Département de formation doctorale en informatique **École doctorale IAE + M**
UFR STMIA

Preuves par récurrence avec ensembles couvrants contextuels. Applications à la vérification de logiciels de télécommunications

THÈSE

présentée et soutenue publiquement le **30 novembre 2000**

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Sorin Stratulat

Composition du jury

Président :	Gilles Bernot	Professeur, Université d'Evry, Evry
Rapporteurs :	Gilles Bernot	Professeur, Université d'Evry, Evry
	Fausto Giunchiglia	Professeur, Université de Trente, Italie
	Claude Godart	Professeur, Université Henri Poincaré, Nancy I
Examinateurs :	Alessandro Armando	Professeur, Université de Gênes, Italie
	Francis Klay	Dr. Ingénieur R & D France–Télécom, Lannion
	Michaël Rusinowitch	Directeur de Recherche, INRIA, Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

C'est un grand plaisir pour moi de remercier toutes les personnes qui, de près ou de loin, m'ont permis de mener à bien cette thèse.

Je remercie tout particulièrement Michaël Rusinowitch, directeur de recherche à l'INRIA Lorraine, qui m'a encadré et aidé tout au long de mon travail de thèse par ses conseils, commentaires et critiques, mais aussi par sa permanente disponibilité, ses encouragements, sa gentillesse et son énorme patience vis-à-vis d'un étudiant souvent persévérant dans son obstination. Je tiens à lui exprimer ma profonde gratitude.

Mes remerciements s'adressent ensuite aux personnes qui ont accepté d'être membre de mon jury. Tous m'ont fait un grand honneur en s'intéressant à mon travail :

- Gilles Bernot, professeur à l'Université d'Evry, qui m'a fait l'honneur de présider ce jury ainsi que d'être rapporteur de ma thèse.
- Fausto Giunchiglia, professeur à l'Université de Trente, Italie, qui a accepté la lourde tâche de rédiger un rapport, malgré l'obstacle de la langue.
- Claude Godart, professeur à l'Université Henri Poincaré, Nancy I, qui a rempli sa mission de rapporteur interne avec beaucoup d'attention.
- Alessandro Armando, professeur à l'Université de Gênes, Italie. Je le remercie sincèrement pour les nombreuses discussions que nous avons eues ensemble, ses commentaires fort judicieux et ses conseils qui m'ont été extrêmement bénéfiques.
- Francis Klay, Dr. Ingénieur R & D France Télécom, Lannion, qui m'a fait l'honneur d'être examinateur de ma thèse. Les études de cas qu'il a proposées lors d'un contrat avec notre équipe ont été le fil directeur de mes recherches. Qu'il trouve ici l'expression de ma sincère gratitude pour cette collaboration fructueuse et ses conseils avisés.

Mes remerciements s'adressent également à Adel Bouhoula pour ses suggestions et conseils pendant mon initiation dans l'utilisation du démonstrateur SPIKE.

Je remercie également tous les membres de l'équipe PROTHEO pour l'accueil chaleureux, plus particulièrement Claude et Hélène Kirchner, les directeurs de l'équipe pendant ma thèse, qui m'ont assuré les moyens pour poursuivre mon travail.

Je tiens à remercier France Télécom, R & D, et la Région Lorraine pour leur soutien financier pendant les trois premières années de thèse.

Je n'oublierai pas de remercier mes collègues de bureau Christophe et Raulent, ainsi que Laurent, Horatiu et Hubert pour les bons moments passés ensemble. Je les ai senti toujours proche de moi et prêts à m'aider avec leur conseils. Leurs commentaires et suggestions m'ont beaucoup aidé pendant la rédaction de la thèse.

Même si le contenu de cette thèse a été ébauché pendant les cinq dernières années, je considère qu'elle représente le point culminant d'une période beaucoup plus grande. Ainsi, je profite de cette occasion pour remercie tous mes professeurs de Roumanie pour la qualité de la formation qu'ils m'ont offerte. En particulier, j'exprime ma gratitude à Cornel Popa, professeur de logique à l'Université « Politehnica » de Bucarest qui m'a « inoculé le virus » de la recherche et pour qui je vais toujours garder une place spéciale dans mon cœur.

Je remercie également Sandrine d'avoir partagé avec moi les bonheurs et les malheurs d'une vie de thésard et d'avoir eu la patience de corriger les erreurs de langue du manuscript.

Enfin, je remercie mes proches, ma famille et mes amis, pour leurs encouragements et leur soutien amical.

A mes proches,
nées le 8 février

Table des matières

Partie II Applications à la vérification de logiciels de télécommunications

Chapitre 7
Analyse des interactions de services téléphoniques

Chapitre 8
Preuve de la conformité du protocole ABR

Table des figures

Introduction

Depuis plus de trente ans, l'informatique a un impact socio-économique de plus en plus important sur notre société. Le développement récent et sans précédent de nouvelles technologies et des télécommunications a banalisé son utilisation dans notre vie courante. Dans le domaine des télécommunications, la transmission à haut débit† et le traitement rapide de l'information permettent la gestion des réseaux téléphoniques complexes qui englobent de nombreux services critiques proposés aux utilisateurs, que ce soit pour effectuer des transactions bancaires à domicile ou pour acheter aux enchères par Internet.

Afin de garantir le bon fonctionnement d'un réseau téléphonique, tout nouveau service ou protocole de communication doit être certifié avant d'être mis en pratique. Il est essentiel pour l'opérateur de justifier que les produits qu'il offre aux consommateurs n'affectent pas sa sécurité ou la qualité du service (QoS) fournie par le réseau. La raison en est que tout dysfonctionnement peut avoir des conséquences économiques importantes. Nous pouvons ainsi citer en exemple la fermeture temporaire en février 2000 du fournisseur d'accès à Internet AOL (America OnLine), comptant 23 millions d'abonnés, à cause des attaques de pirates informatiques.

Le processus de certification de logiciels est, dans la plupart des cas, une tâche laborieuse et coûteuse qui nécessite des outils automatiques et des méthodes mathématiques adéquates à la vérification d'un nombre (potentiellement infini) de cas à tester. Dans cette direction, les méthodes formelles sont un support théorique idéal aussi bien pour spécifier la description du comportement attendu du logiciel que pour prouver certaines de ses propriétés.

Contrairement aux méthodes de spécifications informelles, comme le cahier des charges écrit en langage naturel, les méthodes formelles fournissent des modèles mathématiques qui permettent de s'exprimer sans ambiguïté et de façon structurée. Les propriétés à tester sont vérifiées par l'application de suites d'opérations logiques sur un univers du discours défini rigoureusement. Il existe deux approches répandues à la preuve de logiciels. Celle qui est basée sur le « model checking » [Vardi et Wolper, 1986; Burch *et al.*, 1990] vérifie de manière complètement automatique et efficace des systèmes d'états finis souvent grâce au calcul booléen par des diagrammes de décision binaires. Les inconvénients majeurs de la méthode classique de model checking sont sa limitation à certaines classes de systèmes et l'apparition fréquente de problèmes d'explosion combinatoire. La deuxième approche vise la résolution des problèmes plus difficiles, concernant la vérification des systèmes d'états infinis, par des démonstrateurs fondés sur des logiques expressives, comme COQ [Barras *et al.*, 1997], HOL [Gordon et Melham, 1993] ou PVS [Owre *et al.*, 1992]. Généralement, ils nécessitent une grande interaction avec l'utilisateur afin de guider la construction des preuves. L'utilisateur doit être à la fois un bon connaisseur du problème traité et aussi un expert de l'outil, capable de comprendre son fonctionnement et sa logique

† offerte par les réseaux de télécommunications de type ATM (Asynchronous Transfer Mode) ou par les futurs réseaux de téléphonie mobile basés sur la norme de transmission sans fil UMTS (Universal Mobile Telecommunications System).

sous-jacente. La logique du premier ordre est souvent utilisée puisqu'elle est suffisamment expressive comme langage de spécification et permet de minimiser l'interaction de l'utilisateur avec l'environnement de preuve.

Parmi les techniques de validation formelle, les *spécifications algébriques* [Wirsing, 1990] modélisent la spécification informelle de départ en termes d'objets et d'opérations agissant sur ces objets, à l'aide des *types abstraits de données* [Guttag, 1975]. Selon la sémantique utilisée pour interpréter une spécification algébrique, il y a deux notions de validité très répandues : i) *déductive*, lorsque les propriétés sont prouvées uniquement par des successions de remplacements d'égaux par égaux, et ii) *initiale*, lorsque la vérification se fait dans un modèle particulier, le modèle initial, bien adapté aux spécifications contenant des constructions récursives.

Pour prouver des propriétés liées à une spécification particulière, nous avons besoin d'un *modèle de calcul* pour les inférer, caractérisé par des *techniques de preuve* adéquates. Par exemple, une certaine classe de spécifications algébriques, représentée par les *spécifications conditionnelles*, utilise la *réécriture* comme modèle de calcul. L'ensemble de règles de réécriture conditionnelle, définissant une relation de réécriture pour une spécification conditionnelle donnée, se construit à partir des axiomes représentant des équations conditionnelles, c'est-à-dire des équations (appelées des conclusions) ayant un ensemble d'autres équations comme conditions. Pour cela, on oriente chaque conclusion, en principe de gauche à droite. La sémantique opérationnelle associée à chaque règle de réécriture conditionnelle spécifie qu'on peut toujours remplacer pendant une preuve, par exemple, le membre gauche de la conclusion par le membre droit lorsque les conditions sont satisfaites.

Techniques de récurrence

Parmi les techniques de preuve actuelles, la récurrence s'avère être une méthode puissante pour raisonner sur des structures récursives, adéquate en particulier pour les preuves de logiciels comportant des structures de données infinies (entiers, listes, ...), avec contrôle infini (buffer non-borné), ou qui doivent opérer sur des topologies arbitraires (graphe de communication). Elle demande que les éléments du domaine de raisonnement soient ordonnés par des *ordres bien fondés*, c'est-à-dire qui n'admettent pas de suites décroissantes infinies d'éléments. Inventée par les anciens grecs et redécouverte par Pascal (1623-1662) en 1654, le *principe de récurrence* est utilisé, dans sa forme la plus connue †, pour montrer qu'une propriété P est vraie sur les naturels :

$P(0)$	*si P est vraie pour le naturel* 0*, et*
$\forall x \geq 0,\ P(x) \Rightarrow P(x+1)$	*si P est vraie pour un x arbitraire*
	P est aussi vraie pour $x+1$,
$\forall x,\ P(x)$	*alors P est vraie pour tout naturel.*

Plusieurs démonstrateurs de théorèmes, comme NQTHM [Boyer et Moore, 1979], CLAM [Bundy *et al.*, 1989], RRL [Kapur et Zhang, 1989] ou PVS, utilisent la récurrence structurelle [Boyer et Moore, 1979; Aubin, 1979; Walther, 1993] dont les schémas de récurrence se fondent sur des ordres construits à partir de la structure syntaxique des éléments du domaine de raisonnement. La récurrence est *explicite* puisque les prémisses et les conclusions peuvent être distinguées pendant la preuve. La correction des schémas de récurrence est assurée par l'existence des fonctions de terminaison représentant des ordres bien fondés.

† nommé aussi le *principe de récurrence de Peano.*

D'autres démonstrateurs utilisent un cadre plus général de la récurrence explicite, à savoir la *récurrence noethérienne* [Padawitz, 1992]. Supposant l'existence d'un ordre bien fondé $<_D$ sur (les éléments d'un domaine) D, le principe de récurrence noethérienne se définit comme dans le schéma suivant :

$$\frac{1.\ \forall m \in D,\ (\forall k \in D,\ (k <_D m) \Rightarrow P(k)) \Rightarrow P(m)}{2.\ \forall n \in D,\ P(n)}$$

Supposons que pour tout élément m de D, si $P(k)$ est vraie pour tout élément k inférieur à m alors $P(m)$ est vraie. Alors P est vraie pour tout élément de D.

Une autre forme de récurrence a été redécouverte par Fermat (1601-1665) en 1659 [Wirth, 2000], qui l'a baptisée *descente infinie* ou indéfinie. Elle apparaît en déduction automatique dans le cadre des techniques de *récurrence implicite*. Sa formalisation peut s'obtenir à partir du schéma de récurrence noethérienne, en utilisant la démonstration par l'absurde et la contraposition : i) on considère comme hypothèse la négation de la conclusion (ou conjecture) 2. du schéma, ii) on contrapose l'hypothèse 1. et iii) on déduit une contradiction qui dérive généralement de l'emploi d'ordres bien fondés :

$$\frac{\begin{array}{l}1.\ \forall m \in D,\ \neg P(m) \Rightarrow (\exists k \in D,\ (k <_D m) \wedge \neg P(k)) \\ 2.\ \exists n \in D,\ \neg P(n)\end{array}}{\text{Contradiction}}$$

La méthode est applicable lorsqu'on peut prouver que, pour tout contre-exemple de la conjecture, il existe un autre contre-exemple inférieur à lui, par rapport à un ordre bien fondé. Par exemple, cette méthode nous permet de prouver que $\sqrt{p}$ est irrationnel, pour tout nombre premier p. Autrement dit, il faut démontrer la conjecture suivante : pour tout m, $n \in \mathsf{N}^*$, on a $\sqrt{p} \neq m/n$. Pour appliquer le schéma de la récurrence implicite, on suppose son contraire : il existe m, $n \in \mathsf{N}^*$ tels que $\sqrt{p} = m/n$, ou bien $m^2 = p * n^2$. Ceci implique qu'il existe un nombre $m_1 \in \mathsf{N}^*$ (inférieur à m) tel que $m = p * m_1$. Par conséquent, on obtient $p * m_1^2 = n^2$. Pour qu'elle soit satisfaite, cette relation demande l'existence d'un nombre $n_1 \in \mathsf{N}^*$ (inférieur à n) tel que $n = p * n_1$. Comme la conjecture est aussi infirmée par m_1 et n_1, on peut appliquer le même raisonnement sur m_1 et n_1. On déduit qu'il y a une suite infinie décroissante des valeurs naturelles, ce qui est une contradiction.

Les méthodes de preuve par récurrence implicite dérivent de l'approche de *preuve par cohérence*, décrite pour la première fois par Musser [Musser, 1980] et raffinée ensuite dans [Huet et Hullot, 1982; Jouannaud et Kounalis, 1989; Fribourg, 1989; Bachmair, 1988; Küchlin, 1989]. Basée sur la réécriture et sur des procédures de déduction comme la complétion de Knuth-Bendix, [Knuth et Bendix, 1970], elle vérifie automatiquement la cohérence d'une conjecture par rapport à une spécification équationnelle. La récurrence est justifiée par un ordre bien fondé implicite induit par les règles de réécriture. Pendant le processus de complétion, on ne mentionne pas les prémisses et les conclusions † et il n'y a pas de hiérarchie dans la construction des lemmes. Un des avantages de la méthode est d'effectuer naturellement la *récurrence mutuelle*, autrement dit, un théorème peut contribuer directement à prouver un lemme et vice-versa. Pourtant, les scénarios de preuves sont difficilement compréhensibles [Garland et Guttag, 1988] et, en cas d'échec de la preuve, l'utilisateur n'est pas en mesure de savoir si les conjectures initiales ont été réfutées ou pas.

† C'est la raison pour laquelle ce type de récurrence s'appelle *implicite*.

La récurrence par ensembles couvrants conserve les avantages de la récurrence explicite et de la preuve par cohérence ; (i) l'ordre global bien fondé sur des formules générales est construit explicitement et peut contenir des ordres arbitraires bien fondés, et (ii) elle permet la récurrence mutuelle. Les simplifications de conjectures sont guidées par l'ordre global comme dans l'approche des preuves par cohérence. La correction du processus de déduction est assurée lorsqu'une conjecture est simplifiée par des instances plus petites de prémisses et d'autres conjectures. Une grande souplesse est permise dans le calcul des schémas de récurrence explicite, nommés *ensembles couvrants*. Des démonstrateurs de théorèmes basés sur la récurrence avec ensembles couvrants, comme SPIKE [Bouhoula et Rusinowitch, 1995b], Focus [Bronsard et Reddy, 1991], RRL ou QuodLibet [Kühler, 1999] ont montré l'intérêt de la récurrence implicite pour minimiser l'interaction de l'utilisateur avec le démonstrateur [Bouhoula et Rusinowitch, 1995a; Bronsard *et al.*, 1996]. Plusieurs définitions d'ensembles couvrants et différentes méthodes de récurrence implicite ont été présentées dans [Kounalis et Rusinowitch, 1990; Reddy, 1990; Bouhoula et Rusinowitch, 1995a; Naidich, 1996; Bronsard *et al.*, 1996; Wirth, 1997].

Contributions

I. Preuves par récurrence avec ensembles couvrants contextuels

Ensembles couvrants contextuels Dans ce travail, nous introduisons le concept d'*ensemble couvrant contextuel* comme une généralisation de celui d'ensemble couvrant engendré dans un contexte constitué par des instances de conjectures et de prémisses (par rapport à la récurrence) caractérisant une étape de preuve particulière. Les ensembles couvrants usuels sont, dans notre approche, des ensembles couvrants contextuels ayant un contexte vide. Il y a deux raisons pour manipuler des ensembles couvrants contextuels ayant un contexte *maximal*. D'une part, le nombre de choix pour l'élimination/simplification d'une conjecture peut augmenter et, par conséquent, le nombre de conjectures prouvable également. D'autre part, les éléments du contexte peuvent contribuer à la construction des ensembles couvrants. De ce point de vue, un contexte maximal serait équivalent à un ensemble maximal d'ensembles couvrants disponibles à cette étape de preuve. Or, le choix de l'ensemble couvrant est crucial pour le succès global de la preuve : si le contexte n'est pas maximal, il est alors possible que le bon ensemble couvrant soit indisponible, ce qui peut entraîner un échec global de la preuve.

Règles d'inférence abstraites Un objectif majeur de la démonstration automatique est de proposer des méthodologies et des cadres de travail pour construire des outils de preuve. La similarité de la plupart des procédures de preuves basées sur la récurrence implicite suggère l'existence d'un cadre unificateur qui sépare la composante logique de l'implantation, permettant ainsi leur comparaison et leurs extensions modulaires, ou des généralisations et des modifications faciles. Les différents systèmes d'inférence, proposés dans [Bronsard *et al.*, 1996; Naidich, 1996; Bouhoula, 1997; Wirth, 1997], sont constitués par des règles d'inférence génériques et déclaratives, qui montrent *quelle* information peut être utilisée correctement pendant le processus de raisonnement. Dans ce cadre, les règles d'inférence des procédures de preuve existantes illustrent *comment* les règles abstraites sont mises en œuvre. Nous proposons un système d'inférence abstrait dont les règles d'inférence sont formulées uniformément en termes d'ensembles couvrants contextuels, et qui établit leurs contextes maximaux. De plus, les règles peuvent être conçues de manière modulaire grâce à la possibilité de composer les ensembles couvrants contextuels. Le

principe de la récurrence implicite s'applique puisque i) le traitement de toute conjecture contenant un contre-exemple par une règle d'inférence garantit l'existence d'une autre conjecture ayant un contre-exemple inférieur et ii) l'ordre sur les conjectures est bien fondé.

