-

alences could be proven by simply looking at equality of strategies. However, reasoning on equality of strategies in the pure functional setting is non-trivial (or even undecidable [Loa01]) because models are quotiented with a complex relation. For language with references, there is a good characterization of the quotient, using *complete* strategies, as we have seen in Chapter 5.

Algorithmic game semantics is based on these results, and uses automata theory to represent strategies. In the call-by-name setting, a full classification of the decidability of program equivalence has been given by Ghica and McCusker [GM00] on first-order Idealized Algol. For call-by-value, recent study of a (finitary) ML fragment with bad variables have been developed using variants of pushdown automata [HMO11]. And using variants of *fresh register automata* [Tze11], this method has been extended to "good" references [MT11a, MT12]. The main difficulty in algorithmic game semantics is to encode the pointer structure, that plays carry, ro languages recognizable by automata. To do so, types of considered terms have to be restricted, in order to get decidability results. It would then be interesting to compare these results to the one one could obtained with our method, which would rely on the control of the creation of locations rather than a restriction on types.

Plan of the Chapter In Section 7.1, we define our logic HeapTL, a mix between Presburger arithmetic and temporal logic. It uses LTS introduced in Chapter 6 to define the semantics of temporal connectives, and worlds for the semantics of the predicates on heaps. Symbolic Execution is introduced in Section 7.2, which is crucial in the possibility of automatically generated the temporal logical relation associated to two terms, as defined in Section 7.3. Then, we study in Section 7.5 model checking properties of the temporal logical relations, which has been sketched in the introduction. These results rely on the soundness and completeness of the temporal logical relations (*w.r.t.* concrete logical relations), which are proven respectively in Section 7.6 and 7.7.

7.1 Temporal Logic for Heaps

We first begin by defining our temporal logic for heaps, named HeapTL, used to afterwards to define our temporal logical relations. HeapTL is a first-order sorted logic with ground sorts θ ranging over Int, Loc and Heap. Unusually for a temporal logic, it is based on top of Presburger arithmetic. It contains basic arithmetic propositions, defined as:

— Arithmetic terms: $t, t' \stackrel{def}{=} x \mid n \mid t + t' \mid -t$ with $x \in \text{Var}$ and $n \in \text{Int}$

— Arithmetic propositions: $P \stackrel{def}{=} t = t' \mid t < t'$

We also consider *heap terms* u, u', which are either locations or variables.

$$
\begin{array}{lcl}
(w,\kappa,n) \models_{\mathcal{A}} u \hookrightarrow_i u' & \stackrel{def}{=} & h_i(u) = u' \vee u \in \mathcal{S}_i \\
(w,\kappa,n) \models_{\mathcal{A}} \mathbf{X}\psi & \stackrel{def}{=} & \exists w' \sqsupseteq w.(w',\kappa',n+1) \models_{\mathcal{A}} \psi \\
(w,\kappa,n) \models_{\mathcal{A}} \mathbf{X}^{\mathcal{F}_\iota}_{\mathbf{pub}}\psi & \stackrel{def}{=} & \exists w' \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w.(w',\kappa',n+1) \models_{\mathcal{A}} \psi \\
(w,\kappa,n) \models_{\mathcal{A}} \Box\psi & \stackrel{def}{=} & \forall w' \sqsupseteq^{*} w.(w',\kappa',n+1) \models_{\mathcal{A}} \psi \\
(w,\kappa,n) \models_{\mathcal{A}} \Box_{\mathbf{pub}}\psi & \stackrel{def}{=} & \forall w' \sqsupseteq^{*}_{\mathbf{pub}} w.(w',\kappa',n+1) \models_{\mathcal{A}} \psi \\
(w,\kappa,n) \models_{\mathcal{A}} \mathbf{P}_{\mathbf{pub}}(n) & \stackrel{def}{=} & w \sqsupseteq_{\mathbf{pub}} \kappa(n) \\
(w,\kappa,n) \models_{\mathcal{A}} \mathrm{Discl}_\tau(l_1,l_2) & \stackrel{def}{=} & (l_1,l_2) \in \mathcal{S}_\tau \\
(w,\kappa,n) \models_{\mathcal{A}} h_1 \overset{\mathcal{D}}{\sim} h_2 & \stackrel{def}{=} & h_1 \sim_{\mathcal{S}} h_2 \\
(w,\kappa,n) \models_{\mathcal{A}} \mathcal{H}\, x.\varphi & \stackrel{def}{=} & (w,\kappa,n) \models_{\mathcal{A}} \varphi\,\{n/x\} \\
(w,\kappa,n) \models_{\mathcal{A}} \mathcal{N}_i\, x.\varphi & \stackrel{def}{=} & \exists y \in \mathrm{Loc} \setminus (\mathrm{FHT}_i(\varphi) \cup \mathcal{S}_i).(w,\kappa,n) \models_{\mathcal{A}} \varphi\,\{y/x\}
\end{array}
$$

where $w = (s, h_1, h_2, \mathcal{S})$ and $\kappa' = \kappa \cdot (n \hookrightarrow (s, h_1, h_2))$

Figure 7.1: Kripke Semantics of the Temporal Symbols

HeapTL follows the design of Computational Tree Logic (CTL) with the additional possibility to talk about histories of paths. The grammar of temporal formulas is given by

$$
\begin{array}{rcl}
\psi, \psi' & \stackrel{def}{=} & P \mid \neg\psi \mid \psi \wedge \psi' \mid \psi \Rightarrow \psi' \mid \forall x \in \theta.\psi \mid u \hookrightarrow_i u' \mathbf{X}\psi \mid \mathbf{X}^{\mathcal{F}_\iota}_{\mathbf{pub}}\psi \mid \Box\psi \mid \Box_{\mathbf{pub}}\psi \mid \\
& & \mathbf{P}_{\mathbf{pub}}(n) \mid \mathrm{Discl}_\tau(u_1,u_2) \mid h_1 \overset{\mathcal{D}}{\sim} h_2 \mid \mathcal{H}\, x.\varphi \mid \mathcal{N}_i\, x.\varphi
\end{array}
$$

A world with history W is then defined as a triple (w, κ, n) where w is a world, κ is a function from integers to World and n is an integer representing a counter to the current history. This history is used to force the existence of public transition between states that represent initial calls (opponent answers) and states where a value is returned.

The meaning of HeapTL is given in Figure 7.1 via a Kripke semantics, defined as a validity relation $W \models_{\mathcal{A}} \psi$, indexed by a LTS $\mathcal{A}$ that translates formulas of HeapTL to second-order logic.

The validity of logical connectives is standard and not detailed. When the sort of a variable is clear from the context, we omit its type.

The definition of $W \models_{\mathcal{A}} u \hookrightarrow_i u'$ says that u points to u' in the current heap h_i of the world. Note that it is possible that u does not belong to the domain of h_i as long as it is a disclosed location.

HeapTL is composed of four temporal modalities:

— $\mathbf{X}\psi$ which forces the existence a private future of the current state and which satisfies ψ.

— $\mathbf{X}^{\mathcal{F}_\iota}_{\mathbf{pub}}\psi$, which forces the existence of a public future of the current state, that satisfies ψ, and that extends the set of disclosed locations with fresh ones with respect to a ground type ι.

— $\Box\psi$ (resp. $\Box_{\mathbf{pub}}\psi$) which forces ψ to be true in all private (resp. public) future states of the current one.

The predicate $\mathbf{P}_{\mathbf{pub}}(n)$ forces the existence of a public transition between the current state and the state appearing at position n in the history.

To specify the disclosed part of the heap, the predicate $\mathrm{Discl}_\tau(u_1, u_2)$ imposes u_1 and u_2 to be related by using the span on locations of the world. In the same way, $h_1 \overset{\mathcal{D}}{\sim} h_2$ expresses the equivalence between the disclosed part of heaps h_1 and h_2 by simply using the equivalence of the current span.

Finally, HeapTL contains three specific binder:

— $\mathcal{H}\, x.\varphi$, which substitute x with the current history counter in φ,

— $\mathcal{N}_1\, y.\varphi$ (resp. $\mathcal{N}_2\, y.\varphi$) that provides a fresh existential quantification on locations, that is on locations that do not appear free in φ. We define $\mathrm{FHT}_i(\psi)$ to be the free heap terms of ψ appearing in predicates $u \hookrightarrow_i u'$ of ψ, as
  — $\mathrm{FHT}_i(u \hookrightarrow_i u') = \{u\}$ and $\mathrm{FHT}_i(u \hookrightarrow_j u') = \varnothing$ when $i \neq j$,
  — $\mathrm{FHT}_i(\mathcal{N}_i\, x.\psi) = \mathrm{FHT}_i(\psi)\backslash\{x\}$,
  — $\mathrm{FHT}_i(\psi \wedge \psi') = \mathrm{FHT}_i(\psi) \wedge \mathrm{FHT}_i(\psi')$,
  the rest of the definition going through the structure of ψ. Notice that $(w, \kappa, n) \models_{\mathcal{A}} \mathcal{N}_i\, x.\varphi$ could have been defined as $\exists y \in \mathrm{Loc} \setminus (\nu_L(h_i) \cup \mathcal{S}_i)$, but this would have forbidden the possibility for worlds to reuse an undisclosed location which are no more present in the term.

7.2 Symbolic Execution

The definition of temporal logical relations will not be based on the operational semantics of the language but rather on a symbolic version of it. Symbolic execution uses a *symbolic heap* $\mathbb{H}$, defined as a partial function from heap terms u (i.e. locations or variables) to heap terms or arithmetic terms. Arithmetic is used to force conditions on (symbolic) variables used to represent values pointed to by locations and variables. For example, the heap defined as $[x \hookrightarrow l] \cdot [l \hookrightarrow 3]$ can be represented symbolically as $[x \hookrightarrow x_1] \cdot [x_1 \hookrightarrow x_2]$ plus the arithmetic constraints $x_1 = l \wedge x_2 = 3$.

Figure 7.2 defines the one-step symbolic execution of a triple $(M, \mathbb{H}, \mathbb{C})$ where M is a term, $\mathbb{H}$ a symbolic heap and $\mathbb{C}$ a list of arithmetic and heap constraints. This execution is non-deterministic and generates all possible terms and symbolic heaps that can be obtained from M with a heap satisfying $(\mathbb{H}, \mathbb{C})$. We also use the notation $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (M', \mathbb{H}', \mathbb{C}')$ to represent the fact that $(M', \mathbb{H}', \mathbb{C}')$ is in the list of the redexes of $(M, \mathbb{H}, \mathbb{C})$. When the set of constraints is empty, we simply write $(M, \mathbb{H})$ rather than $(M, \mathbb{H}, \varnothing)$.

The symbolic reduction of $(K[u := v], \mathbb{H}, \mathbb{C})$ is the more evolved rule as it deals with aliasing of locations. That is, it generates all executions for each possible aliasing of u to a location occurring in the set $\mathcal{L}$.

Finally, we define $\Downarrow (M, \mathbb{H}, \mathbb{C})$ as the set of irreducible $(M', \mathbb{H}', \mathbb{C}')$ such that $(M, \mathbb{H}, \mathbb{C}) \mapsto_s^* (M', \mathbb{H}', \mathbb{C}')$. We will only consider such configurations for $\mathbb{H}$ *closed* for M, that is:

$$
\begin{array}{lcll}
(K[M],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[M'],\mathbb{H},\mathbb{C}) \text{ when } (M,\mathbb{H}) \mapsto (M',\mathbb{H}) & \\
(K[x == x'],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[\mathbf{true}],\mathbb{H},\mathbb{C} \wedge (x = x')) \quad \text{or} \quad (K[\mathbf{false}],\mathbb{H},\mathbb{C} \wedge (x \neq x')) & \\
(K[x == \widehat{n}],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[\mathbf{true}],\mathbb{H},\mathbb{C} \wedge (x = n)) \odot (K[\mathbf{false}],\mathbb{H},\mathbb{C} \wedge (x \neq n) & \\
(K[\widehat{n} == x],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[\mathbf{true}],\mathbb{H},\mathbb{C} \wedge (x = n)) \odot (K[\mathbf{false}],\mathbb{H},\mathbb{C} \wedge (x \neq n) & \\
(K[x == l],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[\mathbf{true}],\mathbb{H},\mathbb{C} \wedge (x = l)) \odot (K[\mathbf{false}],\mathbb{H},\mathbb{C} \wedge (x \neq l) & \\
(K[l == x],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[\mathbf{true}],\mathbb{H},\mathbb{C} \wedge (x = l)) \odot (K[\mathbf{false}],\mathbb{H},\mathbb{C} \wedge (x \neq l) & \\
(K[x + \widehat{n}],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[z],\mathbb{H},\mathbb{C} \wedge (z = x + n)) \text{ with } z \text{ fresh in } \mathbb{C} & \\
(K[\widehat{n} + x],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[z],\mathbb{H},\mathbb{C} \wedge (z = x + n)) \text{ with } z \text{ fresh in } \mathbb{C} & \\
(K[x + x'],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[z],\mathbb{H},\mathbb{C} \wedge (z = x + x')) \text{ with } z \text{ fresh in } \mathbb{C} & \\
(K[\texttt{if } b \texttt{ then } M_1 \texttt{ else } M_2],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[M_1],\mathbb{H},\mathbb{C} \wedge (b = \mathbf{true})) & \\
 & & \quad \text{or} \quad (K[M_2],\mathbb{H},\mathbb{C} \wedge (b = \mathbf{false})) & \\
(K[!u],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[\mathbb{H}(u)],\mathbb{H},\mathbb{C}) & \text{if } \mathbb{H}(u) \text{ is defined} \\
(K[u := v],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[()],\mathbb{H}_{|\mathrm{dom}(\mathbb{H})\setminus\mathrm{dom}(\mathcal{L})} \cdot \overrightarrow{u_i \hookrightarrow v},\mathbb{C} \wedge \overrightarrow{u_i = u} \wedge \overrightarrow{u'_j \neq u}) & \\
\multicolumn{4}{l}{\text{with } \mathcal{L} \subseteq \mathrm{dom}(\mathbb{H}), u \in \mathcal{L} \text{ and } \overrightarrow{u_i} \text{ range over } \mathcal{L}, \overrightarrow{u_j}' \text{ range over } \mathrm{dom}(\mathbb{H}) \setminus \mathrm{dom}(\mathcal{L}).} \\
(K[\mathrm{ref}\, v],\mathbb{H},\mathbb{C}) & \mapsto_s & (K[y],\mathbb{H} \cdot (y \hookrightarrow v),\mathbb{C}) & \text{with } y \text{ fresh}
\end{array}
$$

Figure 7.2: Definition of Symbolic Execution

— for all u a free location or a free location variable of M, $\mathbb{H}(l)$ is defined,
— for all $u \in \mathrm{dom}(\mathbb{H})$, if $\mathbb{H}(u) = u'$ is a heap term, then $\mathbb{H}(u')$ is defined.