Nous montrons que notre système d'inférence ne peut pas prouver des conjectures fausses (correction) et que toute réfutation d'une conjecture à une étape intermédiaire d'une preuve permet de réfuter l'ensemble de conjectures initiales (correction réfutationnelle). De plus, on a établi des conditions suffisantes pour que toute conjecture fausse soit réfutée (complétude réfutationnelle).

Généralement, pour obtenir un outil de preuve efficace et performant, il est crucial de décomposer les problèmes et de localiser les preuves qui peuvent être traitées séparément. En fonction de la spécificité d'un sous-problème, il faut faire appel à des algorithmes de décision adaptés, par exemple, des procédures de décision sur des domaines particuliers, comme l'arithmétique linéaire, les listes, les vecteurs de bits ou les tableaux. Mais, l'intégration de différents modules de raisonnement dans un démonstrateur s'avère difficile. Boyer et Moore [Boyer et Moore, 1985] ont montré, suite à de longues expérimentations, que l'utilisation des procédures de décision comme des « boites noires » est insuffisante, et des résultats plus intéressants peuvent être obtenus par une interopérabilité étroite. Pourtant, celle-ci est difficile à comprendre et pose de délicats problèmes de communication. Il existe plusieurs schémas d'interopérabilité, que ce soit pour faire *coopérer* des procédures de décision, ou pour *intégrer* dans un démonstrateur des classes particulières de systèmes de raisonnement comme les OMRS (Open Mechanized Reasoning Systems) [Giunchiglia *et al.*, 1994].

Coopération de procédures de décision Selon [Shostak, 1984; Nelson et Oppen, 1979], on peut induire des techniques de raisonnement puissantes par la *coopération* entre différentes procédures de décision, comme celles pour l'arithmétique linéaire et pour la théorie de l'égalité (clôture par congruence). Nous concevons un schéma formalisant leur coopération dans le cadre des preuves par ensembles couvrants contextuels. L'échange d'informations entre les procédures de décision est décrit par des règles d'inférence spécifiques. Grâce à elles, nous montrons que le schéma est correct et que cette combinaison définit une procédure de décision.

Schéma d'intégration des modules de raisonnement Dans notre cadre de travail, les ensembles couvrants contextuels élémentaires sont engendrés par des *modules de raisonnement* mettant en œuvre des techniques de raisonnement utilisées par le démonstrateur, comme le raisonnement par cas, la subsomption, des techniques de réécriture ou des procédures de décision. Selon la technique de raisonnement, il y a des cas où la génération d'un ensemble couvrant contextuel est dépendante des conditions vérifiées par raisonnement inductif. Dans ce cadre, on dit qu'un module de raisonnement, mettant en œuvre une telle technique, est *intégré* dans le système d'inférence si ses conditions sont vérifiées par un appel récursif au démonstrateur. Par conséquent, ceci crée une dépendance mutuelle et forte entre le démonstrateur et le module de raisonnement. D'autre part, pendant la preuve des conditions, nous montrons que ce schéma d'intégration présente l'avantage d'utiliser des éléments du contexte courant en tant que prémisses.

Cadre formel pour la représentation et la conception des systèmes d'inférence basés sur la récurrence implicite Le système d'inférence abstrait intégrant des modules de

raisonnement est suffisamment général pour instancier les systèmes d'inférence de la plupart des démonstrateurs actuels, basés sur la récurrence implicite. La méthodologie d'instanciation est relativement simple : 1) on détecte dans un premier temps les techniques de raisonnement employées par le démonstrateur ; 2) on vérifie si elles sont suffisamment puissantes pour engendrer des ensembles couvrants contextuels élémentaires. Leurs propriétés de composition permettent de construire des ensembles couvrants contextuels plus complexes ; 3) sur cette base, on définit finalement les règles d'inférence concrètes comme des instances de celles du système abstrait.

Certains avantages, pouvant être facilement obtenus grâce à ce résultat d'instanciation, sont : i) la conclusion immédiate que le démonstrateur est correct, réfutationnellement correct et réfutationnellement complet, ii) l'expansion automatique de certains contextes vers les contextes maximaux permis par le système abstrait, iii) des extensions *modulaires* et *incrémentales* du système d'inférence par l'addition des modules de raisonnement mettant en œuvre de nouvelles techniques de raisonnement, et iv) la conception de nouvelles règles d'inférence basées sur des ensembles couvrants contextuels obtenus par la composition des ensembles déjà existants ou engendrés avec de nouveaux modules de raisonnement.

Comme étude de cas, nous représentons le système d'inférence de SPIKE en termes d'ensembles couvrants contextuels, en tant qu'instance du système abstrait d'inférence proposé.

Nouvelles techniques de raisonnement Différentes procédures de preuve basées sur la récurrence implicite [Gramlich, 1989; Bevers et Lewi, 1990; Bouhoula et Rusinowitch, 1995a; Naidich, 1996] utilisent intensivement (des variantes de) la technique de subsomption afin d'éliminer des informations redondantes rencontrées pendant les preuves et d'éviter ainsi leur divergence. Nous définissons une nouvelle technique de subsomption, adaptée au raisonnement inductif. Le système d'inférence de SPIKE étendu avec des règles d'inférence basées sur cette technique et sur une procédure de décision pour l'arithmétique linéaire, a été utilisé pour montrer la correction de l'algorithme MJRTY [Boyer et Moore, 1991] par une combinaison de raisonnement inductif et arithmétique.

Schéma de parallélisation Une amélioration importante apportée à une preuve est la réduction de son temps d'exécution par l'identification de ses parties indépendantes qui peuvent ainsi s'exécuter en même temps. Nous proposons une interface parallèle pour le système d'inférence de SPIKE. Elle met en œuvre un schéma de parallélisation au niveau des conjectures qui principalement distribue les conjectures courantes aux processeurs, crée un processus SPIKE pour traiter chacune d'entre elles, analyse leurs traces de preuves et interprète leurs résultats.

II. Applications à la vérification de logiciels de télécommunications

Notre méthode de preuve par récurrence avec des ensembles couvrants contextuels a été utilisé, de manière indirecte et par l'intermédiaire du démonstrateur SPIKE, à la vérification de quelques études de cas proposées dans le cadre d'un contrat avec le CNET, France-Télécom [groupe PROTHEO, 1996 a 1998].

Méthodologie de détection/résolution des interactions de services téléphoniques Nous proposons une méthodologie qui permet de détecter et de résoudre *off-line* des interactions de services téléphoniques du point de vue de l'utilisateur avec des techniques basées sur

la réécriture conditionnelle et la récurrence implicite. Elle repose sur une vue fonctionnelle et globale du réseau téléphonique qui se modifie constamment par des séquences de commandes. Les interactions sont détectées aussi bien au niveau du comportement des services, qu'au niveau de leurs propriétés.

Vérification automatisée d'un algorithme de conformité du protocole ABR† Le protocole ABR est utilisé pour régler les débits optimaux de transfert d'information dans les réseaux de haut débit, comme les réseaux ATM. Il garantit un débit minimal aux utilisateurs et peut augmenter de manière dynamique selon les ressources disponibles dans le réseau. Afin qu'ABR acquiert une grande flexibilité, l'opérateur doit maîtriser le débit courant en temps-réel pour éviter la congestion du réseau et pour assurer qu'elle est consistante avec le débit minimal admis, ce qu'on appelle la *vérification (ou le contrôle) de la conformité*. De plus, il doit justifier que les algorithmes employés, mettant en œuvre le contrôle de la conformité, n'affectent pas la qualité du service fournie par les réseaux.

Nous avons dérivé la première preuve automatisée d'un algorithme incrémental et idéal qui vérifie la conformité du protocole ABR. Une preuve « à la main » assez complexe a été décrite auparavant dans [Rabadan et Klay, 1997]. Pourtant, ce type de preuves manuelles n'est pas convaincant en général car des cas limites ou des arguments apparemment triviaux sont souvent omis, ceci constituant des sources permanentes d'erreurs. Dans cette direction, notre preuve automatisée présente plus de confiance dans la correction de l'algorithme car chaque étape a été vérifiée par l'outil de preuve PVS. Même s'il est un démonstrateur interactif qui travaille sous le contrôle direct de l'utilisateur, il est capable de faire de grandes étapes de déduction de manière autonome par l'appel aux procédures de décision pour l'arithmétique, à la réécriture et à la récurrence explicite. L'avantage de l'utilisation de techniques de preuve par induction implicite par rapport aux celles employées par PVS, est qu'elles permettent un degré supérieur d'automatisation. Ainsi, nous arrivons à démontrer de manière complètement automatique la majorité des lemmes de cette preuve.

Plan de la thèse

Le mémoire s'organise en deux parties et comprend deux annexes. Les six chapitres constituant la première partie présentent les fondements théoriques et résultats liés aux preuves par récurrence avec ensembles couvrants contextuels. Voici une présentation rapide des points d'intérêt de chacun des chapitres :

- Dans le chapitre (1), nous survolons les notions de base concernant les systèmes de déduction, les ordres de récurrence, les spécifications conditionnelles, les systèmes de réécriture conditionnelle et les procédures de décision.

- Le chapitre (2) introduit le concept-clé d'ensemble couvrant contextuel et ses propriétés. Puis, nous développons différentes méthodes pour composer des ensembles couvrants contextuels.

- Le chapitre (3) présente dans un premier temps la description du système d'inférence abstrait défini en termes d'ensembles couvrants contextuels et fait une étude comparative

† en anglais, Available Bit Rate

avec plusieurs procédures abstraites similaires, basées sur des ensembles couvrants [Stratulat, 1999; Stratulat, 2000]. Nous montrons successivement ses propriétés de correction, de correction réfutationnelle et les conditions nécessaires pour garantir sa complétude réfutationnelle. Puis, nous introduisons le concept de module de raisonnement et nous montrons que le schéma d'intégration d'un ensemble de modules de raisonnement arbitraires dans le système abstrait d'inférence préserve ses propriétés de correction et de complétude.

– Dans le chapitre (4), nous décrivons le système d'inférence de SPIKE en termes de modules de raisonnement, comme une instance du système abstrait d'inférence intégrant des modules de raisonnement basés sur des techniques de réécriture spécifiques aux théories conditionnelles.

– De nouvelles extensions et améliorations du système d'inférence de SPIKE sont présentées dans le chapitre (5). Dans un premier temps, nous introduisons la technique de subsomption sémantique inductive [Stratulat, 1998a]. Puis, nous montrons comment le système d'inférence de SPIKE, étendu avec des règles d'inférence basées sur cette technique et sur des procédures de décision pour l'arithmétique linéaire, a été utilisé pour montrer la correction de l'algorithme MJRTY. Une autre optimisation proposée est la réduction du temps d'exécution des preuves de SPIKE par parallélisation, grâce à son interface parallèle SPIKEPAR [Stratulat, 1998b].

– Le dernier chapitre de cette partie est consacré à la proposition d'un schéma de coopération entre des procédures de décision pour l'arithmétique linéaire et pour la clôture par congruence. Il est à la base d'une nouvelle technique de raisonnement utilisée par SPIKE.

La dernière partie, divisée en deux chapitres, présente quelques applications de la méthode de preuve par récurrence avec des ensembles couvrants contextuels dans le domaine de la vérification de logiciels de télécommunications.

– Le chapitre (7) est consacré à l'étude des interactions des services téléphoniques en proposant une méthodologie pour les détecter et les résoudre par des techniques de réécriture [Klay *et al.*, 1999]. Il contient aussi une étude de cas concernant l'interopérabilité des services de renvoi inconditionnel et de filtrage des appels à l'arrivée, obtenue avec notre méthodologie.

– Dans le chapitre (8), nous montrons la conformité de l'algorithme incrémental qui calcule les débits admis par le protocole ABR dans les réseaux ATM [Rusinowitch *et al.*, 2000]. En premier lieu, nous présentons le squelette de la preuve, déduite à partir d'une preuve automatisée faite à l'aide du démonstrateur interactif PVS. Puis, nous présentons des lemmes simples qui, contrairement à PVS, sont prouvés complètement automatiquement par SPIKE, en utilisant une combinaison de raisonnements inductif et arithmétique.

L'annexe A contient la spécification SPIKE obtenue avec la méthodologie présentée dans le chapitre (7).

Les spécifications PVS et SPIKE des algorithmes ABR présentés dans le chapitre (8) sont données dans l'annexe B.

Première partie

Preuves par récurrence avec ensembles couvrants contextuels

1

Notions fondamentales

Sommaire

Dans ce chapitre, nous allons introduire des notions de base et des notations qui vont nous servir tout au long de cette thèse.

Le lecteur intéressé peut trouver plus de détails dans [Ehrig et Mahr, 1985; Padawitz, 1988; Wirsing, 1990; Fitting, 1990].

1.1 Formules, termes et substitutions

Dans la suite, on va supposer que $\mathbb{V}$ est un *vocabulaire* dénombrable contenant

1. V, un ensemble dénombrable de *variables*,
2. F, un ensemble fini de *symboles de fonctions* munis d'un profil, et
3. P, un ensemble fini de *symboles de prédicats* munis d'un profil.

Le *langage* $\mathcal{L}$ est un ensemble récursif sur le vocabulaire $\mathbb{V}$ dont les éléments sont des *formules*. On va supposer qu'une étiquette unique, appelée *symbole de sorte*, est associée à chaque élément du vocabulaire. S représente l'ensemble (supposé fini) des symboles de sorte qui contient au moins le symbole `bool` qu'on associe à tous les symboles de prédicats.

Les symboles de fonctions sont caractérisés par des *signatures*.

Définition 1.1.1 (signature, constante) *Une* signature $\Sigma = (\mathsf{S}, \mathsf{F})$ *est représentée par l'ensemble de symboles de sortes* S *et de fonctions* F*. Tout symbole de fonction* $f \in \mathsf{F}$ *est de profil*

$$f : S_1 \times \cdots \times S_n \to S_{n+1}$$

où n *est l'*arité *de* f *et* $S_i \in \mathsf{S}$*, pour tout* $i \in [1..n+1]$*.*

Un symbole de fonction d'arité 0 *est une* constante*.*

Etant donnés une signature (S, F) et un ensemble de variables V, l'ensemble $\mathcal{T}(\mathsf{F}, \mathsf{V})$ est l'ensemble de *termes* construit à partir de F et V. Un symbole de sorte unique est associé à chaque terme. $\mathcal{T}(\mathsf{F}, \mathsf{V})$ peut être partitionné en sous-ensembles disjoints suivant les éléments de S.

$$\mathcal{T}(\mathsf{F}, \mathsf{V}) = \biguplus_{S \in \mathsf{S}} \mathcal{T}(\mathsf{F}, \mathsf{V})_S$$

Définition 1.1.2 (terme) *Soit* (S, F) *une signature et* V *un ensemble de variables. L'ensemble* $\mathcal{T}(\mathsf{F}, \mathsf{V})$ *est le plus petit ensemble tel que :*

- *toute variable* v *à laquelle est associé le symbole de sorte* $S \in \mathsf{S}$ *est un terme appartenant à* $\mathcal{T}(\mathsf{F}, \mathsf{V})_S$*.*
- *pour tout symbole de fonction* f *de* F *de profil* $f : S_1 \times \ldots \times S_n \to S$ *et pour tout* n*-uplet de termes* $(t_1, \ldots, t_n) \in \mathcal{T}(\mathsf{F}, \mathsf{V})_{S_1} \times \cdots \times \mathcal{T}(\mathsf{F}, \mathsf{V})_{S_n}$*,* $f(t_1, \ldots, t_n)$ *est un terme de* $\mathcal{T}(\mathsf{F}, \mathsf{V})_S$*.*

L'ensemble $\mathcal{T}(\mathsf{F}, \mathsf{V})_S$ est aussi nommé le *sorte* S. Chaque sorte est ensuite supposée admettre au moins un élément. Par exemple, la sorte `bool` est constituée par les constantes logiques $\{\mathit{True}, \mathit{False}\}$. Une sorte S est *finitaire* si le nombre de termes de sorte S est fini, sinon elle est *infinitaire*. Par $\equiv$, on note la relation d'égalité syntaxique entre deux termes.

Un terme a la structure d'un arbre. Les chemins d'accès dans l'arborescence d'un terme sont indiqués par des *positions*.

Définition 1.1.3 (positions) *Etant donnés un terme* t *et l'opération de concaténation de deux chaînes ., l'ensemble de* positions *de* t*, noté* $pos(t)$*, est constitué par des chaînes finies de naturels strictement positifs définies récursivement comme suit :*

- *la chaîne vide* $\epsilon \in pos(t)$*, désignant la position de la racine de l'arbre ;*
- *si* $p \in pos(t_i)$*, alors* $i.p \in pos(f(\ldots, t_i, \ldots))$*, où* f *est d'arité* n *et* $i \in [1..n]$*.*

Si t est un terme et p une position, on note $t(p)$ le symbole de t à la position p. Les positions correspondant à des chemins qui mènent à des variables, s'appellent des *positions de variables*. Elles constituent l'ensemble $vpos(t) = \{p \in pos(t) \mid t(p) \in \mathsf{V}\}$. On note par $spos(t)$ l'ensemble $pos(t) \backslash vpos(t)$ des *positions strictes*.

Une des caractéristiques d'un terme est sa *profondeur*. Sachant que la *profondeur d'une position* p dans un terme t est la longueur de la chaîne p, dénotée par $|p|$, alors la profondeur maximale de toutes les positions d'un terme t définit la profondeur du terme t. De la même façon, on définit la *profondeur stricte d'un terme*, qui est la profondeur maximale de toutes les positions strictes du terme.

Les *sous-termes* d'un terme sont désignés par des positions. Soient t un terme et p une de ses positions. La notation $t[t']_p$ indique que le terme t' est le sous-terme de t à la position p, noté lui-même par t/p tel que $t'(p') = t(p.p')$ pour tout $p' \in pos(t')$. Un *sous-terme strict* de t

est un sous-terme de t à la position p telle que $p \neq \epsilon$. Afin de spécifier simplement que s est un sous-terme de t, on écrit $t[s]$.