We now state various lemmas on the symbolic reduction which will be useful to relate it to the operational reduction.

Let $A \subseteq \mathrm{Var}$, we say that a substitution $\alpha : A \to \mathrm{Val}$ is not aliased on a subset $B \subseteq A$ of locations variables iff for all $b \in B$, $a \in A$ s.t. $a \neq b$, $\alpha(a) \neq \alpha(b)$.

Lemma 48: *Let M a ground closed term, and $\mathbb{H}$ a closed symbolic heap for M. Then if $(M,\mathbb{H}^{pre},\mathbb{C}) \mapsto_s^* (M',\mathbb{H}^{post},\mathbb{C})$, $\mathbb{H}^{post}$ is closed for M'.*

Proof By induction on the length of $(M,\mathbb{H}^{pre},\mathbb{C}) \mapsto_s^* (M',\mathbb{H}^{post},\mathbb{C})$, analyzing the reduction taken. If $M = K[\mathrm{ref}\, v]$, then $(M,\mathbb{H}^{pre},\mathbb{C}) \mapsto_s (K[y],\mathbb{H} \cdot [y \hookrightarrow v])$, thus $\mathbb{H}'(y)$ is defined, and if v is a heap term, $\mathbb{H}'(v) = \mathbb{H}(v)$ is also defined since $\mathbb{H}$ is closed.

If $M = K[u := v]$ then the resulting heap $\mathbb{H}'$ is closed because if v is a heap term, then $\mathbb{H}'(v) = \mathbb{H}(v)$ is defined since $\mathbb{H}$ is closed. ∎

Lemma 49: *Let M a ground closed term, and $\mathbb{H}$ a closed symbolic heap for M. Then for all $(M',\mathbb{H}^{post},\mathbb{C}') \in \Downarrow (M,\mathbb{H}^{pre},\mathbb{C})$, M' is either a value v or a* callback *$K[f\, v]$ with f a free functional variable of f.*

Proof $(M',\mathbb{H}^{post},\mathbb{C}')$ is irreducible, so if it is neither a value or a callback, it is equal to $K[!u]$ with $\mathbb{H}(u)$ undefined. But this case is impossible since, by the previous lemma, $\mathbb{H}^{post}$ is closed for M' so $\mathbb{H}(u)$ is defined. ∎

Lemma 50: *Let M a ground closed term and $\mathbb{H}^{pre}$ a symbolic heap for it. Suppose that $(M, \mathbb{H}^{pre}) \mapsto_s (M', \mathbb{H}^{post}, \mathbb{C})$, and $\alpha^{pre} : \mathrm{FV}(\mathbb{H}^{pre}) \to \mathrm{Val}$ a substitution. Then there exists a substitution $\alpha^{post} : (\mathrm{FV}(\mathbb{H}^{post}, \mathbb{C}) \backslash \mathrm{FV}(\mathbb{H}^{pre})) \to \mathrm{Val}$ s.t. $\alpha^{pre} \cdot \alpha^{post}$ is not aliased on $\mathrm{dom}(\mathbb{H}^{post}) \backslash \mathrm{dom}(\mathbb{H}^{pre})$ and $\alpha(\mathbb{C})$ is valid.*

Finally, we state the two following lemmas which make the link between operational and symbolic reductions.

Lemma 51: *Let $(M, \mathbb{H}, \mathbb{C})$ a symbolic configuration, and $\alpha : \mathrm{FGV}(M, \mathbb{H}, \mathbb{C}) \to \mathrm{Val}$ a substitution on its free ground variables and $\alpha(\mathbb{C})$ is valid. Consider $h : \alpha(\mathrm{dom}(\mathbb{H})) \to \mathrm{Val}$ a heap s.t. $\alpha(\mathbb{H}(x)) = h(\alpha(x))$ for all $x \in \mathrm{dom}(\mathbb{H})$. Then if $(\alpha(M), h) \mapsto (N, h')$, there exists a symbolic configuration $(M', \mathbb{H}', \mathbb{C}')$ and $\alpha' : \mathrm{FGV}(M', \mathbb{H}', \mathbb{C}')$ a substitution on its free ground variables such that*

- $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (M', \mathbb{H}', \mathbb{C}')$,
- $\alpha'_{|\mathrm{FGV}(M, \mathbb{H}, \mathbb{C})} = \alpha$,
- *α' is not aliased on* $\mathrm{dom}(\mathbb{H}') \backslash \mathrm{dom}(\mathbb{H})$,
- *$\alpha'(C')$ is valid,*
- $\alpha'(M') = N$,
- $\alpha'(\mathbb{H}'(u)) = h'(\alpha'(u))$ *for all* $u \in \mathrm{dom}(\mathbb{H}')$.

Proof By case analysis on $(\alpha(M), h) \mapsto (N, h')$.

- If $(M, h) \mapsto (N, h)$ then we simply take $\mathbb{H}' = \mathbb{H}$, $\mathbb{C}' = \mathbb{C}$ and $\alpha' = \alpha$.
- If $M = K[x+y]$ then $N = \alpha(K)[\widehat{\alpha(x) + \alpha}(y)]$ and $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (K[z], \mathbb{H}, \mathbb{C} \wedge (z = x + y))$ with z fresh. Thus, defining α' as the extension of α with $z \hookrightarrow \widehat{x} + \widehat{y}$, we have the wanted properties.
- If $M = K[!u]$, then $N = \alpha(K)[h(\alpha(u))]$ and $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (K[\mathbb{H}(u)], \mathbb{H}, \mathbb{C})$ and we conclude using the equality $\alpha(\mathbb{H}(u)) = h(\alpha(u))$.
- If $M = K[\mathrm{ref}\, v]$ then $N = \alpha(K)[l]$ and $h' = h \cdot (l \hookrightarrow \alpha(v))$ with $l \notin \mathrm{dom}(h)$. We have $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (K[y], \mathbb{H} \cdot (y \hookrightarrow v), \mathbb{C})$, so taking $\alpha' = \alpha \cdot (y \hookrightarrow l)$, we get that $\alpha'(\mathbb{H}'(y)) = \alpha(v)$ so using the fact that $h(\alpha(y)) = \alpha(v)$ we get $\alpha'(\mathbb{H}'(y)) = h'(\alpha'(y))$. Moreover, since $l \notin \mathrm{dom}(h) = \alpha(\mathrm{dom}(\mathbb{H}))$, we get that α' is not aliased on $\mathrm{dom}(\mathbb{H}') \backslash \mathrm{dom}(H)$.
- If $M = K[u := v]$, then $N = \alpha(K)[()]$ and $h' = h[\alpha(u) \hookrightarrow \alpha(v)]$. Let consider $\mathcal{L} = \{u' \mid \alpha(u') = \alpha(u)\}$. Then $(K[u := v], \mathbb{H}, \mathbb{C}) \mapsto_s (K[()], \mathbb{H}', \mathbb{C}')$ with $\mathbb{H}' = \mathbb{H}_{|\mathrm{dom}(\mathbb{H}) \backslash \mathrm{dom}(\mathcal{L})} \cdot \overrightarrow{u_i \hookrightarrow v}$ and $\mathbb{C}' = \mathbb{C} \wedge \overrightarrow{u_i = u} \wedge \overrightarrow{u'_j \neq u}$) with $\overrightarrow{u_i}$ ranges over $\mathcal{L}$ and $\overrightarrow{u'_j}$ ranges over $\mathrm{dom}(\mathbb{H}) \setminus \mathrm{dom}(\mathcal{L})$. By definition of $\mathcal{L}$, $\alpha(u) = \alpha(u')$ for all $u' \in \mathcal{L}$ and that $\alpha(u) \neq \alpha(u')$ for all $u' \notin \mathcal{L}$. Thus, $\alpha(\mathbb{C}')$ is valid. Moreover, taking $u' \in \mathcal{L}$, we have $h'(\alpha(u')) = h'(\alpha(u)) = \alpha(v)$ and $\alpha(\mathbb{H}'(u')) = \alpha(v)$, so the wanted property on heaps is true.

Lemma 52: *Let $(M, \mathbb{H}, \mathbb{C})$ a symbolic configuration s.t. $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (M', \mathbb{H}', \mathbb{C}')$, and $\alpha : \mathrm{FGV}(M, M', \mathbb{H}', \mathbb{C}') \to \mathrm{Val}$ a substitution on its free ground variables s.t. α is not*

aliased on $\mathrm{dom}(\mathbb{H}')\backslash\mathrm{dom}(\mathbb{H})$ *and* $\alpha(\mathbb{C}')$ *is valid. Consider* $h : \alpha(\mathrm{dom}(\mathbb{H})) \to \mathrm{Val}$ *a heap s.t.* $\alpha(H(\alpha(u))) = h(\alpha(u))$ *for all* $u \in \mathrm{dom}(\mathbb{H})$. *Then there exists an injection* $\rho : \alpha(\mathrm{dom}(\mathbb{H}^{post})\backslash\mathrm{dom}(\mathbb{H}^{pre})) \hookrightarrow \mathrm{Loc}$ *such that :*

— $(\alpha(M), h) \mapsto (\rho(\alpha(M')), h')$
— $\alpha((\mathbb{H}'(u))) = h'(\rho(\alpha(u)))$ *for all* $u \in \mathrm{dom}(\mathbb{H}')$.

Proof By case analysis on $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (M', \mathbb{H}', \mathbb{C}')$:

— If $(M, \mathbb{H}) \mapsto (M, \mathbb{H})$, then $(\alpha(M'), \mathbb{H}) \mapsto (\alpha(M), \mathbb{H})$.
— If $M = K[x + y]$ then $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (K[z], \mathbb{H}, \mathbb{C} \wedge z = x + y])$ with z fresh. Then, from the fact that $\alpha(z) = \alpha(x) + \alpha(y)$ is valid, we have $(\alpha(M), h) \mapsto (\alpha(M'), h)$.
— If $M = K[!u]$, $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (K[\mathbb{H}(u)], \mathbb{H}, \mathbb{C}])$. Then $(\alpha(M), h) \mapsto (\alpha(K)[h(\alpha(u)], h)$, and we conclude using the equality $\alpha(\mathbb{H}(\alpha(u))) = h(\alpha(u))$.
— If $M = K[\mathrm{ref}\, v]$ then $(M, \mathbb{H}, \mathbb{C}) \mapsto_s (K[y], \mathbb{H} \cdot (y \hookrightarrow v), \mathbb{C})$ with y fresh. Moreover, $(\alpha(M), h) \mapsto (\alpha(K)[l], h \cdot [l \hookrightarrow \alpha(v)])$ with $l \notin \mathrm{dom}(h)$. Then, defining ρ as $\alpha(y) \hookrightarrow l$, we get that $\rho(\alpha(K[y])) = \alpha(K)[\rho(\alpha(y))]$ since α is not aliased on $\mathrm{FLV}(\mathbb{H}')$, so $\alpha(y) \notin \nu_L(K)$ since y is fresh in K. Then, $\alpha(K)[\rho(\alpha(y)]$ is indeed equal to $\alpha(K)[l]$. Moreover, $\alpha(\mathbb{H}'(y)) = \alpha(v)$ and $h'(\rho(\alpha(y))) = h'(l) = \alpha(v)$.
— If $M = K[u := v]$, from the fact that $\alpha(\mathbb{C})$ is valid, we get that the only set containing all the locations aliased to u is $\mathcal{L} = \{u' \mid \alpha(u) = \alpha(u')\}$. Then $(K[u := v], \mathbb{H}, \mathbb{C}) \mapsto_s (K[()], \mathbb{H}', \mathbb{C}')$ with $\mathbb{H}' = \mathbb{H}_{|\mathrm{dom}(\mathbb{H})\backslash\mathrm{dom}(\mathcal{L})} \cdot \overrightarrow{u_i \hookrightarrow v}$ and $\mathbb{C}' = \mathbb{C} \wedge \overrightarrow{u_i = u} \wedge \overrightarrow{u'_j \neq u})$ with $\overrightarrow{u_i}$ ranges over $\mathcal{L}$ and $\overrightarrow{u_j}'$ ranges over $\mathrm{dom}(\mathbb{H}) \setminus \mathrm{dom}(\mathcal{L})$. Thus, $(\alpha(M), h) \mapsto (\alpha(K)[()], h[\alpha(u) \mapsto \alpha(v)])$ and for all $u' \in \mathcal{L}$, $h'(\alpha(u)) = \alpha(v)$ and $\alpha(\mathbb{H}'(u)) = \alpha(v)$. ∎

7.3 Temporal Logical Relations

Using the symbolic execution, we define in Figure 7.3 temporal logical relations as formulas over HeapTL. The reader can check that it follows the very same pattern as the definition of concrete logical relations, where interactive reduction has been replaced by the symbolic execution, where concrete reasoning on location becomes abstract, and where constraints on the structure of public transitions are now explicit.