Exemple 1.1.1 *Soit la signature* ($\{\texttt{nat}, \texttt{bool}\}$, {True, False, 0, s, min, $\leq$}) *avec les symboles de fonction de profil*

True	:		$\rightarrow$	$\texttt{bool}$
False	:		$\rightarrow$	$\texttt{bool}$
0	:		$\rightarrow$	$\texttt{nat}$
s	:	$\texttt{nat}$	$\rightarrow$	$\texttt{nat}$
min	:	$\texttt{nat} \times \texttt{nat}$	$\rightarrow$	$\texttt{nat}$
$\leq$	:	$\texttt{nat} \times \texttt{nat}$	$\rightarrow$	$\texttt{bool}$

Les termes $s(0)$, $min(s(x), 0)$ *et* 0 *sont des termes de sorte* $\texttt{nat}$, $s(s(y)) \leq z$ *est un terme de sorte* $\texttt{bool}$, *si* x, y *et* z *sont des variables de sorte* $\texttt{nat}$.

Etant donné le terme $min(s(x), 0)$, *le symbole* min *se trouve à la position* ϵ, *le symbole* s, *à la position* 1, *le symbole* x *à la position* 1.1, *le symbole* 0 *à la position* 2. *De plus,* $pos(min(s(x), 0)) = \{\epsilon, 1, 2, 1.1\}$, $spos(min(s(x), 0)) = \{\epsilon, 1, 2\}$ *et* $vpos(min(s(x), 0)) = \{1.1\}$. *Le sous-terme de* $min(s(x), 0)$ *à la position* 1 *est* $s(x)$.

L'ensemble de variables d'un terme t est noté $Var(t)$. S'il est vide, on dit que t est un terme *clos*. On notera par $\mathcal{T}(\mathsf{F})$ le plus petit ensemble de termes clos.

Les *substitutions* permettent de remplacer les variables d'un terme par d'autres termes.

Définition 1.1.4 (substitution, substitution close) *Une* substitution *est un ensemble fini d'applications de* V *dans* $\mathcal{T}(\mathsf{F}, \mathsf{V})$. *L'application d'une substitution* $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ *à un terme, telle que pour tout* $i \in [1..n]$ *les termes* x_i *et* t_i *ont la même sorte, est définie récursivement :*

- *s'il existe* $i \in [1..n]$, *tel que* t *est la variable* x_i *alors* $\sigma(t) = t_i$;
- *si* t *est une variable* $x \neq x_i$, *pour tout* $i \in [1..n]$, *alors* $\sigma(t) = t$;
- *si* t *est un terme* $f(s_1, \ldots, s_n)$ *avec* $s_1, \ldots, s_n \in \mathcal{T}(\mathsf{F}, \mathsf{V})$ *et* $f \in \mathsf{F}$, *alors* $\sigma(t) = f(\sigma(s_1), \ldots, \sigma(s_n))$.

Pour tout $i \in [1..n]$, *si* $t_i \in \mathcal{T}(\mathsf{F})$, *alors on dit que* σ *est une* substitution close.

L'application d'une substitution σ à un terme t est notée $t\sigma$. Comme toute fonction, les substitutions peuvent être composées ; si θ et τ sont deux substitutions, on appelle $\theta\tau$ la *composition des substitutions* θ et τ telle que, pour chaque variable x du domaine de θ, la relation $x(\theta\tau) = (x\theta)\tau$ est satisfaite.

Définition 1.1.5 (instance, filtre, unificateur) *Etant donnés deux termes* s *et* t, *on dit que* t *est une* instance *de* s *s'il existe une substitution* σ *telle que* $s\sigma \equiv t$. *Dans ce cas,* σ *est un* filtre *de* s *vers* t *et on dit que* s filtre t. *On dit qu'une substitution* σ *est un* unificateur *de deux termes* s *et* t, *ou que* s *s'*unifie *avec* t, *si* $s\sigma \equiv t\sigma$.

Exemple 1.1.2 *Reprenons l'exemple 1.1.1, on considère le terme* $t = min(x, y)$ *et les deux substitutions* $\sigma_1 = \{x \mapsto 0, y \mapsto z\}$ *et* $\sigma_2 = \{z \mapsto s(0)\}$. *Alors l'instance* $t\sigma_1$ *de* t *correspond au terme* $min(0, z)$, *tandis que* $t\sigma_1\sigma_2$ *correspond à* $(t\sigma_1)\sigma_2 = min(0, s(0))$. *Le terme* t *filtre* $\min(s(0), s(0))$ *avec la substitution* $\{x \mapsto s(0), y \mapsto s(0)\}$.

Les formules les plus élémentaires sont des *atomes*, appelés aussi des *formules atomiques*.

Définition 1.1.6 (formule atomique) *Une* formule atomique *de $\mathcal{L}$ est une expression de la forme*

$$p(t_1, \ldots, t_n),$$

où $p \in \mathsf{P}$ est un symbole de prédicat d'arité n et de profil $S_1 \times \cdots \times S_n \rightarrow \texttt{bool}$, tel que pour tout $i \in [1..n]$, t_i est un terme de sorte S_i.

L'application d'une substitution σ à une formule atomique $p(t_1, \ldots, t_n)$ est la formule atomique $p(t_1\sigma, \ldots, t_n\sigma)$. L'ensemble de formules atomiques est noté $\mathcal{A}(\mathsf{P}, \mathsf{F}, \mathsf{V})$.

Une formule plus compliquée est la *clause*, qui peut être constituée d'une ou plusieurs formules atomiques. Avant de la définir, on va préciser qu'un *multiensemble* est un ensemble dont les éléments peuvent apparaître plusieurs fois. Les opérations $\setminus\!\setminus$, $\uplus$ et $\Cap$ vont dénoter respectivement la différence, l'union et l'intersection définies sur les multiensembles.

Définition 1.1.7 (clause) *Une clause $a_1 \wedge \ldots \wedge a_n \Rightarrow a'_1 \vee \ldots \vee a'_m$ (notée aussi $\neg a_1 \vee \ldots \vee \neg a_n \vee a'_1 \vee \ldots \vee a'_m$, ou bien $P \Rightarrow C$, où P est la conjonction $a_1 \wedge \ldots \wedge a_n$ et C est la disjonction $a'_1 \vee \ldots \vee a'_m$) est une formule représentée par la paire de multiensembles $(\{a_1, \ldots, a_n\}, \{a'_1, \ldots, a'_m\})$, où $a_1, \ldots, a_n$ et $a'_1, \ldots, a'_m$ sont des formules atomiques et n, $m \geq 0$. Si $n = m = 0$, alors on dit que la clause est* vide *et qu'elle représente la valeur logique* False*. Les formules atomiques $a_1, \ldots, a_n$ (resp. $a'_1, \ldots, a'_m$) s'appellent les* antécédents *(resp.* conséquents*) de la clause.*

La notion de position dans un terme peut s'étendre aux clauses par la spécification du multiensemble dans la paire, de la formule atomique dans le multiensemble,* du terme dans la formule atomique et de la position dans le terme.

On peut exprimer des formules plus complexes, représentant des ensembles de clauses, par exemple des formules dans la logique des prédicats du premier ordre sans quantificateur. Dans un premier temps, on va définir la syntaxe du langage $\mathcal{L}_{PO}$ contenant ces formules.

Le *vocabulaire* $\mathbb{V}_{PO}$ est constitué par

- V_{PO}, ensemble dénombrable de variables,
- F_{PO}, un ensemble fini de symboles de fonctions munis d'un profil,
- P_{PO}, un ensemble fini de symboles de prédicats munis d'un profil,
- les connecteurs $\neg \ \Rightarrow \ \vee \ \wedge \ \Leftrightarrow$, et
- des signes de ponctuation [ ] (,).

Ces ensembles sont supposés disjoints.

Définition 1.1.8 (formule de $\mathcal{L}_{PO}$) *$\mathcal{L}_{PO}$ est le plus petit ensemble de formules tel que*

- *toute formule atomique construite de la même façon qu'une formule atomique de $\mathcal{L}$ dans la définition 1.1.6 en utilisant V_{PO}, F_{PO} et P_{PO} est une formule de $\mathcal{L}_{PO}$,*
- *si ϕ est une formule de $\mathcal{L}_{PO}$, alors $\neg\phi$ l'est aussi, et*
- *si ϕ et ψ sont deux formules de $\mathcal{L}_{PO}$, alors $\phi \Rightarrow \psi$, $\phi \vee \psi$, $\phi \wedge \psi$ et $\phi \Leftrightarrow \psi$ le sont aussi.*

∗ On suppose que les multiensembles sont mis en œuvre par des listes.

Toute formule de $\mathcal{L}_{PO}$ peut être normalisée comme une conjonction finie de clauses représentant une de ses formes normales conjonctives (ou formes clausales). Par $[\phi]^{\mathtt{cnf}}$, on va noter une forme clausale de ϕ.

Exemple 1.1.3 *Soit deux clauses $c_1 = (a_1 \vee a_2)$ et c_2. Alors une forme clausale de la formule $\phi = (c_1 \Rightarrow c_2)$ est $[\phi]^{\mathtt{cnf}} = (\neg a_1 \vee c_2) \wedge (\neg a_2 \vee c_2)$.*

Par $Var(\phi)$, on spécifie les variables d'une formule ou d'un terme ϕ. Une variable est *linéaire* dans ϕ si elle n'apparaît qu'une seule fois dans ϕ. Si toutes les variables de ϕ sont linéaires, alors ϕ est linéaire. Généralisant pour des formules la notion de substitution appliquée aux termes, l'application d'une substitution σ sur une formule ϕ, notée $\phi\sigma$, est la formule qu'on obtient après le remplacement dans ϕ de chaque variable x (du domaine de σ et appartenant à $Var(\phi)$) par le terme de son image. Dans ce cas, on dit que $\phi\sigma$ est une *instance* de ϕ. Si la relation $\phi\sigma = \phi$ est satisfaite, on dit que σ est une *substitution identité par rapport à ϕ* et on va la noter par $\{\}$.

On va étendre la notion d'instance d'une formule pour des ensembles de formules. Si Φ est un ensemble de formules et σ une substitution, on va noter par $\Phi\sigma$ l'ensemble d'instances $\{\phi\sigma \mid \phi \in \Phi\}$.

1.2 Interprétations et modèles

Les formules du langage $\mathcal{L}$ peuvent être interprétées dans un autre langage A défini sur un vocabulaire contenant l'ensemble des variables V_A, des symboles de fonctions F_A, de prédicats P_A et de sortes S_A à l'aide i) d'une *fonction d'interprétation* $\mathbb{I}$ de chaque élément de S dans S_A, de F dans F_A et de P dans P_A, ii) d'un *domaine d'interprétation* $D^A_{\mathbb{I}}$, contenu par $\mathcal{T}(\mathsf{F}_A, \mathsf{V}_A)$, et iii) d'une *fonction d'assignation* ν des variables de V dans $D^A_{\mathbb{I}}$.

Définition 1.2.1 (interprétation, domaine d'interprétation) *Une fonction d'interprétation $\mathbb{I}$, nommée plus simplement une* interprétation*, est caractérisée par la famille d'applications $(\mathbb{I}_\mathsf{S}, \mathbb{I}_\mathsf{F}, \mathbb{I}_\mathsf{P})$ $(\mathbb{I}_\mathsf{S} : \mathsf{S} \to \mathsf{S}_A, \mathbb{I}_\mathsf{F} : \mathsf{F} \to \mathsf{F}_A, \mathbb{I}_\mathsf{P} : \mathsf{P} \to \mathsf{P}_A)$ telle que*

- *$\mathbb{I}_\mathsf{S}(\mathtt{bool}) = \mathtt{bool}$, et*
- *pour tout symbole de fonction $f \in \mathsf{F}$ de profil $f : S_1 \times \cdots \times S_n \to S_{n+1}$, il existe un symbole de fonction $\mathbb{I}_\mathsf{F}(f)$ déclaré*

$$\mathbb{I}_\mathsf{F}(f) : \mathbb{I}_\mathsf{S}(S_1) \times \cdots \times \mathbb{I}_\mathsf{S}(S_n) \to \mathbb{I}_\mathsf{S}(S_{n+1}), \text{ et}$$

- *pour tout symbole de prédicats $p \in \mathsf{P}$ de profil $p : S_1 \times \cdots \times S_n \to \mathtt{bool}$, il existe un symbole de prédicats $\mathbb{I}_\mathsf{P}(p)$ déclaré*

$$\mathbb{I}_\mathsf{P}(p) : \mathbb{I}_\mathsf{S}(S_1) \times \cdots \times \mathbb{I}_\mathsf{S}(S_n) \to \mathtt{bool}$$

Le domaine d'interprétation *$D^A_{\mathbb{I}}$ est contenu par l'union disjointe $\biguplus_{S \in \mathsf{S}_A} \mathcal{T}(\mathsf{F}_A, \mathsf{V}_A)_S$.*

Deux interprétations peuvent être comparées par des applications récursives sur la structure des termes de leur domaine d'interprétation, appelées *(homo)morphismes*.

Définition 1.2.2 (morphisme, isomorphisme) *Etant données deux interprétations $\mathbb{I}^1 = (\mathbb{I}^1_\mathsf{S}, \mathbb{I}^1_\mathsf{F}, \mathbb{I}^1_\mathsf{P})$ et $\mathbb{I}^2 = (\mathbb{I}^2_\mathsf{S}, \mathbb{I}^2_\mathsf{F}, \mathbb{I}^2_\mathsf{P})$ définies respectivement pour les langages A^1 et A^2, on appelle*

morphisme *une application* ϕ *de* $\mathcal{T}(\mathsf{F}_{A^1}, \mathsf{V}_{A^1})$ *vers* $\mathcal{T}(\mathsf{F}_{A^2}, \mathsf{V}_{A^2})$ *étendue aux formules atomiques, telle que pour tout n-uplet* $(a_1, \ldots, a_n) \in (\mathcal{T}(\mathsf{F}_{A^1}, \mathsf{V}_{A^1})_{\mathbb{I}^1_{\mathsf{S}}(S_1)} \times \cdots \times \mathcal{T}(\mathsf{F}_{A^1}, \mathsf{V}_{A^1})_{\mathbb{I}^1_{\mathsf{S}}(S_n)})$*, et*

– *pour chaque symbole de fonction* $f \in \mathsf{F}$ *de profil* $f : S_1 \times \cdots \times S_n \to S_{n+1}$*, on a*

$$\phi(\mathbb{I}^1_{\mathsf{F}}(f)(a_1, \ldots, a_n)) = \mathbb{I}^2_{\mathsf{F}}(f)(\phi(a_1), \ldots, \phi(a_n)), \text{ et}$$

– *pour chaque symbole de prédicats* $p \in \mathsf{P}$ *de profil* $p : S_1 \times \cdots \times S_n \to \texttt{bool}$*, on a*

$$\phi(\mathbb{I}^1_{\mathsf{P}}(p)(a_1, \ldots, a_n)) = \mathbb{I}^2_{\mathsf{P}}(p)(\phi(a_1), \ldots, \phi(a_n))$$

Un morphisme bijectif est un isomorphisme*.*

Définition 1.2.3 (assignation) *Soit* A *un langage. Une* assignation $\nu : \mathsf{V} \to D^A_{\mathbb{I}}$ *de l'interprétation* $\mathbb{I}$ *est une famille d'applications disjointes*

$$\biguplus_{j \in [1..n]} \nu_j,$$

qu'on étend à un morphisme de $\mathcal{T}(\mathsf{F}, \mathsf{V})$ *vers* $\mathcal{A}(\mathsf{P}, \mathsf{F}, \mathsf{V})$ *de la façon suivante :*

$$\nu(f(t_1, \ldots, t_n)) = (\mathbb{I}_{\mathsf{F}}(f))(\nu(t_1), \ldots, \nu(t_n))$$
$$\nu(p(t_1, \ldots, t_n)) = (\mathbb{I}_{\mathsf{P}}(p))(\nu(t_1), \ldots, \nu(t_n))$$

où $\nu_j : \mathsf{V}_{S_j} \to \mathcal{T}(\mathsf{F}_A, \mathsf{V}_A)_{\mathbb{I}_{\mathsf{S}}(S_j)}$ *(*$1 \leq j \leq n$*).*

La définition 1.2.3 suffit à interpréter des formules atomiques, mais pas toutes les formules de $\mathcal{L}_{PO}$. Dans la suite, on suppose simplement que les formules générales du $\mathcal{L}$ sont interprétables afin d'introduire quelques notions liées à la théorie des modèles. Puis, on va donner comme exemple le cas où les formules sont des clauses et des formules de $\mathcal{L}_{PO}$.

Définition 1.2.4 (satisfaisabilité, modèle) *Un ensemble de formules* Φ *de* $\mathcal{L}$ *est* satisfaisable *(ou cohérent) s'il existe une interprétation* $\mathbb{I}$ *telle que, pour toute formule* $\phi \in \Phi$ *et toute assignation* ν *de* $\mathbb{I}$*, on a* $\nu(\phi) = \text{True}$*. Dans ce cas, on dit aussi que* $\mathbb{I}$ *est un* modèle *de* Φ*.*

Le domaine d'interprétation caractérise le type de modèle d'un ensemble de formules. Par exemple, si on considère l'ensemble $\mathcal{T}(\mathsf{F}_A, \mathsf{V}_A)$ (resp. $\mathcal{T}(\mathsf{F}_A)$), on dit que les modèles sont *déductifs* (resp. *inductifs*). Dans les notations suivantes, on va utiliser le paramètre λ qui va s'instancier avec des étiquettes nous précisant le domaine d'interprétation choisi. Ainsi, $\lambda = ded$ (resp. ind) si le type du modèle est déductif (resp. inductif). Dans la suite, on suppose que le domaine d'interprétation $D^A_{\mathbb{I}}$ est fixé et que λ est l'étiquette nous indiquant le type de modèle associé à lui.

Notation 1.2.1 ($\models_\lambda$) *Soit* Φ *un ensemble de formules et* ϕ *une formule. On écrit* $\Phi \models_\lambda \phi$ *si* ϕ *est satisfaisable dans tous les modèles de* Φ *de type indiqué par* λ*.*

Définition 1.2.5 (théorie, validité d'une formule, relation de conséquence) *La* théorie *d'un ensemble de formules* Φ *est formée par toutes les formules* ϕ *telles que* $\Phi \models_\lambda \phi$*. Si* $\Phi \models_\lambda \phi$ *on dit que* ϕ *est* valide *dans la théorie de* Φ*, ou bien que* ϕ *est une* conséquence *de* Φ*.*

La classe de modèles qui nous intéresse dans cette thèse est celle dont le domaine d'interprétation est (un sous-ensemble de) l'ensemble de termes clos $\mathcal{T}(\mathsf{F})$. Le domaine d'interprétation $\mathcal{T}(\mathsf{F})$ est aussi appelé *domaine de Herbrand*. Les interprétations (modèles) dans ce domaine s'appellent des *interprétations (modèles) de Herbrand*. Il faut satisfaire une contrainte pour les fonctions de F : tout appel de fonction qui prend comme argument des termes clos, doit retourner aussi un terme clos.

Définition 1.2.6 (interprétation de Herbrand) *Une* interprétation de Herbrand *est une interprétation dont le domaine est $\mathcal{T}(\mathsf{F})$ telle que pour toute fonction $f \in \mathsf{F}$ de profil $f : S_1 \times \cdots \times S_n \to S_{n+1}$ et pour tout n-uplet de termes*

$$(t_1, \ldots, t_n) \in (\mathcal{T}(\mathsf{F})_{S_1} \times \cdots \times \mathcal{T}(\mathsf{F})_{S_n})$$

on a $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathsf{F})_{S_{n+1}}$. Pour les symboles de prédicats, il n'y a aucune condition supplémentaire.

Certains modèles sont intéressants à étudier, en particulier le modèle *initial* qui évite tout risque d'ambiguïté au moment de l'interprétation. Il est défini de la manière suivante :

Définition 1.2.7 (modèle initiale) *Un modèle $\mathbb{M}^I$ est* initial *si, pour tout modèle $\mathbb{M}$, il existe un unique morphisme $\phi : \mathbb{M}^I \to \mathbb{M}$.*

La notion de relation de conséquence d'une clause se définit à partir de celle d'une formule atomique. Si C est la clause $a_1 \wedge \ldots \wedge a_n \Rightarrow a'_1 \vee \ldots \vee a'_m$, alors $\Phi \models_\lambda C$ ssi pour tout modèle $\mathbb{M}$ de Φ de type indiqué par λ,

si $\forall\, i \in [1..n]$ $\mathbb{M}$ est un modèle de a_i de type λ, alors $\exists\, j \in [1..m]$ tel que $\mathbb{M}$ l'est aussi pour a'_j.

Toute formule ϕ de $\mathcal{L}_{PO}$ peut être vue comme une formule de $\mathcal{L}$ sous l'hypothèse que $\mathsf{V}_{PO} \subseteq \mathsf{V}$, $\mathsf{F}_{PO} \subseteq \mathsf{F}$, $\mathsf{P}_{PO} \subseteq \mathsf{P}$. Alors, si $[\phi]^{\mathtt{cnf}} = C_1 \wedge \ldots \wedge C_n$, on va représenter ϕ comme l'ensemble $\{C_1, \ldots, C_n\}$. De plus, $\Phi \models_\lambda \phi$ ssi on a $\Phi \models_\lambda C_i$, pour tout $i \in [1..n]$. A partir de maintenant, on va considérer cette hypothèse vraie.

Les opérations de conjonction (resp. disjonction) des éléments d'un ensemble de formules seront notées $\bigwedge \Phi$ (resp. $\bigvee \Phi$). Si Ψ est l'ensemble vide, alors $\bigwedge \emptyset$ ($\bigvee \emptyset$) désignent la valeur logique *True* (*False*).

1.3 Systèmes de déduction

Dans cette section, nous allons présenter un cadre théorique pour vérifier mécaniquement différentes relations de conséquence entre deux ensembles de formules, appelés des *axiomes* et des *conjectures*. Le concept-clé ici est celui de *système de déduction*. Nous allons définir ce concept du point de vue des preuves par récurrence implicite, où les objets manipulés sont des ensembles de conjectures et de *prémisses* dont certaines instances jouent le rôle d'hypothèses de récurrence.

Définition 1.3.1 (système de déduction) *Un* système de déduction *est un quadruple* $(\mathbb{V}, \mathcal{L}, Ax, \mathsf{J})$*, où*

- $\mathbb{V}$ *est un ensemble dénombrable contenant des variables, des symboles de fonctions et de prédicats,*

- $\mathcal{L}$ *est le langage défini sur le vocabulaire* $\mathbb{V}$,
- *Ax est un ensemble d'*axiomes *qui est un sous-ensemble décidable*[†] *de* $\mathcal{L}$,
- J *est un* système d'inférence *contenant un ensemble fini de* règles d'inférence *représentant des relations binaires entre des paires d'ensembles de formules.*

Une règle d'inférence appartenant à un système d'inférence J est représentée sous la forme :

$$\text{NOM}: \ (E,\, H) \vdash^{\mathsf{J}} (E',\, H') \ \ [\text{si } \textit{Conditions}]$$

où NOM est le nom de la règle, E, E' (resp. H, H') sont deux ensembles de conjectures (resp. prémisses). L'ensemble *Conditions* est optionnel et contient des conditions concernant des propriétés sur E, E', H ou H'. Elles doivent être satisfaites pour que la règle NOM soit appliquée. L'application de la règle correspond à la transition de la paire $(E,\, H)$ à la paire $(E',\, H')$.

Une J-*dérivation linéaire* $(E^0,\, H^0) \vdash^{\mathsf{J}} \ldots \vdash^{\mathsf{J}} (E^n,\, H^n) \ldots$ d'un ensemble de conjectures E^0, partant d'un ensemble de prémisses H^0, est une suite de transitions commençant avec la paire $(E^0,\, H^0)$. L'*état de la dérivation* après i transitions est la paire $(E^i,\, H^i)$ et l'*étape* i *de la dérivation* correspond à la transition $(E^i,\, H^i) \vdash^{\mathcal{J}} (E^{i+1},\, H^{i+1})$. Une dérivation linéaire est *finie* si son nombre d'étapes est fini.

Définition 1.3.2 (dérivation finie avec succès, théorème, preuve) *Une dérivation linéaire d'un ensemble de conjectures* E^0 *finit avec* succès *si elle termine dans un état ayant un ensemble vide de conjectures, de la forme* $(E^0,\, H^0) \vdash^{\mathsf{J}} \ldots \vdash^{\mathsf{J}} (\emptyset,\, H^n)$. *Si la dérivation commence avec un ensemble vide de prémisses* H^0, *les éléments de* E^0 *sont des* théorèmes. *Dans ce cas, on écrit* $Ax \vdash_{\mathsf{J}} E^0$. *Toute dérivation finie d'un ensemble de théorèmes est une* preuve.

Exemple 1.3.1 (système de déduction équationnelle) *Dans un système de déduction équationnelle, il n'y a pas de symbole de prédicats et toute formule est une paire* $(s,\, t)$, *avec* s *et* t *deux termes de même sorte, écrite normalement comme une équation* $s = t$. *La* théorie équationnelle *d'un ensemble d'axiomes équationnelles Ax est formée par toutes les équations déduites de Ax en utilisant les* axiomes de l'égalité *:*

1. *(**réflexivité**)* $Ax \models_{ded} \phi$ *si* ϕ *est une* tautologie, *c'est-à-dire une équation de la forme* $t = t$,
2. *(**symétrie**) si* $s = t$ *est une équation et* $Ax \models_{ded} s = t$ *alors* $Ax \models_{ded} t = s$,
3. *(**transitivité**) si* $Ax \models_{ded} s = t$ *et* $Ax \models_{ded} t = u$, *alors* $Ax \models_{ded} s = u$, *pour tous les termes* s, t *et* u,
4. *(**instanciation**) si* $s = t$ *est une équation telle que* $Ax \models_{ded} s = t$ *et* σ *une substitution, alors* $Ax \models_{ded} s\sigma = t\sigma$
5. *(**congruence**) si* $Ax \models_{ded} \cup_i^n \{s_i = t_i\}$ *alors* $Ax \models_{ded} f(s_1, \ldots,\, s_n) = f(t_1, \ldots,\, t_n)$, *pour tous les termes* t_i, $s_i \in \mathcal{T}(\mathsf{F},\, \mathsf{V})_{S_i}$ $(i \in [1..n])$ *et toute fonction* $f \in \mathsf{F}$ *de profil* $f : S_1 \times \cdots \times S_n \to S_{n+1}$.

Un système d'inférence pour le système de déduction équationnelle, dénoté par E, *est présenté dans la figure 1.1. Les deux premières règles d'inférence éliminent respectivement des instances des axiomes et des identités de l'ensemble courant de conjectures. Les autres règles qui suivent,* Symmetry, Expansion, Generalization *et* Decomposition *correspondent aux axiomes* ***symétrie, transitivité, instanciation*** *et* ***congruence****, lues dans le sens inverse. Par exemple,*

† On dit qu'un ensemble X est décidable si on peut répondre par OUI ou NON à la question « Est-ce que x appartient à X ? », pour tout élément x.

DELETE AXIOM :	$(E \cup \{s\sigma = t\sigma\},\ H)$	$\vdash^{\mathsf{E}}$	$(E,\ H)$
	si σ est une substitution et $s = t \in Ax$		
DELETE IDENTITY :	$(E \cup \{t = t\},\ H)$	$\vdash^{\mathsf{E}}$	$(E,\ H)$
SYMMETRY :	$(E \cup \{s = t\},\ H)$	$\vdash^{\mathsf{E}}$	$(E \cup \{t = s\},\ H)$
EXPANSION :	$(E \cup \{s = u\},\ H)$	$\vdash^{\mathsf{E}}$	$(E \cup \{s = t,\ t = u\}, H)$
GENERALIZATION :	$(E \cup \{s\sigma = t\sigma\},\ H)$	$\vdash^{\mathsf{E}}$	$(E \cup \{s = t\},\ H)$
	si σ est une substitution		
DECOMPOSITION :	$(E \cup \{f(s^1, \ldots, s^n) = f(t^1, \ldots, t^n),\ H)$	$\vdash^{\mathsf{E}}$	$(E \cup \cup_{i=1}^{n}\{s^i = t^i\},\ H)$
	si $f \in \mathsf{F}$ a n arguments et		
	$f(s^1, \ldots, s^n)$, $f(t^1, \ldots, t^n)$ sont deux termes		

FIG. 1.1 – *Le système d'inférence* E *pour les systèmes de déduction équationnelle*

en partant d'un ensemble d'axiomes $\{g(x,x) = x\}$ *et d'une conjecture initiale* $a = g(a,g(a,a))$, *la preuve*

$$\frac{(\{\underline{a = g(a,g(a,a))}\},\ \emptyset) \vdash^{\mathsf{E}} (\{\underline{g(a,g(a,a)) = a}\},\ \emptyset) \vdash^{\mathsf{E}} (\{g(a,g(a,a)) = g(a,a),\ \underline{g(a,a) = a}\},\ \emptyset) \vdash^{\mathsf{E}}}{(\{\underline{g(a,g(a,a)) = g(a,a)}\},\ \emptyset) \vdash^{\mathsf{E}} (\{\underline{a = a},\ g(a,a) = a\},\ \emptyset) \vdash^{\mathsf{E}} (\{\underline{g(a,a) = a}\},\ \emptyset) \vdash^{\mathsf{E}} (\emptyset,\ \emptyset)}$$

est obtenue par des applications successives des règles SYMMETRY, EXPANSION, DELETE AXIOM, DECOMPOSITION, DELETE IDENTITY *et* DELETE AXIOM *sur les équations soulignées.*

On peut montrer que $Ax \vdash_{\mathsf{E}} s = t$ *ssi* $s \leftrightarrow_{Ax}^{*} t$, *où* $\leftrightarrow_{Ax}$ *est une relation définie par* $t[a\theta]_p \leftrightarrow_{Ax} t[b\theta]_p$, *telle que* p *est une position dans* t, θ *une substitution et* $a = b \in Ax$, *c'est-à-dire qu'on peut obtenir* s *à partir de* t *par des successions de remplacements d'égaux par égaux. La relation* $\leftrightarrow_{Ax}$ *est une relation d'équivalence, c'est-à-dire réflexive, symétrique et transitive. En général, par* R^* *on désigne la clôture transitive et réflexive de la relation* R.

Voici quelques propriétés importantes d'un système de déduction :

Définition 1.3.3 (correction, complétude, correction réfutationnelle) *On dit qu'un système de déduction* J *est*

- correct *si tout théorème dérivé à partir d'un ensemble d'axiomes est leur conséquence. Formellement, si* $(E, \emptyset) \vdash^{\mathsf{J}} \ldots \vdash^{\mathsf{J}} (\emptyset, H)$ *alors* $Ax \models_\lambda E$.
- complet *s'il peut déduire toutes les conséquences des axiomes. Formellement, si* $Ax \models_\lambda E$ *alors il existe une* J*-preuve* $(E, \emptyset) \vdash^{\mathsf{J}} \ldots \vdash^{\mathsf{J}} (\emptyset, H)$.
- réfutationnellement correct *si la réfutation d'une conjecture dans une étape de déduction implique la réfutation des conjectures initiales. On suppose qu'il y a un prédicat d'échec,* `Fail`(E), *pour réfuter un ensemble de conjectures* E, *tel que si* `Fail`(E) *est vrai, alors* $Ax \not\models_\lambda E$. *Formellement, si* $(E, \emptyset) \vdash^{\mathsf{J}} \ldots \vdash^{\mathsf{J}} (E', H)$ *et* `Fail`(E') *est vrai, alors* $Ax \not\models_\lambda E$.

Vérifions les propriétés mentionnées dans la définition 1.3.3 par rapport au système E.

Proposition 1.3.1 *Le système de déduction* E

1. *est correct,*

2. *est complet et*
3. *n'est pas réfutationnellement correct.*

Preuve Les arguments de la preuve de correction et de complétude du système E sont similaires à ceux de [Birkhoff, 1935].

On va montrer que E n'est pas réfutationnellement correct lorsque $Ax = \emptyset$. On définit le prédicat $\texttt{Fail}(E)$ ssi $s = t$ est une équation de E qui n'est pas une tautologie. Maintenant, on considère la E-dérivation suivante $(\{t = t\}, \emptyset) \vdash^{\mathsf{E}} (\{s = t, t = s\}, \emptyset)$ telle que s n'est pas syntaxiquement égal à t, qui a été obtenue par l'application de la règle Expansion sur la tautologie $t = t$. On a $\texttt{Fail}(\{s = t, t = s\})$ vrai car $s = t$ n'est pas une tautologie. Pourtant, $Ax \models_{ded} t = t$ parce qu'il existe la preuve $(\{t = t\}, \emptyset) \vdash^{\mathsf{E}} (\emptyset, \emptyset)$ en appliquant la règle Delete Identity.

Fin de preuve

1.4 Ordres de récurrence

Toute preuve par récurrence demande l'existence d'un *ordre* sur les formules. La correction des schémas de récurrence dépend de certaines propriétés des ordres employés. Cette section introduit quelques ordres à utiliser dans les preuves par récurrence, puis détaille un exemple de construction d'ordre sur des formules particulières comme les clauses équationnelles, c'est-à-dire des clauses dont les formules atomiques sont des équations.

Définition 1.4.1 (pré-ordre, pré-ordre total, ordre) *On appelle* pré-ordre *sur un ensemble A, noté $\leq$, une relation binaire transitive et réflexive sur A. Le pré-ordre $\leq$ est* total *si on a soit $s \leq t$, soit $t \leq s$, pour tous les éléments s, t de A. Toute relation binaire transitive et irréflexive est un ordre.*

On rappelle qu'une *relation d'équivalence* est une relation symétrique, transitive et réflexive. Un pré-ordre peut être décomposé en deux : la partie équivalence et la partie stricte.

Définition 1.4.2 (la partie équivalence et la partie stricte d'un pré-ordre) *Soit A un ensemble. La* partie équivalence d'un pré-ordre $\leq$ *sur A est notée $\sim$ telle que pour tous $s, t \in A$ on a*

$$s \sim t \text{ ssi } s \leq t \text{ et } t \leq s$$

La partie stricte d'un pré-ordre *sur A, notée $<$, est un ordre tel que pour tous $s, t \in A$ on a*

$$s < t \text{ ssi } s \leq t \text{ et } t \not\leq s$$

Un ordre de récurrence *bien fondé* peut être utilisé comme argument de la correction des schémas de récurrence.

Définition 1.4.3 ((pré-)ordre bien fondé) *Un (pré-)ordre $<$ ($\leq$) sur un ensemble A est* bien fondé *s'il n'existe pas de suite infinie décroissante $s_1 > s_2 > \ldots$.*

On dit que deux ordres bien fondés, $<_1$ et $<_2$, définis sur un même ensemble A, sont *compatibles*, si l'ordre $<_1 \cup <_2$ est inclus dans un ordre bien fondé.

Comme les formules sont souvent instanciées pendant une preuve par récurrence, on aimerait que les relations d'ordre soient invariantes (ou stables) par substitution.

Définition 1.4.4 (stabilité et stabilité forte par substitution) *Un pré-ordre $\leq$ sur un ensemble A est* stable par substitution *si*

$$\textit{pour tous } x, y \in A \textit{ et toute substitution } \sigma, \textit{ si } x \leq y \textit{ alors } x\sigma \leq y\sigma$$

Un pré-ordre $\leq$ sur un ensemble A est fortement stable par substitution *si*

- *pour tous s, $t \in A$ et toute substitution σ, si $s < t$ alors $s\sigma < t\sigma$, et*
- *pour tous s, $t \in A$ et toute substitution σ, si $s \sim t$ alors $s\sigma \sim t\sigma$*

Si un pré-ordre $\leq$ est fortement stable par substitution, alors il est stable par substitution. Notons que l'inverse n'est pas toujours vrai.

Les ordres sur les formules se construisent normalement à partir des ordres définis sur des termes et des formules atomiques, qu'on appelle génériquement des *ordres sur les termes*. Ici, on distingue deux classes d'ordres sur les termes qui sont les plus utilisés par les démonstrateurs de théorèmes basés sur la réécriture (voir dans la section 1.5) :

- *ordres de réduction* auxquels appartiennent tous les ordres bien fondés et stables par substitution $<_t$ qui sont définis sur $\mathcal{T}(\mathsf{F}, \mathsf{V})$ et $\mathcal{A}(\mathsf{P}, \mathsf{F}, \mathsf{V})$ et qui vérifient la **propriété de monotonie** (ou stabilité par contexte) :

  $$\text{si } s <_t t \text{ alors } f(\ldots, s, \ldots) <_t f(\ldots, t, \ldots) \text{ et } p(\ldots, s, \ldots) <_t p(\ldots, t, \ldots),$$

  pour tout s, $t \in \mathcal{T}(\mathsf{F}, \mathsf{V})$, $f \in \mathsf{F}$ et $p \in \mathsf{P}$,
- *ordres décroissants* $<_t$ qui sont des ordres de réduction vérifiant en outre la **propriété de sous-terme strict** : pour tout terme $t \in \mathcal{T}(\mathsf{F}, \mathsf{V})$, on a $t <_t s$ (resp. $t <_t p$) si t est un sous-terme strict du terme s (resp. de la formule atomique $p \in \mathcal{A}(\mathsf{P}, \mathsf{F}, \mathsf{V})$).