More precisely, the universal quantifications over worlds are replaced by modalities $\Box$ and $\Box_{\mathbf{pub}}$, while the existential ones are replaced by $\mathbf{X}$ and $\mathbf{X}^{\mathcal{F}\tau}_{\mathbf{pub}}$. Thus, the definition of the temporal logical relation is abstract over both LTS and worlds.

As usual for logical relations, the construction is given by a mutual induction between formulas for terms, values, and contexts. The well-foundedness of the construction is guarantee in our case by an analysis on types and on the number of "ground-callbacks", which are the configurations on which, with the values, the symbolic execution stopped.

The definition of $\mathbb{E}[\![\tau]\!]^n_e$ uses an auxiliary predicate $\mathbb{R}\mathrm{el}_{e,\tau}$, in which we check that $\mathbb{C}_1 \wedge \mathbb{C}_2$ to ensure that the two symbolic reductions are consistent, i.e. they correspond to concrete feasible reductions.

$$\mathbb{R}\mathrm{el}^n_{e,\tau}((v_1,\mathbb{H}_1^{post},\mathbb{C}_1),(v_2,\mathbb{H}_2^{post},\mathbb{C}_2)) \stackrel{def}{=} \mathcal{N}_1\,\overrightarrow{y_1}.\mathcal{N}_2\,\overrightarrow{y_2}.\forall\overrightarrow{x'}.\ \mathbf{X}\Big((\mathbb{C}_1\wedge\mathbb{C}_2)\Rightarrow\uparrow_1(\mathbb{H}_1^{post})\wedge\uparrow_2(\mathbb{H}_2^{post}) \wedge(\mathbb{H}_1^{post}\overset{\mathcal{D}}{\sim}\mathbb{H}_2^{post}\wedge\mathbb{V}\,[\![\tau]\!]_e\,(v_1,v_2)\wedge\mathbf{P}_{\mathbf{pub}}(n))\Big)$$

$$\mathbb{R}\mathrm{el}^n_{e,\tau}((K_1[f_1\,v_1],\mathbb{H}_1^{post},\mathbb{C}_1),\ (K_2[f_2\,v_2],\mathbb{H}_2^{post},\mathbb{C}_2)) \stackrel{def}{=} \mathcal{N}_1\,\overrightarrow{y_1}.\mathcal{N}_2\,\overrightarrow{y_2}.\forall\overrightarrow{x'}\ \mathbf{X}\Big((\mathbb{C}_1\wedge\mathbb{C}_2)\Rightarrow(\uparrow_1(\mathbb{H}_1^{post})\wedge\uparrow_2(\mathbb{H}_2^{post}) \wedge\mathbb{H}_1^{post}\overset{\mathcal{D}}{\sim}\mathbb{H}_2^{post}\wedge\mathbb{V}\,[\![\sigma]\!]_e\,(v_1,v_2)\wedge\ \square_{\mathbf{pub}}\mathbb{K}\,[\![\sigma',\tau]\!]^n_e\,(K_1,K_2))$$
$$(\text{when } (f_1,f_2)\in e_{\sigma\to\sigma'})$$

$$\mathbb{R}\mathrm{el}^n_{e,\tau}((\_,\_,\mathbb{C}_1),(\_,\_,\mathbb{C}_2)) \stackrel{def}{=} \mathcal{N}_1\,\overrightarrow{y_1}.\mathcal{N}_2\,\overrightarrow{y_2}.\forall\overrightarrow{x'}\ \neg(\mathbb{C}_1\wedge\mathbb{C}_2) \qquad (\text{otherwise})$$

$$\mathbb{V}\,[\![\mathrm{Unit}]\!]_e\,(v_1,v_2) \stackrel{def}{=} v_1=v_2$$
$$\mathbb{V}\,[\![\mathrm{Bool}]\!]_e\,(v_1,v_2) \stackrel{def}{=} v_1=v_2$$
$$\mathbb{V}\,[\![\mathrm{Int}]\!]_e\,(v_1,v_2) \stackrel{def}{=} v_1=v_2$$
$$\mathbb{V}\,[\![\mathrm{ref}\,\iota]\!]_e\,(l_1,l_2) \stackrel{def}{=} \mathrm{Discl}_\iota(l_1,l_2)$$

$$\mathbb{V}\,[\![\iota\to\tau]\!]_e\,(v_1,v_2) \stackrel{def}{=} \square\mathbf{X}^{\mathcal{F}_\iota}_{\mathbf{pub}}(\forall u_1,u_2:\iota.\mathbb{V}\,[\![\iota]\!]_e\,(u_1,u_2)\Rightarrow \mathcal{H}\,y.\mathbb{E}\,[\![\tau]\!]^y_e\,(v_1\ u_1,v_2\ u_2))$$
$$\mathbb{V}\,[\![\tau\to\sigma]\!]_e\,(v_1,v_2) \stackrel{def}{=} \square\mathbf{X}^{\mathcal{F}_\tau}_{\mathbf{pub}}(\mathcal{H}\,y.\mathbb{E}\,[\![\sigma]\!]^y_{e\cdot(y_1,y_2,\tau)}\,(v_1\ y_1,v_2\ y_2))$$
$$(\tau \text{ not ground }, y_1\notin e_1, y_2\notin e_2)$$

$$\mathbb{K}\,[\![\iota,\sigma]\!]^n_e\,(K_1,K_2) \stackrel{def}{=} \mathbf{X}^{\mathcal{F}_\iota}_{\mathbf{pub}}(\forall v_1,v_2:\iota.\mathbb{V}\,[\![\iota]\!]_e\,(v_1,v_2)\Rightarrow\mathbb{E}\,[\![\sigma]\!]^n_e\,(K_1[v_1],K_2[v_2]))$$
$$\mathbb{K}\,[\![\tau,\sigma]\!]^n_e\,(K_1,K_2) \stackrel{def}{=} \mathbf{X}^{\mathcal{F}_\tau}_{\mathbf{pub}}(\mathbb{E}\,[\![\sigma]\!]^n_{e\cdot(x_1,x_2,\tau)}\,(K_1[x_1],K_2[x_2]))$$
$$(\tau \text{ not ground }, x_1\notin e_1, x_2\notin e_2)$$

$$\mathbb{E}\,[\![\tau]\!]^n_e\,(M_1,M_2) \stackrel{def}{=} \forall\vec{x}.\bigwedge_{\mathbb{S}_i\in\Downarrow(M_i,\mathbb{H}_i^{pre})}(\uparrow_1(\mathbb{H}_1^{pre})\wedge\uparrow_2(\mathbb{H}_2^{pre})\wedge\mathbb{H}_1^{pre}\overset{\mathcal{D}}{\sim}\mathbb{H}_2^{pre})\Longrightarrow\Big(\mathbb{R}\mathrm{el}^n_{e,\tau}(\mathbb{S}_1,\mathbb{S}_2)\Big)$$

$$\Sigma;\Gamma_f,\Gamma_g\vdash M_1\simeq_{llog}M_2:\tau \stackrel{def}{=} \exists S\subseteq\mathrm{Int},\exists\mathcal{A}\in\mathrm{LTS}_S,\exists s\in S,\forall h\in\mathbf{Cl}(\Sigma),\forall\overrightarrow{x_i:\iota_i},\ (w,\varepsilon,1)\models_{\mathcal{A}}\mathbb{E}\,[\![\tau]\!]^{\widetilde{1}}_{\widetilde{\Gamma_f}}\,(M_1,M_2)$$

where $w=(s,h,h,\widetilde{\nu_L(h)})$, Γ_f (resp. Γ_g) only contains variables of functional (resp. ground) types and $(x_i:\iota_i)$ cover all Γ_g.

Figure 7.3: Definition of Temporal Logical Relations

The operation $\uparrow_i$ (for $i \in \{1,2\}$) transforms a symbolic heap $\overrightarrow{u_k \hookrightarrow u'_k}$ into the formula $\bigwedge_k u_k \hookrightarrow_i u'_k$.

The definition of $\mathbb{E}[\![\tau]\!]^n_e$ uses the parameter n to keep track of the time when the execution of the two terms started. Then, the predicate $\mathbf{P_{pub}}(n)$ in the definition of $\mathbb{R}\mathrm{el}^n_{e,\tau}$ ensures the existence of a public transition between the initial state of the symbolic reduction of M_1, M_2 and the final state once values have been obtained.

In the definition of $\mathbb{R}\mathrm{el}^n_{e,\tau}$, the fresh existential quantifications are on locations $\overrightarrow{y_1}$ and $\overrightarrow{y_2}$ introduced during symbolic execution, i.e. in $\mathrm{dom}(\mathbb{H}_i^{post})\backslash\mathrm{dom}(\mathbb{H}_i^{pre})$, while the universal quantification $\overrightarrow{x'}$ is on all the other variables of $\mathrm{FV}(\mathbb{H}_1^{post}, \mathbb{H}_2^{post})\backslash\mathrm{FV}(\mathbb{H}_1^{pre}, \mathbb{H}_2^{pre})$. In the definition of $\mathbb{E}[\![\tau]\!]^n_e$, the universal quantification is on logical variables $\overrightarrow{x}$ introduced in $\mathbb{H}_1^{pre}, \mathbb{H}_2^{pre}$, i.e. during the symbolic execution to impose constraint on the heap.

It is straightforward that concrete and logical relations correspond on ground values.

Theorem 42: *Let v_1, v_2 two closed values of ground type ι, w a world and (κ, n) an history for w. Then $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{V}[\![\iota]\!]\,(v_1, v_2)$ iff $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\iota]\!]_\varepsilon\, w$.*

Proof If $\iota = \mathrm{Int}$, the theorem is direct since in both cases we have $v_1 = v_2$. If $\iota = \mathrm{ref}\,\iota'$, then the equivalence comes from the fact that $(w, \kappa, n) \models_{\mathcal{A}} \mathrm{Discl}_\iota(v_1, v_2)$ iff $(v_1, v_2) \in (w.\mathcal{S})'_\iota$ and that $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\iota]\!]_\varepsilon\, w$ iff $(v_1, v_2) \in (w.\mathcal{S})'_\iota$. ∎

The goal of Sections 7.6 and 7.7 will be to prove that they corresponds on all types.

7.4 Some Examples

We now detail some examples of $\mathbb{E}[\![\tau]\!]\,(M_1, M_2)$.

7.4.1 Awkward example

Let us first consider the so-called "awkward example":

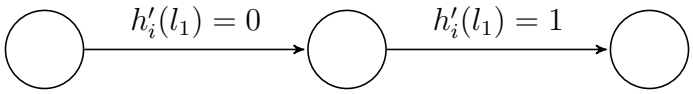
$$\begin{aligned} M_1^{awk} &= \texttt{let x} = \mathrm{ref}\,\texttt{0 in } \lambda\texttt{f.x} := \texttt{1}; \texttt{f}(); !x \\ M_2^{awk} &= \lambda\texttt{f.f}(); \texttt{1} \end{aligned}$$

Its name come from the fact that the first notion of Kripke logical relations introduced by Pitts and Stark[PS98] could not handle it, why their relations were complete. Indeed, they used worlds as fixed invariants on heaps, which couldn't evolve. But here, the state invariant which constrains x to point either to 0 or to 1 is not enough to deduce that the first program returns 1.

The formula $\mathbb{E}[\![(\mathrm{Unit} \to \mathrm{Unit}) \to \mathrm{Int}]\!]^n_e\,(M_1^{awk}, M_2^{awk})$, once simplified with uninterestingly clauses, is equal to:

$$\mathcal{H}\, n_0.\mathcal{N}_1\, l_1.\mathbf{X}\Big((l_1 \hookrightarrow_1 0) \wedge \mathbf{P_{pub}}(n_0) \wedge \Box\big(\mathcal{H}\, n_2.\mathbf{X}((l_1 \hookrightarrow_1 1) \wedge \Box_{\mathbf{pub}}(\forall x_4.((l_1 \hookrightarrow_1 x_4) \Rightarrow \mathbf{X}((l_1 \hookrightarrow_1 x_4) \wedge (x_4 = 1)) \wedge \mathbf{P_{pub}}(n_2))))\big)\Big)$$

The formula imposes that there is a transition from the initial state to state with $l_1 \hookrightarrow 0$ and $\mathbf{P_{pub}}(n_0)$ forces the transition to be public. In the same way, it forces to have a public transition to another state where $l_1 \hookrightarrow 0$. Finally, one need to check that for every public future, one can reach a state where $l_1 \hookrightarrow 0$. This is easily satisfied by the world



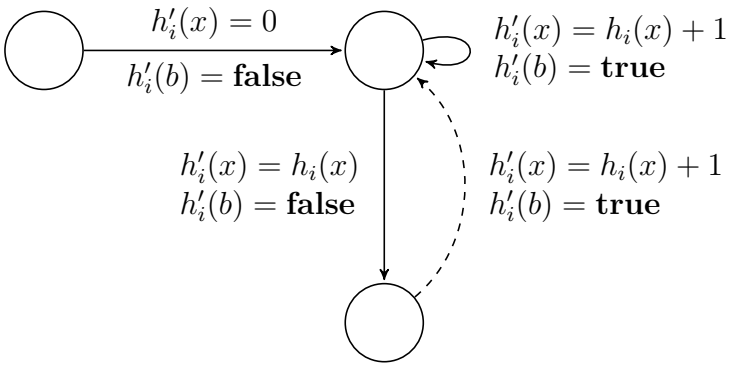
Note that here, we do not really use Hoare-style transitions, because the example already works with an STS.