La plupart des ordres sur les termes qu'on connaît actuellement se construisent récursivement à partir d'un pré-ordre sur les symboles de fonctions, par exemple l'ordre récursif sur des chemins (ou RPO) [Dershowitz, 1982], ou des variantes de celui-ci, comme PSO [Plaisted, 1978], RDO [Jouannaud *et al.*, 1982] ou MPO [Dershowitz et Jouannaud, 1990; Bronsard *et al.*, 1996].

Dans la suite, on va détailler l'ordre MPO. D'abord, on va introduire la notion de congruence sur des termes et celle d'extension multiensemble d'un pré-ordre [Dershowitz et Manna, 1979].

Définition 1.4.5 (relation de congruence sur des termes $\approx$) *Etant donnée une précédence $\leq_{\mathsf{F}}$ sur les symboles de fonctions, la relation $\approx$ est récursivement définie comme*

$$f(s_1, \ldots, s_n) \approx g(t_1, \ldots, t_n) \textit{ lorsque } f \sim_{\mathsf{F}} g \textit{ et } s_i \approx t_i, \textit{ pour tout } i \in [1..n].$$

Dans le cas de base, $s \approx t$ si $s \equiv t$, pour tous les termes s et t.

Définition 1.4.6 (extension multiensemble d'un pré-ordre) *Soient $\leq$ un pré-ordre sur un ensemble A, $\sim$ une relation d'équivalence entre les éléments de A, et A_1, A_2 deux multiensembles contenant des éléments de A. On définit $\setminus_\sim$ comme la relation de différence sur les multiensembles pour laquelle on remplace le test d'égalité par un test d'équivalence :*

- *$X \setminus_\sim Y = X$, s'il n'existe pas d'élément $a \in X$ tel que $a \sim b$, pour tout élément $b \in Y$, et*

– $(X \cup \{a\})\backslash_\sim(Y \cup \{b\}) = X\backslash_\sim Y$ *si* $a \sim b$.

A'_1 *dénote* $A_1\backslash_\sim A_2$ *et* A'_2 *le multiensemble* $A_2\backslash_\sim A_1$. *Alors on écrit*

1. $A_1 \ll A_2$ *si, pour tout* $a \in A'_1$, *il existe un élément* $b \in A'_2$ *tel que* $a < b$, *et*
2. $A_1 \approx A_2$ *si* $A'_1 = A'_2 = \emptyset$, *et*
3. $A_1 \lessdot A_2$ *si* $A_1 \ll A_2$ *ou* $A_1 \approx A_2$.

On dit que $\lessdot$ *est l'*extension multiensemble *de* $\leq$.

Pour tout pré-ordre $\leq$, son extension multiensemble $\lessdot$ est aussi un pré-ordre. De plus, si $\leq$ est stable (fortement stable) par substitution, $\lessdot$ l'est aussi.

Définition 1.4.7 (pré-ordre multiensemble sur les chemins MPO) *Soit* $\leq_\mathsf{F}$ *un pré-ordre sur les symboles de fonctions. Le pré-ordre multiensemble sur les chemins* $<_{MPO}$ *induit sur* $\mathcal{T}(\mathsf{F}, \mathsf{V})$, *est*

$$t = g(t_1, \ldots, t_n) <_{MPO} f(s_1, \ldots, s_m) = s$$

si au moins une des variantes suivantes est satisfaite :

1. $g \sim_\mathsf{F} f$ *et* $\{t_1, \ldots, t_n\} \ll_{MPO} \{s_1, \ldots, s_m\}$, *ou*
2. $g <_\mathsf{F} f$ *et* $t_i <_{MPO} s$, *pour tout* $i \in [1..n]$, *ou*
3. $t <_{MPO} s_i$, *pour un indice* $i \in [1..m]$, *ou*
4. $t \approx s_i$, *pour un indice* $i \in [1..m]$.

De la même façon, on peut définir $<_{MPO}$ *sur* $\mathcal{A}(\mathsf{P}, \mathsf{F}, \mathsf{V})$.

$<_{MPO}$ est un ordre décroissant [Bronsard *et al.*, 1996]. Il est aussi *incrémental* par rapport à la précédence sur les symboles des fonctions $\leq_\mathsf{F}$. Ainsi, l'extension de ordre $\leq_\mathsf{F}$ avec de nouveaux symboles induit un nouvel ordre MPO sur les termes qui est compatible avec $<_{MPO}$.

Le lemme suivant garantit l'existence des relations de comparaison entre les symboles de fonctions de deux termes comparable par $<_{MPO}$.

Lemme 1.4.1 *Soient* s *et* t *deux termes de* $\mathcal{T}(\mathsf{F}, \mathsf{V})$ *tels que* $t <_{MPO} s$. *Alors pour tout symbole de fonction* f_t *de* t *il existe un symbole de fonction* f_s *de* s *satisfaisant* $f_t \leq_\mathsf{F} f_s$.

Preuve On va raisonner par récurrence sur la structure des termes t et s, en utilisant la définition 1.4.7.

– t est une constante.
  – s est aussi une constante. La seule variante possible de la définition est 2, pour laquelle $t <_\mathsf{F} s$;
  – s est de la forme $s = f(s_1, \ldots, s_m)$. On suppose, par hypothèse de récurrence, que tous les termes s' de profondeur inférieure à s, satisfaisant $t <_{MPO} s'$, ont un symbole de fonction $f_{s'}$ tel que $t <_\mathsf{F} f_{s'}$. On peut appliquer soit la variante 2, soit 3, soit 4. Dans le premier cas, on a $t <_\mathsf{F} f$. Dans le deuxième cas, on applique l'hypothèse de récurrence car il existe $i \in [1..m]$ tel que $t <_{MPO} s_i$ et la profondeur de s_i est inférieure à s. Donc il existe un symbole $f'_{s'}$ de s_i, contenu aussi par s, tel que $t <_\mathsf{F} f'_{s'}$. Le dernier cas est trivial.

– t est de la forme $g(t_1, \ldots, t_n)$. On va raisonner par récurrence sur la structure de t. On va supposer par récurrence que pour tous les symboles de fonction $f_{t'}$ d'un terme arbitraire t' de profondeur inférieure à t, si $t' <_{MPO} s$, alors il existe un symbole de fonction f_s dans s satisfaisant $f_{t'} <_{\mathsf{F}} f_s$. On va analyser par cas chaque variante.
  – variante 1. s doit être de la forme $f(s_1, \ldots, s_m)$. Alors $g \sim_{\mathsf{F}} f$. En plus, pour chaque t_i, avec $i \in [1..n]$, il existe un $j \in [1..m]$ tel que $t_i <_{MPO} s_j$. Comme t_i a une profondeur inférieure à t, on peut appliquer l'hypothèse de récurrence : pour chaque symbole de fonction de t_i, il existe un symbole de fonction de s_j supérieur ou équivalent à lui.
  – variante 2. On a $g <_{\mathsf{F}} f$ et pour tout t_i, avec $i \in [1..n]$, $t_i <_{MPO} s$. Puis, on applique l'hypothèse de récurrence.
  – variante 4. Triviale.
  – variante 3. Il existe $i \in [1..m]$ tel que $t <_{MPO} s_i$. Alors on peut appliquer une des variantes 1, 2 ou 4, ou récursivement la variante 3 sur un sous-terme de s_i. Dans le cas de base de la récursivité, ce sous-terme est une constante. On applique alors la variante 2.

Fin de preuve

Les ordres sur les clauses équationnelles peuvent se construire à partir des ordres sur les termes. On rappelle qu'une clause se représente comme une paire de multiensembles de formules atomiques. Donc, il est naturel de considérer les ordres sur les clauses comme des extensions multiensemble d'ordres sur les termes. Par exemple, le pré-ordre suivant, défini sur des clauses équationnelles, est obtenu à partir d'ordres sur les équations [Reddy, 1990].

Définition 1.4.8 ((pré-)ordre sur des clauses équationnelles) *Soient $\leq_t$ un pré-ordre sur les termes et $\lessdot_t$ son extension multiensemble. La complexité d'une équation e de la forme $s = t$, dénotée par Comp(e), est définie comme le multiensemble $\{s, t\}$. On suppose que max(A) représente le multiensemble d'éléments maximaux (par rapport à $\leq$) d'un multiensemble A et que C est la clause $\vee_{i=1}^{m}(\neg e_i^n) \vee \vee_{j=1}^{k} e_j^p$. La complexité de la clause C, notée Rep(C), est*

$$max(\uplus_{i=1}^{m} Comp(e_i^n) \uplus \uplus_{j=1}^{k} Comp(e^p{}_j))$$

L'ordre sur les clauses, $\prec_c$, est défini par

$$C_1 \prec_c C_2 \text{ ssi } Rep(C_1) \ll_t Rep(C_2)$$

Le pré-ordre sur les clauses, $\preceq_c$, est défini par

$$C_1 \preceq_c C_2 \text{ ssi } Rep(C_1) \lessdot_t Rep(C_2)$$

Si la partie stricte de $\leq_t$ est un ordre de réduction, alors $\preceq_c$ est un pré-ordre bien fondé fortement stable [Naidich, 1996].

Exemple 1.4.1 (comparaison de deux clauses) *Soient $\mathsf{F} = \{s, \text{True}, c, minus1, eqc, +, <, ml, mc, count\}$, $\prec_{\mathsf{F}}$ un pré-ordre sur les éléments de F tel que*

$$s \prec_{\mathsf{F}} \text{True} \prec_{\mathsf{F}} c \prec_{\mathsf{F}} minus1 \prec_{\mathsf{F}} eqc \prec_{\mathsf{F}} + \prec_{\mathsf{F}} < \prec_{\mathsf{F}} ml \prec_{\mathsf{F}} count \quad \text{et} \quad mc \sim_{\mathsf{F}} ml$$

et deux clauses

$$eqc(c(j), mc(p, i)) = \text{True} \vee (minus1(ml(p, i)) + count(p, i, c(j))) < s(s(i + minus1(ml(p, i)))) = \text{True} \quad (C_1)$$
$$eqc(c(j), mc(p, i)) = \text{True} \vee (ml(p, i) + count(p, i, c(j))) < s(i + ml(p, i)) = \text{True} \quad (C_2)$$

On a True $<_{MPO}$ $eqc(c(j), mc(p, i)) <_{MPO} (ml(p, i)+count(p, i, c(j))) < s(i+ml(p, i)) <_{MPO}$ $(minus1(ml(p, i))+count(p, i, c(j))) < s(s(i+minus1(ml(p, i))))$. *Donc on obtient successivement que* $max(Comp(C_1)) = \{(minus1(ml(p, i))+count(p, i, c(j))) < s(s(i+minus1(ml(p, i))))\}$ *et* $max(Comp(C_2)) = \{(ml(p, i)+count(p, i, c(j))) < s(i+ml(p, i))\}$, *puis* $max(comp(C_2)) \ll_t max(comp(C_1))$. *Donc,* $C_2 \prec_c C_1$.

1.5 Spécifications conditionnelles et systèmes de réécriture conditionnelle

Les équations conditionnelles sont largement utilisées dans la définition des types abstraits algébriques [Ehrig et Mahr, 1985] grâce à leur grande expressivité et leur caractère opérationnel. Dans cette section, on va étudier la sémantique dénotationnelle et opérationnelle des spécifications conditionnelles. Pour plus d'informations, le lecteur est invité à consulter [Dershowitz et Jouannaud, 1990; Padawitz, 1988; Ehrig et Mahr, 1985; Baader et Nipkow, 1998].

1.5.1 Spécifications conditionnelles

Une *clause de Horn* est une clause particulière de la forme $a_1 \wedge \ldots \wedge a_n \Rightarrow a'_1$, ou bien $a_1 \wedge \ldots \wedge a_n \Rightarrow$, ou bien a'_1, où les formules atomiques $a_1, \ldots, a_n$ s'appellent les *conditions* de la clause et la formule atomique a'_1 sa *conclusion*.

On rappelle qu'une équation est une paire (s, t), avec s et t deux termes de même sorte, écrite $s = t$. Une *équation conditionnelle* est une clause de Horn dont les formules atomiques sont des équations. Formellement, une *spécification conditionnelle multi-sortée*, ou plus simplement spécification conditionnelle, est formée d'une signature et d'un ensemble d'équations conditionnelles, nommées les *axiomes* de la spécification. Elles servent à définir des symboles de fonctions de la signature.

Exemple 1.5.1 *Reprenons la signature de l'exemple 1.1.1. Une spécification conditionnelle est obtenue en ajoutant les équations conditionnelles suivantes :*

$$0 \leq x = \text{True} \tag{1.1}$$
$$s(x) \leq 0 = \text{False} \tag{1.2}$$
$$s(x) \leq s(y) = x \leq y \tag{1.3}$$
$$x \leq y = \text{True} \Rightarrow min(x, y) = x \tag{1.4}$$
$$x \leq y = \text{False} \Rightarrow min(x, y) = y \tag{1.5}$$

La fonction min est censée calculer le minimum entre deux naturels, tandis que $\leq$ *représente l'opération de «plus petit ou égal» sur des naturels.*

Interprétation et initialité dans les spécifications conditionnelles

Dans le cas des langages dont les formules sont des équations conditionnelles, l'ensemble P ne contient que les symboles de prédicats d'égalité associés à chaque sorte. Par conséquent, toute fonction d'interprétation $\mathbb{I}$ va contenir uniquement la composante sur les symboles de sortes $\mathbb{I}_\mathsf{S}$

et celle sur les symboles de fonctions $\mathbb{I}_{\mathsf{F}}$, car la composante $\mathbb{I}_{\mathsf{P}}$ sur les symboles de prédicats peut être engendrée par $\mathbb{I}_{\mathsf{S}}$. On va appeler ces interprétations particulières des *algèbres*.

La relation d'égalité induite par l'ensemble d'axiomes E d'une spécification conditionnelle établit une relation de congruence entre les éléments de $\mathcal{T}(\mathsf{F})$. Un cas particulier d'algèbre, appelé *algèbre initiale* ou *algèbre quotient des termes*, peut être récursivement construit à partir des interprétations définies sur le domaine d'interprétation $\mathcal{T}(\mathsf{F})$ partagé en classes de congruences.

Définition 1.5.1 (algèbre quotient des termes) *Soit $\approx_E$ une congruence sur $\mathcal{T}(\mathsf{F})$; l'ensemble $\mathcal{T}(\mathsf{F})/\approx_E$ des classes de congruence $[t]_{\approx_E}$ de $\mathcal{T}(\mathsf{F})$ constitue le domaine d'interprétation. L'algèbre quotient des termes, notée par $\mathbb{I}(\mathsf{F}, E)$, est l'algèbre telle que*

$$f_{\mathcal{T}(F)/\approx_E}([t_1]_{\approx_E}, \ldots, [t_n]_{\approx_E}) = [f(t_1, \ldots, t_n)]_{\approx_E},$$

pour tous $f : S_1 \times \cdots \times S_n \to S$ appartenant à F, t_1 à $\mathcal{T}(\mathsf{F})_{S_1}, \ldots, t_n$ à $\mathcal{T}(\mathsf{F})_{S_n}$.

La relation $\approx_E$ est la plus petite congruence engendrée par E sur $\mathcal{T}(\mathsf{F})$ qui se construit par récurrence de la façon suivante :

- *soit $\approx_0$ la plus petite relation de congruence qui contient la relation ϕ_0 définie par : $u\phi_0 v$ si et seulement si il existe une équation $s = t$ dans E et une substitution $\sigma : \mathsf{V} \to \mathcal{T}(\mathsf{F})$ telles que $s\sigma \equiv u$ et $t\sigma \equiv v$.*
- *soit $\approx_{i+1}$ la plus petite congruence qui contient la relation ϕ_{i+1} sur $\mathcal{T}(\mathsf{F})$ définie par $u\phi_{i+1}v$ si et seulement si $u \approx_i v$ ou s'il existe une équation conditionnelle $\wedge_{j=1}^{n} u_j = v_j \Rightarrow s = t$ dans E, une substitution $\sigma : \mathsf{V} \to \mathcal{T}(\mathsf{F})$ et une position $p \in pos(u)$ telles que $s\sigma \equiv u/p$, $u[t\sigma]_p \equiv v$ et pour tout j dans $[1..n]$, $u_j\sigma \approx_i v_j\sigma$.*

Alors $\approx_E$ est définie comme $\cup_{i\geq 0}\{\approx_i\}$.

L'algèbre quotient des termes clos par la congruence $\approx_E$ est un modèle de E, si E est cohérente. De plus, elle est initiale dans tous les modèles satisfaisant les équations conditionnelles de E et est unique à un isomorphisme près [Grätzer, 1979]. Dans la suite, on va instancier λ de la notation $\models_\lambda$ avec *ini* afin d'exprimer la relation de conséquence initiale.

Définition 1.5.2 (conséquence initiale) *Une clause équationnelle C est une* conséquence initiale *(ou initialement valide) dans la théorie de E, notée $E \models_{ini} C$, si C est valide dans $\mathbb{I}(\mathsf{F}, E)$.*

On donne ensuite un théorème de caractérisation opérationnelle des conséquences initiales, dérivé de la Proposition 3.1 de [Bouhoula, 1997] † :

Théorème 1.5.1 (caractérisation opérationnelle des conséquences initiales) *Etant donnée une clause équationnelle C, on a $E \models_{ini} C$ ssi, pour toute substitution close σ, on a $E \models_{ini} C\sigma$.*

† Le lecteur peut aussi consulter la preuve du théorème 1.28 du [Bouhoula, 1994].

1.5.2 Systèmes de réécriture conditionnelle

Les systèmes d'inférence pour la déduction équationnelle ne sont pas efficaces pour dériver des conséquences initiales d'un ensemble d'équations. Par exemple, avec le système E de la figure 1.1, on peut avoir des dérivations infinies à cause des règles SYMMETRY ou EXPANSION.