7.4.2 Hoare-style Worlds

We now consider callback with lock example (taken from [ADR09]) that compares two encoding of a counter object:

$$\begin{array}{rcl} M_1^{cbl} & = & C\ [\mathtt{f}();\mathtt{x} := !\mathtt{x} + 1] \\ M_2^{cbl} & = & C\ [\mathtt{let\ n} = !\mathtt{x}\ \mathtt{in}\ \mathtt{f}();\mathtt{x} := \mathtt{n} + 1] \qquad \text{where} \\ C & = & \mathtt{let\ b} = \mathrm{ref}\ \mathbf{true}\ \mathtt{in\ let\ x} = \mathrm{ref}\ \mathtt{0\ in} \\ (\lambda \mathtt{f.if}\ !\mathtt{b\ then\ b} := \mathbf{false};\ \bullet\ ;\mathtt{b} := \mathbf{true}\ \mathtt{else}\ (), \lambda\_.!\mathtt{x}) & & \end{array}$$

Using an STS approach, two states are needed for each natural number stores in the location `x`, to keep track of the increment of the counter [DNB12]. To solve this issue, we use a label transition system (LTS) approach in order to use Hoare-style description of the heap—relating heaps before the transition (noted h_1, h_2) to heaps after the transition (noted h_1', h_2'):



7.4.3 Name Disclosure

We now consider the following example adapted from the ν-calculus.

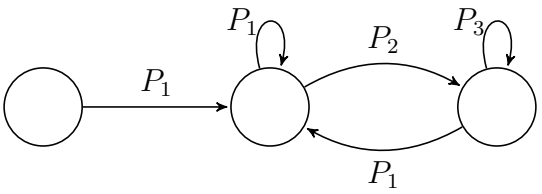
$$\begin{aligned} M_1^{nd} &= \lambda \texttt{c}.\ \texttt{let x} = \text{ref } \texttt{c in } \lambda y.\ x == y \\ M_2^{nd} &= \lambda \texttt{c}.\ \texttt{let x} = \text{ref } \texttt{c in } \lambda y.\ \textbf{false} \end{aligned}$$

Both terms are returning functions that always return **false**. But to know that, the world must explicitly know that `x` is private and thus can not be bound by the context to the variable `y`.

The formula $\mathbb{E}\,[\![\text{ref Int} \to \text{ref (ref Int)} \to \text{Int}]\!]_e^n\,(M_1^{nd}, M_2^{nd})$ is equal to:

$$\begin{aligned} \Box \mathbf{X}_{\mathbf{pub}}^{\mathcal{F}_{\text{ref Int}}} \Big(& \forall l_1, l_2.\text{Discl}_{\text{Int}}(l_1, l_2) \Rightarrow (\mathcal{N}_1\, l_3.\mathbf{X}((l_3 \hookrightarrow_1 l_1) \wedge \\ & \Box \mathbf{X}_{\mathbf{pub}}^{\mathcal{F}_{\text{ref ref Int}}} (\forall l_4, l_5.(\text{Discl}_{\text{ref Int}}(l_4, l_5) \Rightarrow (\forall x_1.(l_3 \hookrightarrow_1 x_1) \Rightarrow \\ & \mathbf{X}(l_3 \neq l_4) \wedge \mathbf{X}((l_3 \neq l_4) \Rightarrow (l_3 \hookrightarrow_1 x_1)))))) \Big) \end{aligned}$$

To simplify the formula, all mentions to public transition have been omitted because it simply forces every transition to be public. The locations l_1 and l_2 represent the location c passed by the context. The location l_3 is the location x and the locations l_4 and l_5 represents the location y passed by the context. This formula is satisfiable because we cannot have $l_3 = l_4$ since we have $\text{Discl}_{\text{ref Int}}(l_4, l_5)$ and l_3 is not disclosed. Here, the world has to be explicit on spans to validate the formula



with

$$\begin{aligned} P_1 &\stackrel{def}{=} \mathcal{S}' \setminus \mathcal{S} = \{(l_1, l_2, \text{Int})\} \qquad (l_1,\, l_2 \text{ fresh}) \\ P_2 &\stackrel{def}{=} \mathcal{S}' \setminus \mathcal{S} = \emptyset \wedge h_i' = H_{l_{x_i}} \\ P_3 &\stackrel{def}{=} (\mathcal{S}' \setminus \mathcal{S} = \emptyset \wedge h_i = H_{l_{x_i}} \wedge h_i' = H_{l'_{x_i}}) \\ &\quad \vee (\mathcal{S}' \setminus \mathcal{S} = \emptyset \wedge h_i = H_{l'_{x_i}} \wedge h_i' = H_{l_{x_i}}) \\ \text{where } H_{l_{x_i}} &\stackrel{def}{=} [l_{x_i} \hookrightarrow l_i] \wedge [l_i \hookrightarrow x_i] \end{aligned}$$

Note that the transition P_3 allows us to rebind the location l_{x_i}, thus performing some kind of local garbage collect to keep the world finite.

Checking that the formula is true is tedious. This leads us to the next section devoted to model checking.

7.5 Model Checking Contextual Equivalence

The definition of $W \models_{\mathcal{A}} \varphi$ introduces quantification over heaps and set of locations. Thus, a general decidability result of the model checking problem is out of reach. Moreover, we need a representation of the transitive closures $\sqsupseteq^*$ and $\sqsupseteq^*_{\mathbf{pub}}$, which are in general not computable. However, in some cases, the decidability of model checking and even sometimes the decidability of the satisfiability of $M_1 \simeq_{llog} M_2 : \tau$ can be obtained.

In this section, we first present a setting in which model checking can be reduced to Presburger arithmetic. Then we show how worlds can be automatically computed in some cases and briefly describe our implementation of those techniques.

7.5.1 Model Checking as Presburger Arithmetic

Suppose that for two terms $\vdash M_1$ and M_2 of type τ:

1. the LTS $\mathcal{A}$ and its transitive closure is expressed in Presburger Arithmetic,
2. heaps are bounded by a constant k,
3. the type τ does not contain ref ι,
4. M_1 and M_2 only use references on integers.

In that situation, the decidability of the model checking is decidable since $W \models_{\mathcal{A}} M_1 \simeq_{llog} M_2 : \tau$ is simply a Presburger formula. Using (2), we can represent heaps as vector of couple (l, v) (with $l \in \mathrm{Loc}$ and $v \in \mathrm{Val}$) of size k. Then, the only second order quantifications are introduced by the quantification on $\mathcal{S}$, but using (3), there is no disclosed reference, so quantification can be removed. Using (4), we know that heaps have a flat structure so we can replace Heap directly by Int^k in the definition of HeapTL.

We can handle creations of location under functions by performing garbage collect as in the name disclosure example. However, as soon as a location is passed by the context through a λ-abstraction or a callback, heaps cannot been bounded. Nevertheless, if there are no private locations introduced before a λ-abstraction expecting a location, we can still work in the bounded-heap setting, using the extensionality principle $\Sigma; \Gamma \vdash \lambda x.M_1 \simeq_{ctx} \lambda x.M_2 : \tau \to \sigma$ iff $\Sigma; \Gamma, x : \tau \vdash M_1 \simeq_{ctx} M_2 : \tau \to \sigma$.

Notice that working with pure terms (i.e. terms which do not create references and whose type does not contain any ref ι occurrences), we can use the single-state world. Thus, using the model checking above with this world, we get the decidability of the contextual equivalence for this fragment—which has already been proved in section 5.5.

7.5.2 An Implementation using SMT-solvers

We have written a program in Haskell that computes the temporal formula $M_1 \simeq_{llog} M_2 : \tau$ for any terms M_1 and M_2. We then verify that heaps can be bounded, by checking that there is no fresh existential quantification on locations under a $\Box$ modality in the

resulting formula. In that case, the program produces a Presburger formula which uses two abstract predicates to represent the transition functions (and their transitive closures) of the world. More precisely, it gives as output a formula written in SMT-LIB2, so that it can be checked, once a world is defined, by an SMT solver. In practice, we have performed our test with Z3. [1].

The program is also able, in some case, to build a world in Presburger arithmetic to check the validity of the formula. The algorithm is based on symbolic execution to compute the shape of the world. Using some heuristic, our program is also able to give the transitive closures of the transition functions for simple cases. Note that the correctness of the transitive closure is checked by a Presburger formula, so the correctness of our heuristics is not critical. It is also possible, as done in Algorithmic Game Semantics, to suppose that integers are bounded, so that the computation of the transitive closure becomes simpler.

The Haskell implementation is available at http://guilhem.jaber.fr/tlr/. Taking two terms, it outputs:

— the formula corresponding to the temporal logical relations, written in HeapTL,
— the Presburger translation written in SMT-LIB2,
— the world generated,
— a definition of the world in SMT-LIB2.

In some cases, as expected since the problem is undecidable in general, the world generated do not model-checked the formula. However, we have been able to prove many examples from the literature. In the Section 7.9 we give as an example the code generated for the awkward example, both for the world and the translation of the temporal formula. More examples can also be found on http://guilhem.jaber.fr/tlr/.

7.6 From Temporal to Concrete Logical Relations

We now prove the soundness of temporal logical relations *w.r.t.* contextual equivalence. To do so, we simply relate it to concrete logical relations, to import the soundness proved in the previous chapter.

First, we state that every possible operational reduction can be simulated with a symbolic execution.

Lemma 53: *Let M a ground-closed term, h a heap such that $\nu_L(M) \subseteq \mathrm{dom}(h)$ and $(M,h) \mapsto^* (M',h')$ with M' irreducible. Then taking $\mathbb{H}^{pre}$ a fresh symbolic heap closed for h, there exists a symbolic execution $(M'',\mathbb{H}^{post},\mathbb{C}) \in \Downarrow (M,\mathbb{H}^{pre})$ and a substitution α on $\mathrm{FV}(\mathbb{H}^{pre},\mathbb{C},\mathbb{H}^{post})$ such that α is not aliased on $\mathrm{dom}(\mathbb{H}^{post})\backslash\mathrm{dom}(\mathbb{H}^{pre})$, $\alpha(\mathbb{C})$ is valid, $h = \alpha(\mathbb{H}^{pre})$ and $h' = \alpha(\mathbb{H}^{post})$. Moreover, if M' is equal to $K[f\,v]$ then $M'' = K'[f\,v']$ with $K = \alpha(K')$ and $v = \alpha(v')$. Otherwise, M' is equal to v, then $M'' = v'$ with $v = \alpha(v')$.*

1. Note that a recent version is needed, since a bug in quantifier elimination in previous versions gives unsatisfiability results for true equivalence.

In the definition above, a substitution $\alpha : A \to \text{Val}$ is said to be *not aliased* on a subset $B \subseteq A$ of locations variables iff for all $b \in B$, $a \in A$ s.t. $a \neq b$, $\alpha(a) \neq \alpha(b)$.

Proof The proof is done by induction on the length of $(M, h) \mapsto^* (M', h')$. When this length is 0, i.e. $(M, h) = (M', h')$, we simply define α as $\alpha(h(l)) = h(l)$ for all $l \in \text{dom}(h)$. The induction step is simply done using Lemma 51. ∎

Lemma 54: *Suppose that* $(M', \mathbb{H}^{post}, \mathbb{C}) \in\Downarrow (M, \mathbb{H}^{pre})$ *such that there exists a substitution* α *on* $\text{FV}(\mathbb{H}^{pre}, \mathbb{C}, \mathbb{H}^{post})$ *such that* $\alpha(\mathbb{C})$ *is valid. Then for all injection* ρ *on* $\nu_L(\alpha(\mathbb{H}^{post}))\backslash\nu_L(\alpha(\mathbb{H}^{pre}))$, $\rho(\alpha(\mathbb{C}))$ *is valid.*

Proof The only constraints on locations generated by the symbolic reduction are equalities and inequalities, whose validity is conserved by injections. ∎

Lemma 55: *Let* $(h_1, h_2, \mathcal{D}) : w$ *and* $\mathbb{H}_1, \mathbb{H}_2$ *two symbolic heaps with* α *a substitution on* $\text{FV}(\mathbb{H}_1, \mathbb{H}_2)$. *Then if* $h_i = \alpha(\mathbb{H}_i)$, *we have* $W \models_{\mathcal{A}} \alpha(\uparrow_1(\mathbb{H}_1) \wedge \uparrow_2(\mathbb{H}_2) \wedge \mathbb{H}_1 \overset{\mathcal{D}}{\sim} \mathbb{H}_2)$ *where* $W = (w, \kappa, n)$.

Proof We first decompose h_i into $h_i^{priv} \cdot h_i^{pub}$ with $h_i^{priv} = w.h_i$ and $\text{dom}(h_i^{pub}) = D_i$. Then, for all $l \mapsto_i v$ in $\alpha(\uparrow_i(\mathbb{H}_i))$, if $l \in \text{dom}(h_i^{priv})$, we have $w.h_i(l) = h(l) = v$ so $W \models_{\mathcal{A}} l \mapsto_i v$. Otherwise, $W \models_{\mathcal{A}} l \mapsto_i v$ is trivial. Finally, by definition of $(h_1, h_2, \mathcal{D}) : w$, $h_1^{pub} \sim h_2^{pub}$ so $W \models_{\mathcal{A}} \mathbb{H}_1 \overset{\mathcal{D}}{\sim} \mathbb{H}_2$. ∎

Lemma 56: *Let* $\mathbb{H}_1, \mathbb{H}_2$ *two symbolic heaps with* α *a substitution on* $\text{FV}(\mathbb{H}_1, \mathbb{H}_2)$. *If* $(w, \kappa, n) \models_{\mathcal{A}} \alpha(\uparrow_1(\mathbb{H}_1) \wedge \uparrow_2(\mathbb{H}_2) \wedge \mathbb{H}_1 \overset{\mathcal{D}}{\sim} \mathbb{H}_2)$ *then there exists a span* $\mathcal{D}$ *on location such that* $(\alpha(\mathbb{H}_1), (\alpha(\mathbb{H}_2), \mathcal{D}) : w$.

Proof We first decompose $\alpha(\mathbb{H}_i)$ into $h_i^{priv} \cdot h_i^{pub}$ with h_i^{pub} closed and $\text{dom}(h_i^{pub}) = D_i \subseteq (w.\mathcal{S})_i$. From $(w, \kappa, n) \models_{\mathcal{A}} \alpha(\uparrow_i(\mathbb{H}_i))$ we get that $h_i^{priv} = w.h_i$. And from $(w, \kappa, n) \models_{\mathcal{A}} \alpha(\mathbb{H}_1 \overset{\mathcal{D}}{\sim} \mathbb{H}_2)$ we get that $h_1^{pub} \sim_{w.\mathcal{S}} h_2^{pub}$. Thus, $(\alpha(\mathbb{H}_1), \alpha(\mathbb{H}_2), \mathcal{D}) : w$. ∎

Lemma 57: *Suppose* $W \models_{\mathcal{A}} \alpha(\mathbb{V}[\![\tau]\!]_e(v_1, v_2))$ *with* α *a substitution on* $\text{FGV}(v_1, v_2)$. *Then* $W \models_{\mathcal{A}} \mathbb{V}[\![\tau]\!]_e(\alpha(v_1), \alpha(v_2))$.

Lemma 58: *Suppose* $W \models_{\mathcal{A}} \alpha(\mathbb{K}[\![\sigma, \tau]\!]_e^n(K_1, K_2))$ *with* α *a substitution on* $\text{FGV}(K_1, K_2)$. *Then* $W \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!]_e^n(\alpha(K_1), \alpha(K_2))$.

Lemma 59: *Suppose* $W \models_{\mathcal{A}} \alpha(\mathbb{E}[\![\tau]\!]_e^n(M_1, M_2))$ *with* α *a substitution on* $\text{FGV}(M_1, M_2)$. *Then* $W \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!]_e^n(\alpha(M_1), \alpha(M_2))$.

Proof The three lemmas are proved by a mutual induction on type and on the number of ground callbacks of terms considered. It is straightforward using the fact that substitutions go through logical connectives. The only case which is not direct is the generation of all the $\mathbb{S}_i \in\Downarrow (M_i, \mathbb{H}_i)$ in the definition of $\mathbb{E}[\![\tau]\!]_e^n(M_1, M_2)$. In this case, we simply use the fact that $\Downarrow (\alpha(M_i), \alpha(\mathbb{H}_i))$ is included in $\alpha(\Downarrow (M_i, \mathbb{H}_i))$, since some rules which where non-deterministic because of the variables become deterministic. ∎

Lemma 60: *Let h_1, h_2 two closed heaps, $\mathcal{D}, \mathcal{D}'$ two spans and v_1, v_2 two values s.t. $\mathcal{D}'_i = \mathrm{discl}(v_i, h_i, \mathcal{D}_i)$. Let (w, κ, n) a world with history s.t. $(w, \kappa, n) \models_{\mathcal{A}} h_1 \overset{\mathcal{D}}{\sim} h_2$. Moreover, suppose that $(w, \kappa, n) \models_{\mathcal{A}} v_1 \overset{\mathcal{D}}{\sim} v_2 \iota$ if $v_1, v_2 \in \mathrm{Loc}_\iota$. Then $\mathcal{D}' \subseteq w.\mathcal{S}$.*

Then, using an induction on types and on the number of ground callbacks, we can prove that when two terms are in temporal logical relation for a given world then they are in concrete logical relation for class of worlds obtained by permuting location of the initial world. A similar result holds for values and contexts.

Theorem 43 (Soundness for Values): *Let v_1, v_2 two ground-closed values with e a relational environment for them, and (w, κ, n) a world with history. If $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{V}[\![\tau]\!]_e (v_1, v_2)$, then $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\tau]\!]_e w$.*

Theorem 44 (Soundness for Contexts): *Let K_1, K_2 two ground-closed contexts and e a relational environment for them, and (w, κ, n) a world with history. suppose that $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{K}[\![\tau, \sigma]\!]^m_e (K_1, K_2)$, then, writing w_0 for $(w.\mathcal{A}, \kappa(m))$, we have $(K_1, K_2) \in \mathcal{K}_{\mathcal{A}}[\![\tau, \sigma]\!]_e (w, w_0)$.*

Theorem 45 (Soundness for Terms): *Let M_1, M_2 two ground-closed terms and e a relational environment for them, and (w, κ, n) a world with history. suppose that $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!]^m_e (M_1, M_2)$, then, writing w_0 for $(w.\mathcal{A}, \kappa(m))$, we have (M_1, M_2) is in $\mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e (w, w_0)$.*

Proof The three theorems are proved by a mutual induction on types and on the number of ground callbacks of the considered terms.

Proof (Soundness for Values) By case analysis on τ :

— When τ is ground, we get the result from theorem 42.

— When $\tau = \iota \to \sigma$, suppose $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{V}[\![\iota \to \sigma]\!]_e (v_1, v_2)$. Let $w_1 \sqsupseteq^* w$, $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $W_1 = (w_1, \kappa_1, n+1)$, so from the hypothesis we get the existence of $w_2 \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w_1$. Then, taking $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}}[\![\iota]\!]_e w_2$, from theorem 42, we get that $W_2 \models_{\mathcal{A}} \mathbb{V}[\![\iota]\!]_e (u_1, u_2)$ where $W_2 = (w_2, \kappa_2, n+2)$, $\kappa_2 = \kappa_1 \cdot (n+1 \hookrightarrow (w_1.s, w_1.h_1, w_1.h_2, w_1.\mathcal{S}))$. So from the hypothesis, we know that $W_2 \models_{\mathcal{A}} \mathbb{E}[\![\sigma]\!]^{n+2}_e (v_1\ u_1, v_2\ u_2)$. Then, using the induction hypothesis $(v_1\ u_1, v_2\ u_2) \in \mathcal{E}_{\mathcal{A}}[\![\sigma]\!]_e (w_2, w_2)$.

— When $\tau = \sigma \to \sigma'$, suppose $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{V}[\![\sigma \to \sigma']\!]_e (v_1, v_2)$. Let $w_1 \sqsupseteq^* w$, $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $W_1 = (w_1, \kappa_1, n+1)$, so from the hypothesis we get the existence of $w_2 \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} w_1$ s.t.

$$W_2 \models_{\mathcal{A}} \mathbb{E}[\![\sigma']\!]^{n+1}_{e \cdot (y_1, y_2, \sigma)} (v_1\ y_1, v_2\ y_2)$$

with y_1, y_2 fresh in e. Using the induction hypothesis we get that

$$(v_1\ y_1, v_2\ y_2) \in \mathcal{E}_{\mathcal{A}}[\![\sigma']\!]_{e \cdot (y_1, y_2, \sigma)} (w_2, w_2).$$

■

Proof (Soundness for Contexts) By case analysis on τ

— When $\tau = \iota$, suppose $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{K}\,[\![\iota, \sigma]\!]^m_e\,(K_1, K_2)$. Let $w_1 \sqsupseteq^* w$, $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $W_1 = (w_1, \kappa_1, n+1)$, so from the hypothesis we get the existence of $w_2 \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w_1$. Then, taking $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}}\,[\![\iota]\!]_e\,w_2$, from theorem 42, we get that $W_2 \models_{\mathcal{A}} \mathbb{V}\,[\![\iota]\!]_e\,(u_1, u_2)$ where $W_2 = (w_2, \kappa_2, n+2)$, $\kappa_2 = \kappa_1 \cdot (n+1 \hookrightarrow (w_1.s, w_1.h_1, w_1.h_2, w_1.\mathcal{S}))$. So from the hypothesis, we know that $W_2 \models_{\mathcal{A}} \mathbb{E}\,[\![\sigma]\!]^{n+2}_e\,(K_1[u_1], K_2[u_2])$. Then, using the induction hypothesis $(K_1[u_1], K_2[u_2]) \in \mathcal{E}_{\mathcal{A}}\,[\![\sigma]\!]_e\,(w_2, w_2)$.

— When τ is of functional type,, suppose $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{K}\,[\![\tau, \sigma]\!]^m_e\,(K_1, K_2)$. Let $w_1 \sqsupseteq^* w$, $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $W_1 = (w_1, \kappa_1, n+1)$, so from the hypothesis we get the existence of $w_2 \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} w_1$ s.t.

$$W_2 \models_{\mathcal{A}} \mathbb{E}\,[\![\sigma]\!]^{n+1}_{e\cdot(x_1,x_2,\sigma)}\,(K_1[x_1], K_2[x_2])$$

Using the induction hypothesis we get that $(K_1[x_1], K_2[x_2]) \in \mathcal{E}_{\mathcal{A}}\,[\![\sigma]\!]_{e\cdot(x_1,x_2,\tau)}\,(w_2, w_2)$.

■

Proof (Soundness for Terms) Let $(h_1, h_2, \mathcal{D}) : w$ s.t. $h_i \in \mathbf{Cl}(\nu_L(M_i))$ and suppose $\langle M_i, h_i, \mathcal{D}_i\rangle \to \langle M_i', h_i', \mathcal{D}_i'\rangle$. Then from Lemma 53, we get two symbolic execution $(M_i, \mathbb{H}_i^{pre}) \mapsto_s^* (M_i'', \mathbb{H}_i^{post}, \mathbb{C}_i)$ and two substitution α_i on $\mathrm{FV}(\mathbb{H}_i^{pre}, \mathbb{C}_i, \mathbb{H}_i^{post})$ which are not aliased on $\mathrm{dom}(\mathbb{H}_i^{post})\backslash\mathrm{dom}(\mathbb{H}_i^{pre})$ Writing L_i for $\mathrm{dom}(\mathbb{H}_i^{post})\backslash\mathrm{dom}(\mathbb{H}_i^{pre})$, we can decompose α_i into three substitutions α_i^{pre} on $\mathrm{FV}(\mathbb{H}_i^{pre})$, α_i^L on L_i and α_i^{post} on $\mathrm{FV}(\mathbb{C})\backslash(\mathrm{FV}(\mathbb{H}_i^{pre}) \cup L$. Then, $h_i = \alpha_i^{pre}(\mathbb{H}_i^{pre})$, so writing α^{pre} for $\alpha_1^{pre} \cdot \alpha_1^{pre}$ (we can suppose that $\mathrm{FV}(\mathbb{H}_1^{pre})$ and $\mathrm{FV}(\mathbb{H}_2^{pre})$ are disjoint), we get from lemma 55 that

$$(w, \kappa, n) \models_{\mathcal{A}} \alpha^{pre}(\uparrow_1(\mathbb{H}_1^{pre}) \wedge \uparrow_2(\mathbb{H}_2^{pre}) \wedge \mathbb{H}_1 \overset{\mathcal{D}}{\sim} \mathbb{H}_2)$$

Using this with the hypothesis $W \models_{\mathcal{A}} \mathbb{E}\,[\![\tau]\!]^n_e\,(M_1, M_2)$ we get two substitution $\beta_i^L : L_i \to \mathrm{Loc}$ such that $\alpha_i^{post} \cdot \beta_i^L$ is not aliased on L_i. Define an injection

$$\rho_i : \mathrm{dom}(\alpha_i(\mathbb{H}^{post}))\backslash\mathrm{dom}(\alpha_i(\mathbb{H}^{pre})) \hookrightarrow \mathrm{Loc}$$

as $\beta_i \circ (\alpha_i^L)^{-1}$. Then, from the fact that $\alpha_i(\mathbb{C}_i)$ is valid and lemma 54, we get that $\rho_i(\alpha_i(\mathbb{C}_i))$ is valid, i.e. $(\underbrace{\alpha_i^{pre} \cdot \beta_i \cdot \alpha_i^{post}}_{\alpha_i'})(\mathbb{C}_i)$ is valid (indeed $\rho_i \circ \alpha_i = \alpha_i'$).

Thus, using again the hypothesis $W \models_{\mathcal{A}} \mathbb{E}\,[\![\tau]\!]^n_e\,(M_1, M_2)$, we get the existence of a world $w_1 \sqsupseteq w$ s.t. defining $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $W_1 = (w_1, \kappa_1, n+1)$, we have $W_1 \models_{\mathcal{A}} \alpha'(\uparrow_1(\mathbb{H}_1^{post}) \wedge \uparrow_2(\mathbb{H}_2^{post}) \wedge \mathbb{H}_1^{post} \overset{\mathcal{D}}{\sim} \mathbb{H}_2^{post})$. Moreover, $\rho_i(h_i') = \alpha_i'(\mathbb{H}_i^{post})$ and $\rho_i(M_i') = \alpha_i'(M_i'')$.