Exemple 1.5.2 *Soit* $(\{a = b\}, \emptyset)$ *l'état initial d'une* E*-dérivation. La dérivation* $(\{a = b\}, \emptyset) \vdash^{\mathsf{E}} (\{b = a\}, \emptyset) \vdash^{\mathsf{E}} (\{a = b\}, \emptyset) \ldots$ *est une dérivation infinie obtenue par application successive de la règle* SYMMETRY.

Ces dernières décennies, de nombreux efforts ont été consacrés à trouver des restrictions à ces systèmes qui préservent cependant les propriétés de complétude et correction. Une solution est d'orienter des équations de la forme $s = t$ et de les transformer en *règles de réécriture* de la forme $s \to t$ avec la signification opérationnelle que toute occurrence d'une instance $s\sigma$ de s peut être remplacée par $t\sigma$ (mais non l'inverse). L'ensemble des règles de réécriture obtenues à partir d'un ensemble d'axiomes Ax forment un *système de réécriture*. Pour plus de détails sur les systèmes de réécriture, le lecteur peut consulter les références suivantes [Dershowitz et Jouannaud, 1990; Klop, 1992; Plaisted, 1993].

Dans la suite, on s'intéresse à l'étude des propriétés des clauses équationnelles dans le modèle initial avec des systèmes de réécriture obtenus à partir d'un ensemble d'équations conditionnelles. Une *règle conditionnelle* $\wedge_{i=1}^{n} l_i = r_i \Rightarrow s \to t$ est dérivée d'une équation conditionnelle $\wedge_{i=1}^{n} l_i = r_i \Rightarrow s = t$ dont la conclusion $s = t$ est orientée de la gauche vers la droite. Le terme s (resp. t) s'appelle membre gauche (resp. droit) de la règle. L'application de cette règle conditionnelle sur un terme $u[s\sigma]$ avec la substitution σ consiste à remplacer dans u l'instance $s\sigma$ par $t\sigma$ si l'instance des préconditions de la règle $\wedge_{i=1}^{n} l_i\sigma = r_i\sigma$ est valide. On dit qu'on a appliqué sur u une *opération de réécriture*. Un *système de réécriture conditionnelle* est un ensemble de règles de réécriture conditionnelles. Un système de réécriture est *linéaire gauche* si toutes ses règles de réécriture ont le membre gauche linéaire.

Une caractéristique importante d'un système de réécriture R est sa *profondeur*, notée $depth(R)$, qui est la profondeur maximale des membres gauches de toutes les règles conditionnelles de R. De la même façon, on note par $sdepth(R)$ la *profondeur stricte* de R, qui représente la profondeur stricte maximale des membres gauches de toutes les règles conditionnelles de R.

Un système de réécriture conditionnelle R introduit la relation binaire $\to_R$ sur les termes, définie ci-dessous.

Définition 1.5.3 (relation de réécriture conditionnelle, joignabilité) *Soient* R *un système de réécriture conditionnelle,* u *un terme,* p *une position dans* u, $\wedge_{i=1}^{n} l_i = r_i \Rightarrow s \to t$ *une règle conditionnelle et* σ *une substitution. On écrit*

$$u[s\sigma] \to_R u[t\sigma]$$

si les termes $l_i\sigma$ *et* $r_i\sigma$ *sont* joignables*, pour tout* $i \in [1..n]$*, c'est-à-dire qu'il existe un terme* c *tel que* $l_i\sigma \to_R^* c$ *et* $r_i\sigma \to_R^* c$.

Une *dérivation d'un système de réécriture conditionnelle* est une suite d'opérations de réécriture. Il se peut que la propriété de terminaison ne soit pas satisfaite en raison des dérivations

infinies horizontales, comme on a déjà vu dans l'exemple 1.5.2, ou verticales lors de l'évaluation récursive des conditions.

Exemple 1.5.3 (dérivations infinies horizontales et verticales) *Si on considère la règle de réécriture* $x \leq y \rightarrow s(x) \leq s(y)$ *de* R*, alors on peut avoir une dérivation infinie horizontale obtenue par l'application successive des opérations de réécriture sur le terme* $x \leq y$ *en utilisant des instances de cette règle :* $x \leq y \rightarrow_R s(x) \leq s(y) \rightarrow_R s(s(x)) \leq s(s(y)) \rightarrow_R \ldots$.

D'autre part, si on veut appliquer la règle $s(x) \leq s(y) = \text{True} \Rightarrow x \leq y \rightarrow \text{True}$ *de* R *pour réécrire la conjecture* $x \leq y = \text{True}$ *sur le terme* $x \leq y$*, alors la condition à évaluer est* $s(x) \leq s(y) = \text{True}$ *qui demande à son tour l'évaluation de la condition* $s(s(x)) \leq s(s(y)) = \text{True}$ *de l'instance* $s(s(x)) \leq s(s(y)) = \text{True} \Rightarrow s(x) \leq s(y) \rightarrow \text{True}$ *de la règle, etc.*...

Dans un système de déduction basé sur la réécriture, la propriété de terminaison de toute dérivation est essentielle pour l'application automatique de la méthode. La définition suivante d'une règle de réécriture conditionnelle, similaire à [Kaplan, 1984; Kaplan, 1987], est suffisante pour assurer la terminaison de $\rightarrow_R$:

Définition 1.5.4 (règle de réécriture conditionnelle) *Etant donné un ordre* $<_t$ *sur les termes qui est bien fondé et stable par substitution, alors* $\wedge_{i=1}^{n} l_i = r_i \Rightarrow s \rightarrow t$ *est une* règle de réécriture conditionnelle *si* $Var(l_i) \cup Var(r_i) \subseteq Var(s)$, $Var(r_i) \subseteq Var(l_i)$*, pour tout* $i \in [1..n]$*, et*

- $t <_t s$
- $\bigcup_{i=1}^{n} \{l_i, r_i\} \ll_t \{s\}$

pour éviter respectivement des dérivations infinies horizontales et verticales.

Définition 1.5.5 (forme normale) *Soit* R *un système de réécriture. On dit qu'un terme* t *est en* forme normale *pour* R*, s'il n'existe pas de terme* t' *tel que* $t \rightarrow_R t'$*. On dit aussi que* t *est* R-irréductible.

On distingue deux notions d'irréductibilité des termes ou des clauses équationnelles, lorsqu'il s'agit de systèmes de réécriture conditionnelle : *faible* et *forte*.

Définition 1.5.6 (terme et clause équationnelle faiblement irréductibles) *Soient* R *un système de réécriture conditionnelle. Un terme* t *est* faiblement R-irréductible *si pour tout sous-terme* s *de* t *tel qu'il existe une règle* $\wedge_{i=1}^{n} l_i = r_i \Rightarrow g \rightarrow d$ *dans* R *et une substitution* σ *avec* $s = g\sigma$ *et pour toute substitution close* τ*, on a* $Ax \not\models_{ini} (\wedge_{i=1}^{n} l_i = r_i)\sigma\tau$.

La clause équationnelle $u_1 = v_1 \vee \ldots \vee u_n = v_n$ *est* faiblement irréductible *si* $u_i \not\equiv v_i$ *et l'élément maximal (resp. les éléments maximaux) de* $\{u_i, v_i\}$ *par rapport à* $\leq_t$ *est (resp. sont) faiblement* R*-irréductible(s), pour tout* i.

Définition 1.5.7 (terme et clause équationnelle fortement irréductibles) *Soient* R *un système de réécriture conditionnelle et* t *un terme. Si, pour tout sous-terme* s *de* t*, il n'y a pas de règle de* R *dont le membre gauche filtre* s*, on dit que* t *est fortement* R*-irréductible.*

Une clause équationnelle $u_1 = v_1 \vee \ldots \vee u_n = v_n$ *est fortement* R*-irréductible si tous les termes* u_i *et* v_i *sont fortement* R*-irréductibles, pour tout* $i \in [1..n]$.

Si la relation $\rightarrow_R$ est un ordre décroissant, alors R peut être utilisé pour décider si un terme est en forme normale ou si deux termes sont joignables [Dershowitz *et al.*, 1988].

La relation $\rightarrow_R$ est *non-déterministe* car un terme peut se réduire en plusieurs positions. La propriété de *confluence* d'un système de réécriture assure que l'ordre d'application des opérations de réécriture n'a pas d'importance pour déterminer la forme normale d'un terme.

Définition 1.5.8 (confluence, confluence sur les termes clos) *Etant donné un système de réécriture conditionnelle R, la relation $\rightarrow_R$ est dite* confluente (confluente sur les termes clos) *si, pour tous termes (clos) t_1 et t_2 tels que $t \rightarrow_R^* t_1$ et $t \rightarrow_R^* t_2$, il existe t' (clos) tel que $t_1 \rightarrow_R^* t'$ et $t_2 \rightarrow_R^* t'$.*

Une relation de réécriture bien fondée et confluente sur les termes clos est *convergente sur les termes clos*. Cette propriété assure l'existence, par la propriété de terminaison, et l'unicité, par la propriété de confluence, de la forme normale de tout terme. Alors R peut être utilisée pour décider l'égalité de deux termes : deux termes sont égaux s'ils ont la même forme normale.

Plusieurs méthodes ont été proposées pour vérifier la propriété de confluence sur les termes clos d'un système de réécriture, parmi lesquelles les techniques de saturation de [Rusinowitch, 1989; Kounalis et Rusinowitch, 1990; Kounalis et Rusinowitch, 1991] qui, par rapport à [Rémy, 1982; Kaplan, 1987], fonctionnent même si les équations ne sont pas orientables ou si l'ordre induit par les règles n'est pas décroissant. D'autres travaux plus récents sont [Becker, 1996; Bouhoula, 1999].

Complétude suffisante des spécifications conditionnelles

Généralement, le problème de complétude des fonctions récursives [Guttag, 1975] d'une spécification algébrique [Ehrig et Mahr, 1985] (dont les spécifications conditionnelles sont des cas particuliers) consiste à savoir si les axiomes de la spécification *suffisent* à décrire (où spécifier) de manière complète le système. Même pour le cas des spécifications conditionnelles, le problème est indécidable, mais il existe des critères syntaxiques pour prouver cette propriété en posant des restrictions sur les spécifications. Dans la suite, on analyse ce problème lorsque l'ensemble de symboles de fonctions F est partitionné en des symboles de *fonctions constructeures*[†] CT et des symboles de *fonctions définies* D à l'aide des constructeurs. Formellement, on a $\mathsf{F} = \mathsf{CT} \uplus \mathsf{D}$. S'il n'y a pas de règle de réécriture dont le membre gauche a comme racine un symbole de constructeur c, on dit que c est un symbole de *constructeur libre*.

Tout au long de ce mémoire, nous supposons que les systèmes de réécriture dérivés à partir des spécifications conditionnelles basées sur des constructeurs sont *structurés*.

Définition 1.5.9 (système de réécriture structuré) *Soit $\leq_{\mathsf{F}}$ un pré-ordre sur les éléments de F et R un système de réécriture. Alors R est* structuré *si*

- *pour toute règle de réécriture $\wedge_{i=1}^{n} l_i = r_i \Rightarrow s \rightarrow t$ de R, si $s \in \mathcal{T}(\mathsf{CT}, \mathsf{V})$, alors $t \in \mathcal{T}(\mathsf{CT}, \mathsf{V})$, et*
- *pour tout symbole de fonction définie f_{D} il n'existe pas de symbole de constructeur f_{CT} tel que $f_{\mathsf{D}} \leq_{\mathsf{F}} f_{\mathsf{CT}}$.*

† appelés plus simplement des *constructeurs*

Définition 1.5.10 (terme constructeur, complétude suffisante) *Tout terme appartenant à* $\mathcal{T}(\mathsf{CT}, \mathsf{V})$ *est un* terme constructeur.

Soient E une spécification conditionnelle et R le système de réécriture associé. Alors, un symbole de fonction $f \in \mathsf{D}$ *est* suffisamment complet *ssi, pour tout terme clos* $f(t_1, \ldots, t_n)$ *avec* $t_1, \ldots, t_n \in \mathcal{T}(\mathsf{CT})$*, il existe un terme constructeur clos* $t \in \mathcal{T}(\mathsf{CT})$ *tel que* $f(t_1, \ldots, t_n) \rightarrow_R^* t$*. On dit que R est* suffisamment complète *si tout* $f \in \mathsf{D}$ *est suffisamment complet.*

Pour tester la complétude des spécifications conditionnelles, quelques critères ont été proposés, notamment dans [Bousdira, 1990; Bousdira et Rémy, 1990] et [Bouhoula *et al.*, 1995], en se basant sur les travaux de Kounalis [Kounalis, 1985b; Kounalis, 1985a]. A partir de la notion de couverture d'une disjonction de préconditions, Bousdira et Rémy ont mis au point une méthode de test de complétude suffisante lorsque les axiomes sont à préconditions booléennes et définissent un système de réécriture décroissant. Bouhoula, Kounalis et Rusinowitch ont étendu leurs résultats aux équations conditionnelles, pas nécessairement à préconditions booléennes. L'approche reste valide même lorsqu'il y a des relations entre les constructeurs. Leur procédure, implantée dans le système SPIKE [Bouhoula et Rusinowitch, 1995b], apporte en outre une aide à la construction des définitions complètes lorsque les spécifications sont à préconditions booléennes. Un exemple de cette procédure sera présenté dans la section 7.6.3 du chapitre 7. Cette méthode est améliorée dans [Bouhoula, 1999] et permet de tester simultanément les propriétés de complétude et de confluence sur les termes clos.

1.6 Procédures de décision

Les démonstrateurs de théorèmes et les outils de raisonnement automatique sont souvent difficiles à utiliser ; la raison principale est que l'utilisateur doit accéder à des bases de connaissances de grande taille, écrites par des experts, afin d'aboutir à la preuve avec succès. Ceci demande de l'interaction de l'utilisateur avec l'outil, ce qui implique une connaissance approfondie de ses mécanismes de raisonnement. C'est pour cela qu'une condition *sine qua non* à satisfaire par les démonstrateurs automatiques est de pouvoir exécuter automatiquement les tâches routinières ou de larges étapes d'inférences sur certains domaines d'application. Une solution à ce problème est l'utilisation des procédures de (semi-)décision dans des cadres logiques permettant leur combinaison et leur intégration dans les prouveurs. Dans cette section, on va détailler seulement ce qu'on comprend par une procédure de (semi-)décision. On va développer un schéma d'intégration dans le chapitre 3, ainsi qu'un schéma de coopération dans le chapitre 6.

Intuitivement, une *procédure de décision* prend comme entrée une propriété (ou formule) à vérifier sur un certain domaine d'application et « répond » en *temps fini* si la propriété est satisfaite ou pas. On peut rencontrer des procédures de décision pour l'égalité, les nombres (entiers, naturels, rationnels), les booléens, les vecteurs de bits, les ensembles finis ou les listes finies. Si la procédure peut échouer sur une des deux alternatives, alors il s'agit d'une *procédure de semi-décision*.

Le raisonnement sur les nombres est très utilisé en vérification [Boyer et Moore, 1985], en particulier si les spécifications contiennent des structures de données linéaires comme les listes ou les tableaux, où on manipule des indices naturels ou entiers. Dans la suite, on va présenter des procédures de (semi-)décision pour une sous-théorie de l'*arithmétique de Presburger* [Enderton, 1977], connue aussi sous le nom d'*arithmétique linéaire* [Boyer et Moore, 1985].

Définition 1.6.1 (arithmétique linéaire, ses termes et ses formules atomiques) *L'arithmétique linéaire représente la théorie des nombres (entiers, naturels ou rationnels) sans quantificateurs, constituée par des variables numériques, des symboles de relations arithmétiques ($\geq$, $>$, $\leq$, $<$, $=$, $\neq$), les opérateurs d'addition et de soustraction. Dans sa version pure*[†]*, elle ne contient que ces symboles, mais nous considérons aussi les* symboles de fonctions non-interprétés*, qui ne sont pas contraints à avoir de propriétés.*

Un terme de l'arithmétique linéaire *est soit un nombre, soit une variable, soit un terme de la forme $f(t_1, \ldots, t_n)$, où f est un symbole d'une opération arithmétique, et chaque t_i est un terme général (pas forcément de l'arithmétique linéaire). Une* formule atomique de l'arithmétique linéaire *est de la forme $p(t_1, \ldots, t_n)$, où p est un symbole d'une relation arithmétique.*

Si n est un naturel et t un terme de l'arithmétique linéaire, par les notations $n*t$ et $(-n*t)$, on comprend respectivement $\underbrace{t+t+\cdots+t}_{n \text{ fois}}$ et $\underbrace{-t-t-\cdots-t}_{n \text{ fois}}$.

L'algorithme de Fourier-Motzkin, décrit en détail dans [Lassez et Maher, 1992], représente une procédure de décision pour les rationnels et il peut être utilisé pour vérifier l'incohérence d'un ensemble (vu comme un système) d'inégalités linéaires sur les entiers ou les naturels. Ceci est dû au fait que tout système d'inégalités qui n'a pas de solutions sur les rationnels, n'a pas non plus de solutions sur les entiers ou les naturels. Par les transformations équivalentes affichées dans le tableau 1.1 :

formule	$s \geq t$	$s > t$	$s < t$	$s = t$	$s \neq t$
transformation	$t \leq s$	$t+1 \leq s$	$s+1 \leq t$	$s \leq t \wedge t \leq s$	$s+1 \leq t \vee t+1 \leq s$

TAB. 1.1 – *Transformations de formules atomiques de l'arithmétique linéaire*

où s et t sont des termes naturels (resp. entiers), toute formule atomique de l'arithmétique linéaire sur les naturels (resp. entiers) peut se transformer dans i) des formules ne contenant que des formules atomiques de l'arithmétique linéaire de la forme $s \leq t$, ou bien ii) des conjonctions ou disjonctions de ce type de formules atomiques.

1.6.1 Procédures de (semi-)décision pour l'arithmétique linéaire

On va montrer un exemple de procédure de (semi-)décision pour l'arithmétique linéaire basée sur l'algorithme de Fourier-Motzkin. Cet algorithme prend en entrée un ensemble d'*inégalités linéaires* de la forme

$$(\Sigma_{j=1}^{n} a_j * t_j) - b \leq 0$$

où b, $a_1, \ldots, a_n$ sont des entiers et $t_1, \ldots, t_n$ des termes de l'arithmétique linéaire, appelés des *monômes*. Notons que toute formule atomique de l'arithmétique linéaire de la forme $s \leq t$ peut se transformer dans l'inégalité $s - t \leq 0$.

Définition 1.6.2 (inégalité linéaire impossible) *On dit qu'une inégalité linéaire $(\Sigma_{j=1}^{n} a_j * t_j) - b \leq 0$ est* impossible *si $b < 0$ et $a_j = 0$, pour tout $1 \leq j \leq n$.*

† Presburger a montré en 1929 que la théorie quantifiée du premier ordre sur les nombres, ne contenant que ces symboles, est décidable.