— If both $M_i' = K_i[f_i\, v_i]$, we have $M_i'' = K_i'[f_i\, v_i']$ with $K_i = \alpha_i(K_i')$ and $v_i = \alpha_i(v_i')$. Then, from $(w, \kappa, n) \models_{\mathcal{A}} \mathbb{E}\,[\![\tau]\!]^n_e\,(M_1, M_2)$ we get that

— there exists a functional type $\sigma \to \sigma'$ such that $(f_1, f_2) \in e_{\sigma\to\sigma'}$

— $w_1 \models_{\mathcal{A}} \alpha'(\mathbb{V}[\![\sigma]\!]_e(v_1', v_2'))$, so using lemma 59 and the induction hypothesis on values, we get that $(\rho_1(v_1), \rho_2(v_2)) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_e w_1$.
— $w_1 \models_{\mathcal{A}} \Box_{\mathbf{pub}}\alpha'(\mathbb{K}[\![\sigma', \tau]\!]_e^m(K_1', K_2'))$, so using again lemma 59 and the induction hypothesis on contexts, we get that $(\rho_1(K_1), \rho_1(K_2)) \in \mathcal{K}_{\mathcal{A}}[\![\sigma', \tau]\!]_e(w_1, w_0)$.

So we still have to prove that $(\rho_1(h_1'), \rho_2(h_2'), \mathcal{D}'') : w_1$ where $\mathcal{D}'' = (\rho_1 \times \rho_2)(\mathcal{D}')$. To do so, we first prove that $\mathcal{D}'' \subseteq (w_1.\mathcal{S})_i$ using lemma 60, since $\mathcal{D}_i'' = \mathrm{discl}(\rho_i(v_i), \rho_i(h_i'), \rho_i(\mathcal{D}_i))$. And we conclude using lemma 56.

— Otherwise, if both $M_i' = v_i$, we have $M_i'' = v_i'$ with $v_i = \alpha(v_i')$. Then, using the same reasoning as before, we get that $(\rho_1(v_1), \rho_2(v_2)) \in \mathcal{V}_{\mathcal{A}}[\![\sigma]\!]_e w_1$ and $(\rho_1(h_1', \rho_1(h_2', \mathcal{D}'') : w_1$. Moreover, $W_1 \models_{\mathcal{A}} \mathbf{P}_{\mathbf{pub}}(m)$ gives us that $w_1 \sqsupseteq_{\mathbf{pub}}^* w_0$. So $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}}[\![\tau]\!]_e(w, w_0)$ ∎

Corollary 7: *Let M_1, M_2 two terms such that $\Sigma; \Gamma \vdash M_1, M_2 : \tau$. Then $\Sigma; \Gamma \vdash M_1 \simeq_{llog} M_2 : \tau$ implies $\Sigma; \Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$.*

7.7 From Concrete to Temporal Logical Relations

Finally, we prove the completeness of temporal logical relations *w.r.t.* contextual equivalence. To do so, we simply relate it to concrete logical relations, to import the soundness proved in the previous chapter.

Similarly to Lemma 53, we state that every possible symbolic execution of a term can be simulated with an operational reduction leading to a normal form.

Lemma 61: *Let M a ground-closed term, $\mathbb{H}^{pre}$ a fresh closed symbolic heap for M and α^{pre} a substitution $\mathrm{FV}(\mathbb{H}^{pre}) \to \mathrm{Val}$. Let $(M', \mathbb{H}^{post}, \mathbb{C}) \in \Downarrow (M, \mathbb{H}^{pre})$, then defining h as $\alpha^{pre}(\mathbb{H}^{pre})$, there exists*

— *a reduction $(M, h) \mapsto^* (M'', h')$ with (M'', h') irreducible,*
— *a substitution $\alpha^{post} : (\mathrm{FV}(\mathbb{H}^{post}, \mathbb{C}) \backslash \mathrm{FV}(\mathbb{H}^{pre})) \to \mathrm{Val}$*

such that, writing α for $\alpha^{pre} \cdot \alpha^{post}$,

— *$\alpha(\mathbb{C})$ is valid,*
— *α is not aliased on $\mathrm{dom}(\mathbb{H}^{post}) \backslash \mathrm{dom}(\mathbb{H}^{pre})$,*
— *if M' is equal to $K[f\ v]$ then $M'' = K'[f\ v']$ with $K' = \alpha(K)$ and $v' = \alpha(v)$,*
— *otherwise, M' is equal to v, and $M'' = v'$ with $v' = \alpha(v)$.*

Theorem 46 (Completeness for Values): *Let v_1, v_2 two ground-closed values, e a relational environment for them and$W = (w, \kappa, n)$ a world with history, If $(v_1, v_2) \in \mathcal{V}_{\mathcal{A}}[\![\tau]\!]_e w$, then $W \models_{\mathcal{A}} \mathbb{V}[\![\tau]\!]_e(v_1, v_2)$.*

Theorem 47 (Completeness for Contexts): *Let $W = (w, \kappa, n)$ a world with history, K_1, K_2 two ground-closed terms, e an environment for them. Suppose $w_0 = (w.\mathcal{A}, \kappa(m))$, then if $(K_1, K_2) \in \mathcal{K}_{\mathcal{A}}[\![\sigma, \tau]\!]_e w, w_0$ then $W \models_{\mathcal{A}} \mathbb{K}[\![\sigma, \tau]\!]_e^m(K_1, K_2)$.*

Theorem 48 (Completeness for Terms): *Let $W = (w, \kappa, n)$ a world with history, M_1, M_2 two ground-closed terms, e an environment for M_1, M_2. Suppose $w_0 = (w.\mathcal{A}, \kappa(m))$, if $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e (w, w_0)$ then $W \models_{\mathcal{A}} \mathbb{E} [\![\tau]\!]^m_e (M_1, M_2)$.*

Proof Proof (Completeness for Values) By case analysis on τ

— When τ is ground, we get the result from theorem 42.

— When $\tau = \iota \to \tau'$, from the hypothesis we get that for all $w_1 \sqsupseteq w$ there exists $w_2 \sqsupseteq^{\mathcal{F}_\iota}_{\mathbf{pub}} w_1$ s.t. for all $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}} [\![\iota]\!]_e\, w_2$, $(v_1\ u_1, v_2\ u_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau']\!]_e (w_2, w_2)$.
We define $W_1 = (w_1, \kappa_1, n+1)$ and $W_2 = (w_2, \kappa_2, n+2)$, where $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $\kappa_2 = \kappa_1 \cdot (n+1 \hookrightarrow (w_1.s, w_1.h_1, w_1.h_2, w_1.\mathcal{S}))$.
Then, taking u_1, u_2 two values of type ι s.t. $W_2 \models_{\mathcal{A}} \mathbb{V} [\![\iota]\!]_e (u_1, u_2)$, from theorem 42, we get that $(u_1, u_2) \in \mathcal{V}_{\mathcal{A}} [\![\iota]\!]_e\, w_2$. So from the theorem hypothesis, we know that $(v_1\ u_1, v_2\ u_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau']\!]_e (w_2, w_2)$, and from the induction hypothesis on terms, we get that $W_2 \models_{\mathcal{A}} \mathbb{E} [\![\tau']\!]^{n+2}_e (v_1\ u_1, v_2\ u_2)$.

— When $\tau = \sigma \to \tau'$ with σ not ground, from the hypothesis we get that for all $w_1 \sqsupseteq w$ there exists $w_2 \sqsupseteq^{\mathcal{F}}_{\mathbf{pub}} w_1$ s.t. $(v_1\ y_1, v_2\ y_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau']\!]_{e\cdot(y_1,y_2,\sigma)} (w_2, w_2)$.
We define $W_1 = (w_1, \kappa_1, n+1)$ and $W_2 = (w_2, \kappa_2, n+2)$, where $\kappa_1 = \kappa \cdot (n \hookrightarrow (w.s, w.h_1, w.h_2, w.\mathcal{S}))$ and $\kappa_2 = \kappa_1 \cdot (n+1 \hookrightarrow (w_1.s, w_1.h_1, w_1.h_2, w_1.\mathcal{S}))$.
Then from the induction hypothesis on terms, we get that

$$W_2 \models_{\mathcal{A}} \mathbb{E} [\![\tau']\!]^{n+2}_{e\cdot(y_1,y_2,\sigma)} (v_1\ y_1, v_2\ y_2).$$

■

Proof (Completeness for Terms) Let $\mathbb{H}_1^{pre}, \mathbb{H}_2^{pre}$ two fresh symbolic heaps for respectively M_1, M_2, and $(M_i', \mathbb{H}_i^{post}, \mathbb{C}_i) \in \Downarrow (M_i, \mathbb{H}_i^{pre})$. Let α_i^{pre} a substitution on $\mathrm{FV}(\mathbb{H}_i^{pre})$, and suppose that, writing α^{pre} for $\alpha_1^{pre} \cdot \alpha_2^{pre}$, we have $W \models_{\mathcal{A}} \alpha^{pre}((\uparrow_1 (\mathbb{H}_1^{pre}) \wedge \uparrow_2 (\mathbb{H}_2^{pre}) \wedge \mathbb{H}_1^{pre} \overset{\mathcal{D}}{\sim} \mathbb{H}_2^{pre}))$. Then defining h_i as $\alpha_i^{pre}(\mathbb{H}_i^{pre})$ and writing $\mathcal{D}_i$ for $(w.\mathcal{S})_i \cap \mathrm{dom}(h_i)$, we get using lemma 56 that $(h_1, h_2, \mathcal{D}) : w$. So, from lemma 61, we get that there exists

— a substitution α_i^{post} a substitution on $\mathrm{FV}(\mathbb{H}_1^{pre}, \mathbb{H}_2^{pre})$

— an interactive reduction $\langle M_i, h_i, \mathcal{D}_i \rangle \to \langle M_i'', h_i', \mathcal{D}_i' \rangle$

such that, writing α_i for $\alpha_i^{pre} \cdot \alpha_i^{post}$, $h_i' = \alpha_i(\mathbb{H}_i^{post})$ and $\alpha_i(\mathbb{C}_i)$ is valid and α_i is not aliased on $\nu_L(\mathbb{H}_i^{post}) \backslash \nu_L(\mathbb{H}_i^{pre})$.

Let consider two heaps h_i^{pub} of domain $(w.\mathcal{S})_i \backslash \mathrm{dom}(h_i)$ s.t. $h_1^{pub} \sim_{w.\mathcal{S}} h_2^{pub}$. Then, writing $\widetilde{h_i}$ for $h_i \cdot h_i^{pub}$ and $\widetilde{\mathcal{D}}$ for $\mathcal{D} \cup \mathrm{dom}(h_i^{pub})$ (i.e. $\widetilde{\mathcal{D}} = (w.\mathcal{S})_i$, we have $(h_1 \cdot h_1^{pub}, h_2 \cdot h_2^{pub}, w.\mathcal{S}) : w$ and $\left\langle M_i, \widetilde{h_i}, \widetilde{\mathcal{D}_i} \right\rangle \to \left\langle M_i'', \widetilde{h_i'}, \widetilde{\mathcal{D}_i'} \right\rangle$ where $\widetilde{h_i'} = h_i' \cdot h_i^{pub}$ and $\widetilde{\mathcal{D}_i'} = \mathcal{D}_i' \cup \mathrm{dom}(h_i^{pub})$.

So from the hypothesis $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e (w, w_0)$, we get the existence of two injections $\rho_i : (\mathrm{dom}(h_i') \backslash \mathrm{dom}(\widetilde{h_i})) \in \mathrm{Loc}$ (since $\mathrm{dom}(\widetilde{h_i'}) \backslash \mathrm{dom}(\widetilde{h_i}) = \mathrm{dom}(h_i') \backslash \mathrm{dom}(\widetilde{h_i})$, and a world $w_1 \sqsupseteq w$ such that $(\rho_1^{-1} \cdot h_1', \rho_2^{-1} \cdot h_2', (\rho_1^{-1} \times \rho_2^{-1}) \cdot \mathcal{D}') : w_1$.

Then, defining α_i' as $\rho_i^{-1} \circ \alpha_i$, we get from lemma 54 that $\alpha_i'(\mathbb{C}_i)$ is also valid. Thus, we instantiate the fresh names quantifiers with the locations in $\alpha'(\mathrm{dom}(\mathbb{H}_i^{post}) \backslash \mathrm{dom}(\mathbb{H}_i^{pre}))$, which are fresh since α is not aliased on on $\mathrm{dom}(\mathbb{H}_i^{post}) \backslash \mathrm{dom}(\mathbb{H}_i^{pre})$ and ρ_i^{-1} is a bijection.

So, using using lemma 55, $W_1 \models_{\mathcal{A}} \alpha'(\uparrow_1 (\mathbb{H}_1^{post}) \wedge \uparrow_2 (\mathbb{H}_2^{post}) \wedge \mathbb{H}_1^{post} \overset{\mathcal{D}}{\sim} \mathbb{H}_2^{post})$ where $W_1 = (w_1, \kappa \cdot n \hookrightarrow (w.s, w.h_1, w.h_2), n+1)$. Then,

— If both M_i'' are equal to some $K_i'[f_i\, v_i']$, then $K_i' = \alpha(K)$ and $v_i' = \alpha(v)$. So using the fact that $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e\, w, w_0$, we get that $(f_1, f_2) \in e_{\sigma \to \sigma'}$ with $(\rho_1^{-1} \cdot v_1', \rho_2^{-1} \cdot v_2') \in \mathcal{V}_{\mathcal{A}} [\![\sigma]\!]_e\, w_1$ and $(\rho_1^{-1}(K_1'), \rho_2^{-1}(K_2')) \in \mathcal{K}_{\mathcal{A}} [\![\sigma', \tau]\!]_e\, (w_2, w_0)$. Thus, the induction hypothesis gives us that

$$W_1 \models_{\mathcal{A}} \mathbb{V} [\![\sigma]\!]_e\, (\alpha'(v_1), \alpha'(v_2)) \wedge \mathbb{K} [\![\sigma', \tau]\!]_e^n\, (\alpha'(K_1), \alpha'(K_2))$$

and using lemma 59 we get that $W_1 \models_{\mathcal{A}} \alpha'(\mathbb{V} [\![\sigma]\!]_e\, n(v_1, v_2) \wedge \mathbb{K} [\![\sigma', \tau]\!]_e^n\, (K_1, K_2)$.

— If both M_i'' are to equal to some v_i', then $v_i' = \alpha(v_i)$, and with the same reasoning we prove that $W_1 \models_{\mathcal{A}} \alpha'(\mathbb{V} [\![\sigma]\!]_e\, (v_1, v_2)$. And from $(M_1, M_2) \in \mathcal{E}_{\mathcal{A}} [\![\tau]\!]_e\, (w, w_0)$ we get that $w_1 \sqsupseteq_{\mathbf{pub}} w_0$, so $W_1 \models_{\mathcal{A}} \mathbf{P}_{\mathbf{pub}}(m)$ since $\kappa(m) = w_0$. ∎

Corollary 8: *Let M_1, M_2 two terms such that $\Sigma; \Gamma \vdash M_1, M_2 : \tau$. Then $\Sigma; \Gamma \vdash M_1 \simeq_{clog} M_2 : \tau$ implies $\Sigma; \Gamma \vdash M_1 \simeq_{llog} M_2 : \tau$.*

7.8 Discussion

Finally, we can state the soundness and completeness of temporal logical relations *w.r.t.* contextual equivalence:

Theorem 49: *Let M_1, M_2 two terms such that $\Sigma; \Gamma \vdash M_1, M_2 : \tau$. Then $\Sigma; \Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$ iff $\Sigma; \Gamma \vdash M_1 \simeq_{llog} M_2 : \tau$.*

To achieve this result, we have used four intermediate layers:

— Nominal Game Semantics, via the work of Murawski and Tzevelekos [MT11b],
— Trace Semantics, presented in Chapter 5,
— Kripke Trace Semantics, presented in Section 6.3,
— Concrete Logical Relations, presented in Chapter 6.

It would have been possible to avoid the use of nominal game semantics, since Laird had already proved [Lai07] the full abstraction for a similar model of our trace semantics. However, we have preferred to establish a link between trace and game semantics since there is now an important literature on game models for various languages, that we can import following our approach. It would be particularly interesting to extend this work to universal and existential types. This would allow us to apply standard methods to derive parametricity and representation independance from logical relations, following the seminal work of Reynolds[Rey83].

An other possibility is to reason on terms with recursion, via a marriage between guarded recursive types and temporal logic. Indeed, the Löb rule is particularly appropiate to reason on recursion. However, to achieve some model-checking or decidability results in this setting seems uncertain.

7.9 Appendix: SMTLIB Code for the "Awkward Example"

We provide in a file awkward.l the following pair of term:

```
(let x1 = ref 0 in \f1:Unit-> Unit.x1:=1;f1 unit;!x1,
\f2:Unit->Unit.f2 unit;1)
```

We generate the world using

```
./world -flat -compact Test-Suite/Awkward/awkward.l
```

which generate the following code:

```
(define-fun TransPriv ((s1 Int) (s2 Int) (_x6 Int) (_x7 Int) ) Bool
(ite (and (= s1 4) (= s2 4)) (or (and (= _x6 _x7) ) (= _x7 1) )
(ite (and (= s1 2) (= s2 2)) (and (= _x6 _x7) )
(ite (and (= s1 0) (= s2 0)) (and (= _x6 _x7) )
(ite (and (= s1 2) (= s2 4)) (= _x7 1)
(ite (and (= s1 0) (= s2 2)) (= _x7 0)
false ))))))

(define-fun TransPrivT ((s1 Int) (s2 Int) (_x6 Int) (_x7 Int) ) Bool
(ite (and (= s1 4) (= s2 4)) (or (and (= _x6 _x7) ) (= _x7 1) )
(ite (and (= s1 0) (= s2 0)) (and (= _x6 _x7) )
(ite (and (= s1 2) (= s2 2)) (and (= _x6 _x7) )
(ite (and (= s1 0) (= s2 2)) (= _x7 0)
(ite (and (= s1 2) (= s2 4)) (= _x7 1)
false ))))))

(define-fun TransPub ((s1 Int) (s2 Int) (_x6 Int) (_x7 Int) ) Bool
(ite (and (= s1 4) (= s2 4)) (or (and (= _x6 _x7) ) (= _x7 1) )
(ite (and (= s1 2) (= s2 2)) (and (= _x6 _x7) )
(ite (and (= s1 0) (= s2 0)) (and (= _x6 _x7) )
(ite (and (= s1 2) (= s2 4)) (= _x7 1)
(ite (and (= s1 0) (= s2 2)) (= _x7 0)
false ))))))

(define-fun TransPubT ((s1 Int) (s2 Int) (_x6 Int) (_x7 Int) ) Bool
(ite (and (= s1 4) (= s2 4)) (or (and (= _x6 _x7) ) (= _x7 1) )
(ite (and (= s1 0) (= s2 0)) (and (= _x6 _x7) )
(ite (and (= s1 2) (= s2 2)) (and (= _x6 _x7) )
(ite (and (= s1 0) (= s2 2)) (= _x7 0)
```

```
(ite (and (= s1 2) (= s2 4)) (= _x7 1)
false ))))))
```

We generate the formula using

`./equiv -flat -smt Test-Suite/Awkward Example/awkward.l`,

which generates the following code:

```
(assert (exists ((_s5 Int)(_x6 Int)(_s7 Int)(_x8 Int))
 (and (TransPriv _s5 _s7 _x6 _x8) (and (= _x8 0)
  (forall ((_s9 Int)(_x10 Int)) (=> (TransPrivT _s7 _s9 _x8 _x10)
   (forall ((_x3 Int))
    (=> (= _x10 _x3) (exists ((_s11 Int)(_x12 Int))
     (and (TransPriv _s9 _s11 _x10 _x12)
      (and (= _x12 1) (forall ((_s13 Int)(_x14 Int))
       (=> (TransPubT _s11 _s13 _x12 _x14) (forall ((_x4 Int))
        (=> (= _x14 _x4) (exists ((_s15 Int)(_x16 Int))
         (and (TransPriv _s13 _s15 _x14 _x16)
          (and (= _x16 _x4) (= _x4 1)
           (TransPub _s9 _s15 _x10 _x16)))))))))))))))
  (TransPub _s5 _s7 _x6 _x8) ) )))

(check-sat)
```

Checking the two codes together with `z3`, we get that it is indeed satisfiable.

Bibliography

[ADR09] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *Proceedings of POPL'09*, 2009. 13, 14, 16, 18, 30, 38, 39, 41, 43, 159, 203

[AGM+04] S. Abramsky, D. Ghica, A. Murawski, L. Ong, and I. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS'04)*, pages 150–159. IEEE, 2004. 25, 108

[AHM98] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98)*, pages 334–344. IEEE, 1998. 25, 108

[Ahm04] A. Ahmed. *Semantics of types for mutable state*. PhD thesis, Princeton University, Princeton, NJ, USA, 2004. 13, 39

[Ahm06] A. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. *Programming Languages and Systems*, pages 69–83, 2006. 14, 39

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000. 25, 108

[AM98] S. Abramsky and G. McCusker. Call-by-value games. In *Computer Science Logic*, pages 1–17. Springer, 1998. 110

[AM01] A.W. Appel and D. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(5):657–683, 2001. 13, 14, 39

[AM13] Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional programming*, pages 197–208. ACM, 2013. 104

[AMRV07] A.W. Appel, P.-A. Melliès, C.D. Richards, and J. Vouillon. A very modal model of a modern, major, general type system. In *Proceedings of the 34th Symposium on Principles of Programming Languages (POPL'07)*, pages 109–122. ACM, 2007. 14, 23, 39, 71

[Bet59] E. Beth. *The Foundations of Mathematics*. Amsterdam, North-Holland Pub. Co., 1959. 105

[BH09] N. Benton and C.-K. Hur. Biorthogonality, step-indexing and compiler correctness. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*, ICFP '09, pages 97–108, New York, NY, USA, 2009. ACM. 11, 37

[BM13] L. Birkedal and R. Møgelberg. Intensional type theory with guarded recursive types qua fixed points on universes. In *Proceedings of the 28th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS'13)*, pages 213–222. IEEE, 2013. 95, 104, 105

[BMSS11] L. Birkedal, R. Møgelberg, J. Schwinghammer, and K. Støvring. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, pages 55–64. IEEE, 2011. 15, 23, 24, 40, 71, 72, 95, 99, 103, 104

[BT09] N. Benton and N. Tabareau. Compiling functional types to relational specifications for low level imperative code. In *TLDI '09: Proceedings of the 4th international workshop on Types in language design and implementation*, pages 3–14, New York, NY, USA, 2009. ACM. 11, 37

[CD66] P.J. Cohen and M. Davis. *Set theory and the continuum hypothesis*. WA Benjamin New York, 1966. 15, 22, 40, 70

[CG94] P.-L. Curien and G. Ghelli. Theoretical aspects of object-oriented programming. chapter Coherence of Subsumption, Minimum Typing and Typechecking in $F_{\leq}$, pages 247–292. MIT Press, Cambridge, MA, USA, 1994. 58

[CGH14] P.-L. Curien, R. Garner, and M. Hofmann. Revisiting the categorical interpretation of dependent type theory. *Theoretical Computer Science*, pages –, 2014. 58, 75, 76, 91

[CH09] P.-L. Curien and H. Herbelin. Abstract machines for dialogue games. *Panoramas et synthèses-Société mathématique de France*, (27):231–275, 2009. 145

[CJ10] T. Coquand and G. Jaber. A Note on Forcing and Type Theory. *Fundamenta Informaticae*, 100(1):43–52, 2010. 22, 70, 105

[CJ12] T. Coquand and G. Jaber. A computational interpretation of forcing in type theory. In *Epistemology versus Ontology*, pages 203–213. Springer, 2012. 22, 70, 105

[Cur93] P.-L. Curien. Substitution up to isomorphism. *Fundamenta Informaticae*, 19(1/2):51–85, 1993. 58, 91

[Cur98] P.-L. Curien. Abstract böhm trees. *Mathematical Structures in Computer Science*, 8(06):559–591, 1998. 145

[DAB09] D. Dreyer, A. Ahmed, and L. Birkedal. Logical step-indexed logical relations. In *Proceedings of LICS'09*, 2009. 18, 43

[DFH95] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in coq. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Typed Lambda Calculi and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 124–138. Springer Berlin Heidelberg, 1995. 104

[DNB10] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *Proceeding of the 15th Conference on Functional programming, (ICFP'10)*, pages 143–156. ACM, 2010. 18, 30, 31, 43, 159, 161, 191

[DNB12] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. *Journal of Functional Programming*, 22:477–528, 9 2012. 13, 16, 18, 38, 41, 43, 65, 191, 203

[DNRB10] D. Dreyer, G. Neis, A. Rossberg, and L. Birkedal. A relational modal logic for higher-order stateful ADTs. In *Proceedings of POPL'10*, 2010. 15, 18, 32, 40, 43, 194

[Esc13] M. Escardó. Continuity of gödel's system t definable functionals via effectful forcing. *Electronic Notes in Theoretical Computer Science*, 298:119–141, 2013. 106

[FP91] T. Freeman and F. Pfenning. Refinement types for ml. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, PLDI '91, pages 268–277, New York, NY, USA, 1991. ACM. 11, 36

[FPT99] M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, page 193. IEEE Computer Society, 1999. 104

[GG12] M. Gabbay and D. Ghica. Game semantics in the nominal model. *Electronic Notes in Theoretical Computer Science*, 286:173–189, 2012. 154

[GM00] D. Ghica and G. McCusker. Reasoning about idealized algol using regular languages. In *Proceedings of ICALP'00*, pages 103–115, London, UK, UK, 2000. Springer-Verlag. 33, 155, 195

[GP99] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 214–224. IEEE, 1999. 104

[GvdB11] R. Garner and B. van den Berg. Types are weak ω-groupoid. *Proceedings of the London Mathematical Society*, 102(2):370—-394, 2011. 51

[GW08] H. Geuvers and F. Wiedijk. A logical framework with explicit conversions. *Electronic Notes in Theoretical Computer Science*, 199:33–47, 2008. 58

[HDNV12] C.-K. Hur, D. Dreyer, G. Neis, and V. Vafeiadis. The marriage of bisimulations and kripke logical relations. In *POPL '12: Proceedings of the 39th ACM*

Symposium on Principles of Programming Languages, volume 47, pages 59–72. ACM, 2012. 28, 158, 190