Pour simplifier, on va nommer les inégalités linéaires des inégalités. Soient P et P' deux ensembles d'inégalités. On écrit $P \Leftrightarrow_A P \cup P'$ si toute inégalité de P' est une combinaison linéaire non-négative des inégalités de P, obtenues par des opérations de « multiplication croisée et addition ». Ces opérations permettent d'éliminer successivement des monômes des inégalités. Par exemple, si on a les inégalités

$$2 * y - i \leq 0 \tag{1.6}$$

$$-y + i \leq 0 \tag{1.7}$$

on peut dériver l'inégalité $i \leq 0$ par la multiplication par 2 de l'inégalité (1.7) et puis par l'addition de l'inégalité (1.6).

Plus formellement, si I est l'inégalité $(\Sigma_{j=1}^{n} a_j * t_j) - b \leq 0$ et α est un naturel, par αI on dénote l'inégalité $(\Sigma_{j=1}^{n}(\alpha * a_j) * t_j) - \alpha * b \leq 0$, obtenue par la multiplication de I par α. La somme $I_1 + I_2$ des inégalités I_1 $((\Sigma_{j=1}^{n} a'_j * t'_j) - b' \leq 0)$ et I_2 $((\Sigma_{j=1}^{n} a_j * t_j) - b \leq 0)$ est l'inégalité $(\Sigma_{j=1}^{n}(a'_j * t'_j + a_j * t_j)) - (b' + b) \leq 0$. Une combinaison linéaire non-négative des inégalités $I_1, \ldots, I_n$ est l'inégalité $\Sigma_{j=1}^{n} \alpha_j I_j$, avec α_j naturel, pour tout $j \in [1..n]$.

Par une opération de « multiplication croisée et addition » entre deux inégalités, on élimine un monôme commun aux deux. Si I_i est une inégalité avec un coefficient c_i du monôme t et I_j une inégalité avec un coefficient c_j de x telles que $c_i * c_j < 0$, alors $|c_j|I_i + |c_i|I_j$ est une inégalité qui ne contiendra pas x, où $|c|$ est l'opération « module » sur les entiers. Toute solution de I_i et I_j est une solution de $|c_j|I_i + |c_i|I_j$ et toute solution de $|c_j|I_i + |c_i|I_j$ peut être étendue pour obtenir une solution de I_i et I_j. Si t est un monôme d'une inégalité I d'un ensemble d'inégalités P et il n'y a pas d'autres inégalités contenant des coefficients de t de signe opposé, alors on peut éliminer de P toutes les inégalités qui contiennent t parce qu'on peut toujours trouver une valeur adéquate de t qui les satisfait. L'ordre d'élimination des variables dans une opération de « multiplication croisée et addition » est dicté par un ordre sur les monômes ; dans une inégalité, on élimine d'abord les monômes les plus grands par rapport à cet ordre. Un pré-ordre total sur les monômes sera présenté dans la section 6.3.

Si l'algorithme termine sans trouver une inégalité impossible, la procédure échoue pour les naturels ou les entiers sans pouvoir déterminer si l'ensemble initial d'inégalités est cohérent ou pas. Par exemple, l'inégalité (qui n'est pas impossible) $3 * x \leq 2$ a une solution sur les rationnels, mais pas sur les naturels. Sinon, l'existence d'une inégalité impossible garantit l'incohérence de l'ensemble initial d'inégalités sur les rationnels, entiers et naturels. Ceci est dû au fait que toute inégalité de l'ensemble final d'inégalités est une combinaison linéaire non-négative des inégalités initiales. Formellement, on écrit $\not\models_{LA} P$ si P contient des inégalités impossibles ou il existe P' tel que $P \Leftrightarrow_A^* P'$ et P' contient une inégalité impossible. Pour que le test d'incohérence fonctionne aussi pour les naturels, il faut ajouter à l'ensemble initial d'inégalités P des inégalités de la forme $-x \leq 0$, pour toute variable x de P.

L'algorithme peut être utilisé pour dériver aussi des *équations implicites* à partir de l'ensemble initial d'inégalités, selon les théorèmes suivants de Lassez et Maher [Lassez et Maher, 1992] et Käufl [Käufl, 1988], aussi présentés dans [Kapur et Nie, 1994]. Formellement, si I est une inégalité $(\Sigma_{j=1}^{n} a_j * t_j) - b \leq 0$, par $I^{=}$ on note l'équation $(\Sigma_{j=1}^{n} a_j * t_j) - b = 0$, où a_j, b sont des entiers et t_j des termes de l'arithmétique linéaire, pour tout $j \in [1..n]$.

Définition 1.6.3 (équation implicite) *Soit P est un ensemble d'inégalités. Alors $I^{=}$ est une* équation implicite*, dérivée de P pour un domaine donné, si I est une inégalité de P telle que $P \models_{LA} I^{=}$ dans ce domaine, où le domaine peut être les rationnels, les entiers ou les naturels.*

Théorème 1.6.1 ([Käufl, 1988]) *Si P implique une égalité e, alors il existe un sous-ensemble fini $\{I_1, \ldots, I_n\}$ de P tel que $P \models_{LA} e$ ssi $\{I_1^=, \ldots, I_n^=\} \models_{LA} e$.*

Théorème 1.6.2 ([Lassez et Maher, 1992]) *Si $(\Sigma_{k=1}^{m} \alpha_k * I_k) \equiv (0 \leq 0)$ tel que pour tout $k \in [1..m]$ on a $I_k \in P$ et α_k est un naturel, alors toute inégalité I_j, avec $j \in [1..m]$, est une équation implicite.*

Théorème 1.6.3 ([Lassez et Maher, 1992]) *Une inégalité I_k dans un ensemble cohérent d'inégalités P est une équation implicite pour les rationnels ssi, par l'algorithme de Fourier-Motzkin, on dérive l'inégalité $0 \leq 0$ en utilisant I_k.*

Il existe plusieurs algorithmes permettant de décider des propriétés dans l'arithmétique de Presburger, comme celui de Cooper [Cooper, 1972], introduit au début des années 70. Cinq ans plus tard, Shostak [Shostak, 1977] présente une nouvelle méthode en utilisant une procédure basée sur la méthode *sup-inf* de Bledsoe [Bledsoe, 1975; Bledsoe et Shostak, 1979] pour résoudre des inégalités linéaires sur des rationnels. Un autre algorithme, basé sur la programmation linéaire entière, a été proposé par Shostak [Shostak, 1979] pour manipuler des formules de l'arithmétique linéaire avec des symboles de fonctions non-interprétés. De même, Nelson et Oppen [Nelson et Oppen, 1979] ont présenté un algorithme basé sur la méthode simplexe pour résoudre des inégalités linéaires sur les rationnels.

Les semi-procédures de décision de l'arithmétique linéaire basées sur la méthode de Fourier-Motzkin pour les entiers et les naturels peuvent être utilisées pour montrer par réfutation la validité inductive des clauses, car $\models_{PA} \subseteq \models_{ind}$ (les termes clos du domaine d'interprétation sont soit des naturels, soit des entiers).

Soit Ax un ensemble d'axiomes d'une spécification conditionnelle qui est compatible avec la théorie de l'arithmétique linéaire telle que toute clause dont la validité est montrée par réfutation, par la procédure de décision, est une conséquence inductive des axiomes. En particulier, dans la spécification on ne définit pas les symboles d'addition et de soustraction ou les relations arithmétiques $\geq$, $>$, $\leq$, $<$[†]. On suppose qu'il existe une fonction de traduction, $\mathcal{TR}$, qui extrait d'une clause une disjonction de conjonctions d'inégalités linéaires telle que, pour toute clause C, il résulte $Ax \models_{ind} C$ lorsque $\mathcal{TR}(\neg C) = \bigvee_{i=1}^{n} P_i$ et $\not\models_{LA} P_i$, pour tout $i \in [1..n]$.

Exemple 1.6.1 *Soit C la clause $2*(x+y) \geq (i+x+1) = True \vee 2*(x-1+y) < (1+i+x) = True$. On suppose que la fonction de traduction $\mathcal{TR}$ transforme $\neg C$ dans le système d'inégalités linéaires*

$$x + 2 * y - i \leq 0 \tag{1.8}$$

$$-x - 2 * y + i + 3 \leq 0 \tag{1.9}$$

Le processus de « multiplication croisée et addition » consiste dans ce cas en l'addition des inégalités (1.8) et (1.9), afin d'obtenir l'inégalité linéaire impossible $3 \leq 0$. Alors C est inductivement valide.

† Des propriétés sur ces symboles seront considérées comme des lemmes.

2

Ensembles couvrants contextuels

Sommaire

2.1 Introduction

Comme on a déjà vu dans le chapitre 1, le domaine d'interprétation caractérise le type de relation de conséquence entre les axiomes et les conjectures. En fait, son choix dépend de la nature des spécifications. Dans le cas de spécifications algébriques, les domaines qui présentent intérêt son souvent des sous-ensembles du domaine de Herbrand. Par exemple, si les spécifications sont équationnelles, le domaine d'interprétation adéquate est formé par les représentants des classes de congruence sur le domaine de Herbrand, induites par les axiomes (voir la notion de conséquence initiale de la section 1.5.1).

Dans ce chapitre, nous sommes intéressés premièrement à étudier les relations de conséquence qui satisfont la propriété suivante : une conjecture est une conséquence des axiomes si et seulement si toutes ses instances closes le sont. Puis, nous présentons le concept d'ensemble couvrant contextuel comme une caractérisation de l'ensemble de ces instances closes. Enfin, nous introduisons quelques propriétés de composition des ensembles couvrants contextuels.

2.2 Conséquences inductives raffinées

Soit Ax un ensemble d'axiomes. On définit une classe de relations de conséquences, appelées des conséquences inductives raffinées, d'une manière opérationnelle comme suit :

Définition 2.2.1 (conséquence inductive raffinée) *La formule (conjecture) ϕ du langage $\mathcal{L}$ est une* conséquence inductive raffinée *de Ax, dénoté par $Ax \models_{ind^*} \phi$, ssi pour toute substitution close γ, on a $Ax \models_{ind^*} \phi\gamma$.*

A part les conséquences inductives, il existe d'autres relations de conséquence qui font partie de cette classe. Les relations suivantes sont définies lorsque les conjectures sont des clauses équationnelles :

1. *relation de conséquence initiale*. Elle est utilisée dans le cadre des spécifications équationnelles ou conditionnelles. Le théorème 1.5.1, qui donne une caractérisation opérationnelle des conséquences initiales, est une instance de la définition 2.2.1.
2. *relation de conséquence observable*. Elle peut être utilisée lorsqu'il existe des différences entre les propriétés énoncées par une spécification et son implémentation [Bernot *et al.*, 1994]. L'implémentation peut être considérée correcte si ces différences ne sont pas observables, dans le sens que son comportement externe satisfait les propriétés de la spécification. Dans [Berregeb *et al.*, 1998], on définit une relation de conséquence observable adéquate aux spécifications conditionnelles. Son théorème de caractérisation opérationnelle, représenté par le théorème 1 du [Berregeb *et al.*, 1998], montre que cette relation est une conséquence inductive raffinée.
3. *relation de conséquence dans les spécifications paramétrées*. Ce type de spécifications algébriques multi-sortées [Navarro et Orejas, 1987; Kirchner, 1991] admet des sortes paramétrées. La relation de conséquence est définie sur un domaine d'interprétation formé seulement par des termes clos de sorte non-paramétrée. Pour des spécifications conditionnelles paramétrées, on définit dans [Bouhoula, 1996] une relation de conséquence qui, suite au lemme 3.4 de cet article, appartient à la classe des conséquences inductives raffinées.
4. *relation de conséquence dans les spécifications conditionnelles de type positif/négatif*. Ces spécifications contiennent des formes générales d'équations conditionnelles, qui acceptent des diséquations dans les conditions. Une relation de conséquence adéquate à elles, construite de manière similaire à celle initiale, a été définie dans [Avenhaus et Madlener, 1997]. Suite a son corollaire 3.12, elle appartient aussi à la classe des conséquences inductives raffinées.

Notons que la relation de conséquence déductive n'est pas une conséquence inductive raffinée.

Dans la suite, on va fixer une relation de conséquence inductive raffinée arbitraire autour de laquelle on va développer notre cadre de travail. Par abus de notation, elle est représentée par $\models_{ind^*}$ pour montrer qu'elle appartient à la classe des conséquences inductives raffinées.† Lorsqu'il faut (surtout dans les exemples), on va l'identifier avec une relation de conséquence concrète.

Par la notation $Ax \not\models_{ind^*} \phi$, on présente ϕ comme une formule qui n'est *pas* une conséquence de Ax. Si ϕ est close, alors ϕ est un *contre-exemple*. La notion de conséquence peut s'étendre aux ensembles de formules. Soit Φ un ensemble de formules. Alors, i) $Ax \models_{ind^*} \Phi$ ssi $Ax \models_{ind^*} \phi$ pour toute $\phi \in \Phi$, et ii) $Ax \not\models_{ind^*} \Phi$ ssi il existe $\phi \in \Phi$ tel que $Ax \not\models_{ind^*} \phi$.

Toute relation de conséquence arbitraire $\models_\lambda$ satisfait les propriétés suivantes :

Proposition 2.2.1 *Les propriétés suivantes (P1), (P2) et (P3) sont satisfaites.*

(P1). si $\phi \in \Phi$ *alors* $\Phi \models_\lambda \{\phi\}$ *;*
(P2). si $\Phi \models_\lambda \Gamma$ *et* $\Phi \subseteq \Psi$ *alors* $\Psi \models_\lambda \Gamma$ *;*
(P3). si $\Phi \models_\lambda \Gamma$ *et* $\Phi \cup \Gamma \models_\lambda \Gamma'$ *alors* $\Phi \models_\lambda \Gamma'$*,*

pour toute formule ϕ *et ensembles de formules* Φ*,* Ψ*,* Γ *et* Γ'*.*

† On va appeler une conséquence inductive raffinée tout simplement conséquence lorsqu'il n'y a pas de risques d'ambiguïtes.

Preuve Soit Φ un ensemble de formules et ϕ une formule. On va vérifier les propriétés (P1), (P2) et (P3) une après l'autre :

- (P1). Si $\phi \in \Phi$ alors tout modèle de Φ est aussi un modèle de ϕ. Donc, on a $\Phi \models_\lambda \{\phi\}$.
- (P2). Si $\Phi \subseteq \Psi$, alors tout modèle de Ψ est aussi un modèle de Φ. Donc, si $\Phi \models_\lambda \Gamma$ alors $\Psi \models_\lambda \Gamma$.
- (P3). La relation $\models_\lambda$ est transitive : si tout modèle de Φ est un modèle de Ψ ($\Phi \models_\lambda \Psi$) et tout modèle de Ψ est un modèle de Γ ($\Psi \models_\lambda \Gamma$), alors tout modèle de Φ est un modèle de Γ ($\Phi \models_\lambda \Gamma$).
  Par (P1), on peut déduire que $\Phi \models_\lambda \Phi$. De plus, si $\Phi \models_\lambda \Gamma$, alors $\Phi \models_\lambda \Phi \cup \Gamma$. Avec l'hypothèse $\Phi \cup \Gamma \models_\lambda \Gamma'$ et la propriété de transitivité de $\models_\lambda$, on conclut que $\Phi \models_\lambda \Gamma'$.

Fin de preuve

Ces propriétés sont aussi satisfaites par $\models_{ind^*}$. A partir d'elles, on peut déduire d'autres propriétés utiles. Par exemple, étant donnée une formule ϕ, s'il existe une formule ψ telle que ψ est un contre-exemple et que ψ est une conséquence de $\{\phi\} \cup Ax$, alors ϕ est aussi un contre-exemple. Cette propriété est généralisée pour des ensembles de formules par la proposition 2.2.2.

Proposition 2.2.2 *Soient Φ, Γ et Γ' trois ensembles de formules tels que $\Phi \not\models_{ind^*} \Gamma'$ et $\Phi \cup \Gamma \models_{ind^*} \Gamma'$. Alors $\Phi \not\models_{ind^*} \Gamma$.*

Preuve Il suffit de contraposer $\Phi \models_{ind^*} \Gamma'$ et $\Phi \models_{ind^*} \Gamma$ dans la propriété (P3).

Fin de preuve

2.3 Le concept d'ensemble couvrant contextuel

Dans la définition 2.2.1, on doit vérifier un nombre, potentiellement infini, d'instances closes d'une formule ϕ pour établir la relation de conséquence entre les axiomes et la formule. Pour cette raison, une solution est de manipuler des descriptions finies de l'ensemble de toutes ces instances closes. Plus précisément, on va considérer des ensembles finis de formules qui les *couvrent*, appelés des *ensembles couvrants* de ϕ.

Dans l'exemple suivant, on montre une façon de construire dans le modèle initial des ensembles couvrants pour des clauses équationnelles.

Exemple 2.3.1 (ensemble couvrant d'une clause) *Soit $<_t$ un ordre décroissant sur les termes, Ax un ensemble d'axiomes, R un système de réécriture conditionnelle obtenu par l'orientation des axiomes de Ax et $\rightarrow_R$ la relation de réécriture induite qui est supposée compatible avec $<_t$.*

On dit qu'une substitution est R-irréductible *si les termes de son codomaine sont des termes R-irréductibles. Un terme t est* inductivement R-irréductible *(resp.* inductivement R-réductible*) si pour toute substitution close R-irréductible γ, $t\gamma$ est R-irréductible (resp. R-réductible).*

La décidabilité de la relation de réductibilité inductive pour un ensemble fini de règles inconditionnelles est décidable [Plaisted, 1985; Comon, 1989; Comon et Jacquemard, 1997], tandis que pour des systèmes de réécriture conditionnelle elle est seulement semi-décidable [Kaplan et Choquer, 1986].

Un ensemble couvrant pour le système de réécriture conditionnelle *R, dénoté par $CS(R)$, est un ensemble fini de termes R-irréductibles tels que, pour tout terme clos R-irréductible s, il*

existe un terme $t \in CS(R)$ et une substitution close σ telle que $Ax \models_{ded} t\sigma = s$. Notons que dans le modèle initial, tout terme clos R-irréductible t est considéré comme le représentant de la classe de congruence sur $\mathcal{T}(\mathsf{F})$ contenant les termes clos dont t est leur forme normale.