[HMO11] D. Hopkins, A. Murawski, and L. Ong. A fragment of ML decidable by visibly pushdown automata. In *Proceedings of the ICALP'11*, pages 149–161. Springer-Verlag, 2011. 33, 155, 195

[HO00] M. Hyland and L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163(2):285–408, December 2000. 25, 108

[Hof95a] M. Hofmann. *Extensional concepts in intensional type theory*. PhD thesis, University of Edinburgh, 1995. 50, 58

[Hof95b] M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *Computer Science Logic*, pages 427–441. Springer, 1995. 91

[Hof97] M. Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997. 20, 48, 49, 50

[Hof99] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, page 204. IEEE Computer Society, 1999. 104

[HS98] M. Hofmann and T. Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998. 51

[HY99] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretical Computer Science*, 221(1):393–456, 1999. 110

[Jec03] T.J. Jech. *Set theory*. Springer Verlag, 2003. 94, 101

[JT10] G. Jaber and N. Tabareau. Krivine realizability for compiler correctness. In *Workshop LOLA 2010, Syntax and Semantics of Low Level Languages*, 2010. 11, 37

[JT11] G. Jaber and N. Tabareau. The journey of biorthogonal logical relations to the realm of assembly code. In *Workshop LOLA 2011, Syntax and Semantics of Low Level Languages*, 2011. 11, 37

[JTS12] G. Jaber, N. Tabareau, and M. Sozeau. Extending Type Theory with Forcing. In *Proceedings of LICS'12*, Dubrovnik, Croatie, June 2012. 95

[Kri09] J.-L. Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009. 22, 70

[Kri11] J.L. Krivine. Realizability algebras: a program to well order $\mathbb{R}$. *Logical Methods in Computer Science*, 7(3):1–47, 2011. 22, 70, 105, 106

[Lai07] J. Laird. A fully abstract trace semantics for general references. In *Proceedings of the ICALP'07*, pages 667–679. Springer, 2007. 17, 25, 27, 42, 108, 110, 120, 140, 212

[Lai08] J. Laird. A game semantics of names and pointers. *Annals of Pure and Applied Logic*, 151(2):151–169, 2008. 25, 26, 108, 109, 116, 117, 118

[Lai10] J. Laird. Game semantics for call-by-value polymorphism. In *Proceedings of ICALP'10*, pages 187–198. Springer-Verlag, 2010. 154

[Ler09] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009. 11, 37

[Lev77] A.M. Levin. One conservative extension of formal mathematic analysis with a scheme of dependent choice. *Mathematical Notes*, 22:524–528, 1977. 106

[LL07] S. Lassen and P. Levy. Typed normal form bisimulation. In *Computer Science Logic*, pages 283–297. Springer, 2007. 164

[LL08] S. B Lassen and P. Levy. Typed normal form bisimulation for parametric polymorphism. In *Logic in Computer Science, 2008. LICS'08. 23rd Annual IEEE Symposium on*, pages 341–352. IEEE, 2008. 164

[Loa01] R. Loader. Finitary pcf is not decidable. *Theoretical Computer Science*, 266(1):341–364, 2001. 12, 33, 37, 195

[LS14] P. Levy and . Staton. Transition systems over games. In *Proceedings of LICS'14*, pages 64:1–64:10, 2014. 154

[Lum09] P. Lumsdaine. Weak ω-categories from intensional type theory. In P.-L. Curien, editor, *Typed Lambda Calculi and Applications*, volume 5608 of *Lecture Notes in Computer Science*, pages 172–187. Springer Berlin Heidelberg, 2009. 51

[McB00] C. McBride. *Dependently typed functional programs and their proofs*. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics., 2000. 58

[Miq11] A. Miquel. Forcing as a program transformation. In *Proceedings of the 26th Symposium on Logic in Computer Science (LICS'11)*, pages 197–206. IEEE, 2011. 22, 70, 90, 105

[ML73] P. Martin-Löf. An intuitionistic theory of types: predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium*, pages 73–118. North-Holland, 1973. 50, 51

[MLM92] S. Mac Lane and I. Moerdijk. *Sheaves in geometry and logic: A first introduction to topos theory*. Springer, 1992. 72, 94, 101, 105

[MLS84] P. Martin-Löf and G. Sambin. *Intuitionistic type theory*, volume 17. Bibliopolis Naples, 1984. 49, 51

[Møg] R. Møgelberg. A type theory for productive coprogramming via guarded recursion. 104

[Mog91] E. Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991. 110

[MT11a] A. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *Proceedings of the 20th European Symposium on Programming, (ESOP'11)*, volume 6602, page 419. Springer, 2011. 19, 33, 44, 140, 195

[MT11b] A. Murawski and N. Tzevelekos. Game semantics for good general references. In *Proceedings of LICS'11*, pages 75–84. IEEE, 2011. 17, 25, 26, 27, 42, 108, 109, 110, 118, 119, 130, 132, 133, 135, 140, 154, 212

[MT12] A. Murawski and N. Tzevelekos. Algorithmic games for full ground references. In *Proceedings of the ICALP'12*, pages 312–324, Berlin, Heidelberg, 2012. Springer-Verlag. 19, 25, 33, 44, 108, 144, 155, 195

[MT13] A. Murawski and N. Tzevelekos. Full abstraction for reduced ml. *Annals of Pure and Applied Logic*, 2013. 154, 192

[Mur05] A. Murawski. Functions with local state: regularity and undecidability. *Theoretical Computer Science*, 338(1):315–349, 2005. 19, 44

[Nak00] H. Nakano. A modality for recursion. In *Proceeding of the 15th Symposium on Logic in Computer Science (LICS'00)*, pages 255–266. IEEE, 2000. 14, 23, 40, 71

[Nak01] H. Nakano. Fixed-point logic with the approximation modality and its kripke completeness. In *Theoretical Aspects of Computer Software*, pages 165–182. Springer, 2001. 14, 40

[NMB08] A. Nanevski, J. Morrisett, and L. Birkedal. Hoare type theory, polymorphism and separation. *J. Funct. Program.*, 18(5-6):865–911, 2008. 11, 36

[Our05] N. Oury. Extensionality in the calculus of constructions. In *Proceedings of the 18th Conference on Theorem Proving in Higher Order Logics (TPHOLs'05)*, pages 278–293. Springer, 2005. 58

[PA93] G. Plotkin and M. Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications*, 1993. 18, 43

[PE88] F. Pfenning and C. Elliot. Higher-order abstract syntax. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, PLDI '88, pages 199–208, New York, NY, USA, 1988. ACM. 104

[Pit00] A. Pitts. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science*, 10(3):321–359, 2000. 16, 41

[Pit03] A. Pitts. Nominal logic, a first order theory of names and binding. *Information and computation*, 186(2):165–193, 2003. 21, 25, 66, 67, 108

[Pit13] A. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57. Cambridge University Press, 2013. 17, 42

[Pnu77] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977. 32, 194

[PR97] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical structures in computer science*, 7(05):453–468, 1997. 26, 109, 110

[PS98] A. Pitts and i. Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*. CUP, 1998. 13, 16, 18, 30, 38, 41, 43, 61, 159, 202

[PT99] J. Power and H. Thielecke. Closed freyd-and kappa-categories. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, pages 625–634. Springer-Verlag, 1999. 26, 109, 110, 118

[Rey78] John C Reynolds. Syntactic control of interference. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 39–46. ACM, 1978. 118

[Rey83] J. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP Congress*, pages 513–523, 1983. 12, 38, 212

[Rey02] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, 2002*, pages 55–74. IEEE, 2002. 18, 43

[Sco82] D. Scott. Domains for denotational semantics. In *Automata, languages and programming*, pages 577–610. Springer, 1982. 13, 39

[SH12] V. Siles and H. Herbelin. Pure type system conversion is always typable. *Journal of Functional Programming*, 22(02):153–180, 2012. 50

[Shu12] M. Shulman. Univalence for inverse diagrams, oplax limits, and gluing, and homotopy canonicity. *arXiv preprint arXiv:1203.3253*, 2012. 105

[Shu13] M. Shulman. The univalence axiom for elegant reedy presheaves. *arXiv preprint arXiv:1307.6248*, 2013. 105

[SKS11] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(1):5, 2011. 28, 157

[SS71] D. Scott and C. Strachey. Toward A Mathematical Semantics for Computer Languages. In *Proceedings of the Symposium on Computers and Automata*, volume XXI, pages 19–46. Polytechnic Press, 1971. 13, 39

[Sta98] I. Stark. Names, equations, relations: Practical ways to reason about new. *Fundamenta Informaticae*, 33(4):369–396, 1998. 29, 67, 159

[Sum09] E. Sumii. A complete characterization of observational equivalence in polymorphic λ-calculus with general references. In *Computer Science Logic '09*, pages 455–469. Springer, 2009. 28, 157

[Tie72] M. Tierney. Sheaf theory and the continuum hypothesis. *LNM 274*, pages 13–42, 1972. 15, 22, 40, 71

[Tze07] N. Tzevelekos. Full abstraction for nominal general references. In *Logic in Computer Science, 2007. LICS 2007. 22nd Annual IEEE Symposium on*, pages 399–410. IEEE, 2007. 25, 108, 119

[Tze08] N. Tzevelekos. *Nominal game semantics*. PhD thesis, University of Oxford, 2008. 66, 119

[Tze11] N. Tzevelekos. Fresh-register automata. In *Proceedings of POPL'11*, pages 295–306. ACM, 2011. 33, 195

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013. 20, 48, 49, 50, 51, 52, 53, 59, 91, 94

[vDGW13] F. van Doorn, H. Geuvers, and F. Wiedijk. Explicit convertibility proofs in pure type systems. In *Proceedings of the Eighth ACM SIGPLAN international workshop on Logical frameworks & meta-languages: theory & practice*, pages 25–36. ACM, 2013. 58

[XE13] C. Xu and M. Escardó. A constructive model of uniform continuity. *Typed Lambda Calculi and Applications LNCS 7941*, pages 236–249, 2013. 106

[YCER11] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in c compilers. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 283–294, New York, NY, USA, 2011. ACM. 11, 37

Index

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

MINES Nantes
Institut Mines-Télécom

Thèse de Doctorat

Guilhem JABER

Une étude logique de l'équivalence de programmes

A Logical Study of Program Equivalence

Résumé

Prouver l'équivalence de programmes écrits dans un langage fonctionnel avec références est un problème notoirement difficile. L'objectif de cette thèse est de proposer un système logique dans lequel de telles preuves peuvent être formalisées, et dans certains cas inférées automatiquement.
Dans la première partie, une méthode générique d'extension de la théorie des types dépendants est proposée, basée sur une interprétation du forcing vu comme une traduction de préfaisceaux de la théorie des types. Cette extension dote la théorie des types de constructions récursives gardées, qui sont utilisées ensuite pour raisonner sur les références d'ordre supérieure.
Dans une deuxième partie, nous définissons une sémantique des jeux nominale opérationnelle pour un langage avec références d'ordre supérieur. Elle marie la structure catégorique de la sémantique des jeux avec une représentation sous forme de traces de la dénotation des programmes, qui se calcule de manière opérationnelle et dispose donc de bonnes propriétés de modularité. Cette sémantique nous permet ensuite de prouver la complétude de relations logiques à la Kripke définit de manière directe, via l'utilisation de types récursifs gardés, sans utilisation de la biorthogonalité. Une telle définition directe nécessite l'utilisation de mondes omniscient et un contrôle fin des locations divulguées.
Finalement, nous introduisons une logique temporelle qui donne un cadre pour définir ces relations logiques à la Kripke. Nous ramenons alors le problème de l'équivalence contextuelle à la satisfiabilité d'une formule de cette logique générée automatique, c'est à dire à l'existence d'un monde validant cette formule. Sous certaines conditions, cette satisfiabilité peut être décidée via l'utilisation d'un solveur SMT. La complétude de notre méthode devrait permettre d'obtenir des résultats de décidabilité pour l'équivalence contextuelle de certains fragment du langage considéré, en fournissant un algorithme pour construire de tels mondes.

Mots clés

Théorie des Types, Forcing, Equivalence Contextuelle, Relation Logique, Sémantique des Jeux, Logique Temporelle.

Abstract

Proving program equivalence for a functional language with references is a notoriously difficult problem. The goal of this thesis is to propose a logical system in which such proofs can be formalized, and in some cases inferred automatically.
In the first part, a generic extension method of dependent type theory is proposed, based on a forcing interpretation seen as a presheaf translation of type theory. This extension equips type theory with guarded recursive constructions, which are subsequently used to reason on higher-order references.
In the second part, we define a nominal game semantics for a language with higher-order references. It marries the categorical structure of game semantics with a trace representation of denotations of programs, which can be computed operationally and thus have good modularity properties.
Using this semantics, we can prove the completeness of Kripke logical relations defined in a direct way, using guarded recursive types, without using biorthogonality. Such a direct definition requires omniscient worlds and a fine control of disclosed locations.
Finally, we introduce a temporal logic which gives a framework to define these Kripke logical relations. The problem of contextual equivalence is then reduced to the satisfiability of an automatically generated formula defined in this logic, i.e. to the existence of a world validating this formula. Under some conditions, this satisfiability can be decided using a SMT solver. Completeness of our methods opens the possibility of getting decidability results of contextual equivalence for some fragments of the language, by giving an algorithm to build such worlds.

Key Words

Type Theory, Forcing, Contextual Equivalence, Kripke Logical Relation, Game Semantics, Temporal Logic.