A partir d'un ensemble couvrant pour un système de réécriture conditionnelle, on peut construire un ensemble couvrant pour des clauses. Un ensemble couvrant pour la clause *C est l'ensemble d'instances de C obtenu par l'instanciation d'un sous-ensemble particulier de variables de $Var(C)$ par des termes de $CS(R)$ dont les variables sont renommées par de nouvelles variables. On va dénoter par $CS\Sigma(C)$ l'ensemble de toutes les substitutions possibles pour la clause C ainsi construites, appelées des* substitution couvrantes*. Alors, l'ensemble $\{C\sigma \mid \sigma \in CS\Sigma(C)\}$ est un ensemble couvrant pour la clause C.*

Des définitions générales d'ensembles couvrants se trouvent dans [Bronsard *et al.*, 1996; Naidich, 1996]. Par exemple, celle de [Bronsard *et al.*, 1996] est une généralisation de [Reddy, 1990]. Des variantes sont rencontrées dans les méthodes de récurrence basées sur la complétion [Bachmair, 1988; Zhang *et al.*, 1988; Kounalis et Rusinowitch, 1990; Kapur *et al.*, 1991]. Dans la suite, nous proposons une définition d'ensemble couvrant plus générale, basée sur la notion de *contexte*. Le contexte est un ensemble de formules, autres que les axiomes, qui peuvent être utilisées pendant le processus de déduction sans affecter sa correction. Introduisons les notations suivantes :

Notation 2.3.1 ($\Phi_{<\psi}$, $\Phi_{\sim\psi}$, $\Phi_{\leq\psi}$) *Soient $\leq$ un pré-ordre sur les formules, Φ un ensemble de formules et ψ une formule, on définit les notations suivantes :*

$$\Phi_{\#\psi} = \{\phi\tau \mid \tau \text{ est une substitution et } \phi \in \Phi \text{ tels que } \phi\tau\#\psi\},$$

où $\#$ est un des symboles de $\{<\ ,\ \sim\ ,\ \leq\}$.

Lemme 2.3.1 *Soient $\leq$ un pré-ordre fortement stable par substitution, ϕ une formule close et Φ un ensemble de formules. Alors pour toute formule $\psi \in \Phi_{\#\phi}$ et toute substitution close τ, on a $\psi\tau\#\phi$, où $\#$ est un des symboles de $\{<\ ,\ \sim\ ,\ \leq\}$.*

Preuve On observe que la relation $\#$ est stable par substitution. Soit ψ une formule arbitraire de $\Phi_{\#\phi}$. Alors $\psi\#\phi$ par la notation 2.3.1. On suppose que τ est une substitution close. Par la propriété de stabilité de $\#$, on a $\psi\tau\#\phi\tau$. D'autre part, $\phi\tau = \phi$, car ϕ est une formule close. Donc $\psi\tau\#\phi$.

Fin de preuve

Notons que si Φ est l'ensemble vide, alors $\Phi_{\#\phi}$ l'est aussi, pour toute formule ϕ et symbole $\#$ dans $\{<\ ,\ \sim\ ,\ \leq\}$.

Maintenant, on introduit le concepte clé d'*ensemble couvrant contextuel*.

Définition 2.3.1 (contexte et ensemble couvrant contextuel) *Le* contexte C *est une paire d'ensembles de formules (C^1, C^2).*

Soit Ax un ensemble d'axiomes, $\leq$ un pré-ordre bien fondé sur les formules et Φ, Ψ deux ensembles de formules. On dit que Ψ couvre contextuellement *Φ dans le contexte C et on écrit $\Phi \sqsubset_{\mathsf{C}}^{Ax} \Psi$ ssi $Ax \cup \mathsf{C}^1_{\leq\phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \cup \Psi_{\leq\phi\tau} \models_{ind^*} \phi\tau$, pour toute formule $\phi \in \Phi$ et toute substitution close τ. Si $\Phi = \{\phi\}$, on appelle Ψ un* ensemble couvrant contextuel *de ϕ.*

Voici quelques cas particuliers d'ensembles couvrants contextuels obtenus à partir de la définition 2.3.1 :

- l'*ensemble couvrant*, si $\mathsf{C}^1 = \mathsf{C}^2 = \emptyset$,
- l'*ensemble couvrant contextuel strict* si $\Psi_{\leq\phi\tau}$ est remplacé par $\Psi_{<\phi\tau}$, et
- l'*ensemble couvrant contextuel vide* si $\Psi = \emptyset$.

Ces définitions ne s'excluent pas mutuellement. Par exemple, tout ensemble couvrant contextuel vide est aussi strict.

Exemple 2.3.2 (suite de l'exemple 2.3.1) *On va montrer que l'ensemble couvrant* $\Psi = \{C\sigma \mid \sigma \in CS\Sigma(C)\}$ *de la clause* C *est une instance de l'ensemble couvrant considéré comme cas particulier de l'ensemble couvrant contextuel de la définition 2.3.1.*

Pour cela, on prend une substitution close arbitraire $\tau = \{x_i \mapsto t_i \mid x_i \in Var(C)$ *et* t_i *un terme clos*$\}$*. On va montrer par construction qu'il existe une instance* $C\sigma \in \Psi$ *de* C *et une substitution close* τ' *telles que* $Ax \models_{ini} C\tau = C\sigma\tau'$*.*

Soit $\overline{t_i}$ *un terme clos qui représente la R-forme normale de* t_i*. Notons que* $Ax \models_{ini} t_i = \overline{t_i}$ *car* $Ax \models_{ded} t_i = \overline{t_i}$*, puisque* $\overline{t_i}$ *est obtenu de* t_i *par une succession de remplacements d'égaux par égaux. Comme* $\overline{t_i}$ *est un terme R-irréductible, il existe un terme* $s_i \in CS(R)$ *(dont les variables sont fraîches) et une substitution close* θ_i *tels que* $Ax \models_{ini} \overline{t_i} = s_i\theta_i$*. Donc, on a aussi* $Ax \models_{ini} t_i = s_i\theta_i$*. Alors, on construit* $\sigma = \cup_i\{x_i \mapsto s_i\}$ *et* $\tau' = \cup_i\theta_i$*.*

Finalement, il résulte que pour toute substitution close τ *il existe une clause* $C' \in \Psi$ *et une substitution close* τ' *telles que* $Ax \models_{ini} C\tau = C'\tau'$*. Par conséquent,* $Ax \cup \{C'\tau'\}_{\preceq_c C\tau} \models_{ini} C\tau$*.*

Comme les ensembles couvrants, les ensembles couvrants contextuels servent à engendrer des schémas de récurrence noethérienne, c'est-à-dire des schémas de récurrence basés sur un ordre bien fondé, utilisés par les prouveurs basés sur la récurrence explicite [Padawitz, 1992].

Proposition 2.3.1 (récurrence noethérienne avec ensembles couvrants contextuels) *Etant donnés un pré-ordre* $\leq$ *bien fondé sur des formules qui est fortement stable par substitution et un contexte* $\mathsf{C} = (\mathsf{C}^1, \mathsf{C}^2)$ *qui satisfait* $Ax \models_{ind^*} \mathsf{C}^1 \cup \mathsf{C}^2$*, on a* $Ax \models_{ind^*} \phi$ *s'il existe un ensemble de formules* $\cup_i\{\psi_i\}$ *tel que*

1. $\{\phi\} \sqsubset_{\mathsf{C}}^{Ax} \cup_i\{\psi_i\}$*, et*
2. *pour chaque formule* ψ_i*, on a* $Ax \cup \{\phi\}_{<\psi_i} \models_{ind^*} \psi_i$*.*

Preuve La preuve est faite par réfutation. On suppose que $\phi\sigma$ est un contre-exemple. Comme $\leq$ est bien fondé, il existe une instance close *minimale* $\phi\theta$ qui est un contre-exemple.

A partir de la première hypothèse, on déduit que $Ax \cup \mathsf{C}^1_{\leq\phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \cup (\cup_i\{\psi_i\})_{\leq\phi\tau} \models_{ind^*} \phi\tau$, pour toute substitution close τ et, en particulier, pour θ. Comme $Ax \models_{ind^*} \mathsf{C}^1 \cup \mathsf{C}^2$, par la proposition 2.2.2 on déduit que $Ax \not\models_{ind^*} (\cup_i\{\psi_i\})_{\leq\phi\theta}$, c'est-à-dire qu'il existe j et une substitution close η telle que $\psi_j\eta$ qui est un contre-exemple inférieur ou équivalent à $\phi\theta$. A partir de la deuxième hypothèse et la proposition 2.2.2, on a $Ax \not\models_{ind^*} \{\phi\}_{<\psi_j}$. Par conséquent, il existe un contre-exemple $\phi' \in \{\phi\}_{<\psi_j}$ tel que $\phi' < \psi_j$. D'autre part, on a $\phi' < \psi_j\eta$ grâce à $\phi' < \psi_j$, $\phi'\eta = \phi'$ et à la propriété de stabilité de $<$. En utilisant la transitivité de $\leq$, il résulte que ce contre-exemple est plus petit que $\phi\theta$. Ceci contredit la minimalité de $\phi\theta$.

Fin de preuve

Soient $\mathsf{C}_1 = (\mathsf{C}^1_1, \mathsf{C}^2_1)$ et $\mathsf{C}_2 = (\mathsf{C}^1_2, \mathsf{C}^2_2)$ deux contextes associés à un ensemble couvrant contextuel d'une formule ϕ. On dit que C_1 est *plus petit* que C_2 si $(\mathsf{C}^1_{1\leq\phi} \cup \mathsf{C}^2_{1<\phi}) \subset (\mathsf{C}^1_{2\leq\phi} \cup \mathsf{C}^2_{2<\phi})$.

2.3.1 Propriétés des ensembles couvrants contextuels

Dans la suite, on présente quelques propriétés compositionnelles des ensembles couvrants contextuels qui sont utiles en particulier à la construction de nouveaux ensembles couvrants contextuels.

Dans un premier temps, on va montrer que la relation de « couverture contextuelle » entre deux ensembles de formules est préservée par l'augmentation du contexte.

Proposition 2.3.2 *Soient Φ et Ψ deux ensembles arbitraires de formules et $\mathsf{C}_1 = (\mathsf{C}_1^1, \mathsf{C}_1^2)$ et $\mathsf{C}_2 = (\mathsf{C}_2^1, \mathsf{C}_2^2)$ deux contextes pour Φ et Ψ, respectivement. Si $\mathsf{C}_1^1 \subseteq \mathsf{C}_2^1$, $\mathsf{C}_1^2 \subseteq \mathsf{C}_2^2$ et $\Phi \sqsubset_{\mathsf{C}_1}^{Ax} \Psi$ alors $\Phi \sqsubset_{\mathsf{C}_2}^{Ax} \Psi$.*

Preuve Par la définition 2.3.1 et les hypothèses, $Ax \cup \mathsf{C}_{1\leq\phi\tau}^1 \cup \mathsf{C}_{1<\phi\tau}^2 \cup \Psi_{\leq\phi\tau} \models_{ind^*} \phi\tau$, pour toute substitution close τ et toute formule $\phi \in \Phi$. Comme $\mathsf{C}_1^1 \subseteq \mathsf{C}_2^1$ et $\mathsf{C}_1^2 \subseteq \mathsf{C}_2^2$, on obtient $\mathsf{C}_{1\leq\phi\tau}^1 \subseteq \mathsf{C}_{2\leq\phi\tau}^1$ et $\mathsf{C}_{1<\phi\tau}^2 \subseteq \mathsf{C}_{2<\phi\tau}^2$. Par la propriété (P2) de la section 2.2, il résulte que $\Phi \sqsubset_{\mathsf{C}_2}^{Ax} \Psi$.

Fin de preuve

La relation de « couverture contextuelle » pour un contexte arbitraire C est aussi un pré-ordre parce que

- (**réflexivité**) $\Phi \sqsubset_{\mathsf{C}}^{Ax} \Phi$, pour tout ensemble de formules Φ, et
- (**transitivité**) pour tous ensembles de formules Φ, Ψ et Γ, si $\Phi \sqsubset_{\mathsf{C}}^{Ax} \Psi$ et $\Psi \sqsubset_{\mathsf{C}}^{Ax} \Gamma$, alors $\Phi \sqsubset_{\mathsf{C}}^{Ax} \Gamma$.

On suppose que $\mathsf{C} = (\mathsf{C}^1, \mathsf{C}^2)$. Conformément à la définition de l'ensemble couvrant contextuel, la propriété de réflexivité peut se reformuler comme $Ax \cup \mathsf{C}_{\leq\phi\tau}^1 \cup \mathsf{C}_{<\phi\tau}^2 \cup \Phi_{\leq\phi\tau} \models_{ind^*} \phi\tau$, pour toute substitution close τ et formule $\phi \in \Phi$. Comme $\phi\tau \in \Phi_{\leq\phi\tau}$, la propriété de réflexivité est satisfaite, par la propriété (P1).

La propriété de transitivité peut se généraliser à des relations de « couverture contextuelle » définies pour des contextes différents.

Proposition 2.3.3 *On suppose que $\leq$ est un pré-ordre bien fondé et fortement stable et $\mathsf{C}_1 = (\mathsf{C}_1^1, \mathsf{C}_1^2)$, $\mathsf{C}_2 = (\mathsf{C}_2^1, \mathsf{C}_2^2)$ deux contextes arbitraires. Pour tout ensemble de formules Φ, Ψ, Γ, si $\Phi \sqsubset_{\mathsf{C}_1}^{Ax} \Psi$ et $\Psi \sqsubset_{\mathsf{C}_2}^{Ax} \Gamma$, alors $\Phi \sqsubset_{\mathsf{C}}^{Ax} \Gamma$, où $\mathsf{C} = (\mathsf{C}_1^1 \cup \mathsf{C}_2^1, \mathsf{C}_1^2 \cup \mathsf{C}_2^2)$.*

Preuve Conformément à la définition 2.3.1, les hypothèses sont respectivement $Ax \cup \mathsf{C}_{1\leq\phi\tau}^1 \cup \mathsf{C}_{1<\phi\tau}^2 \cup \Psi_{\leq\phi\tau} \models_{ind^*} \phi\tau$ et $Ax \cup \mathsf{C}_{2\leq\psi\sigma}^1 \cup \mathsf{C}_{2<\psi\sigma}^2 \cup \Gamma_{\leq\psi\sigma} \models_{ind^*} \psi\sigma$, pour toutes substitutions closes τ, σ et formules $\phi \in \Phi$, $\psi \in \Psi$.

On suppose que ϕ est une formule de Φ et η est une substitution close arbitraire. Il faut prouver la conjecture $Ax \cup \mathsf{C}_{1\leq\phi\eta}^1 \cup \mathsf{C}_{1<\phi\eta}^2 \cup \mathsf{C}_{2\leq\phi\eta}^1 \cup \mathsf{C}_{2<\phi\eta}^2 \cup \Gamma_{\leq\phi\eta} \models_{ind^*} \phi\eta$. Par la première hypothèse, $Ax \cup \mathsf{C}_{1\leq\phi\eta}^1 \cup \mathsf{C}_{1<\phi\eta}^2 \cup \Psi_{\leq\phi\eta} \models_{ind^*} \phi\eta$. Si $Ax \cup \mathsf{C}_{1\leq\phi\eta}^1 \cup \mathsf{C}_{1<\phi\eta}^2 \models_{ind^*} \phi\eta$, alors la conjecture est satisfaite.

Sinon, on déduit que $\Psi_{\leq\phi\eta} \models_{ind^*} \phi\eta$. On suppose que ψ est une formule arbitraire de $\Psi_{\leq\phi\eta}$ pour laquelle $\psi \leq \phi\eta$. Alors, par la seconde hypothèse, on a $Ax \cup \mathsf{C}_{2\leq\psi\mu}^1 \cup \mathsf{C}_{2<\psi\mu}^2 \cup \Gamma_{\leq\psi\mu} \models_{ind^*} \psi\mu$, pour toute substitution close μ. Les formules de $\Gamma_{\leq\psi\mu}$ se trouvent aussi dans $\Gamma_{\leq\phi\eta}$ parce que $\psi\mu \leq \phi\eta$ (grâce à $\psi \leq \phi\eta$, à $\phi\eta\mu = \phi\eta$ et à la propriété de stabilité forte de $\leq$). Pour des raisons similaires, les formules de $\mathsf{C}_{2\leq\psi\mu}^1$ et $\mathsf{C}_{2<\psi\mu}^2$ sont incluses dans $\mathsf{C}_{2\leq\phi\eta}^1$ et $\mathsf{C}_{2<\phi\eta}^2$, respectivement.

En utilisant la propriété (P2), on peut déduire que $Ax \cup \mathsf{C}^1_{2\leq\phi\eta} \cup \mathsf{C}^2_{2<\phi\eta} \cup \Gamma_{\leq\phi\eta} \models_{ind^*} \Psi_{\leq\phi\eta}$. Par conséquent, $\phi\eta$ est aussi leur conséquence.

Fin de preuve

Par la propriété de transitivité de la relation « couverture contextuelle », pour tous les ensembles de formules Φ_i et Φ_j d'une chaîne arbitraire $\Phi_1 \sqsubset^{Ax}_{\mathsf{C}_1} \ldots \sqsubset^{Ax}_{\mathsf{C}_{n-1}} \Phi_{n-1} \sqsubset^{Ax}_{\mathsf{C}_n} \Phi_n$, il existe un contexte C tel que $\Phi_i \sqsubset^{Ax}_{\mathsf{C}} \Phi_j$ si $i \leq j$. La proposition suivante montre quels sont les rapports de « couverture contextuelle » entre différents ensembles de formules d'une chaîne de relations de « couverture contextuelle » lorsqu'une de ces relations devient stricte.

Proposition 2.3.4 *On suppose l'existence d'une chaîne de relations de « couverture contextuelle » $\Phi_1 \sqsubset^{Ax}_{\mathsf{C}_1} \ldots \sqsubset^{Ax}_{\mathsf{C}_{n-1}} \Phi_{n-1} \sqsubset^{Ax}_{\mathsf{C}_n} \Phi_n$ entre les ensembles de formules $\Phi_1 \ldots \Phi_n$. S'il existe $i \in [1..n-1]$ tel que Φ_{i+1} couvre contextuellement et strictement Φ_i, alors tout Φ_j couvre contextuellement et strictement Φ_k, où $j \in [i+1..n]$ et $k \in [1..i]$.*

Preuve Par la propriété de transitivité de $\leq$ et de $<$ et par la proposition 2.3.3.

Fin de preuve

Des ensembles de formules qui couvrent contextuellement un ensemble donné de formules Φ peuvent être construits à partir d'autres ensembles couvrants contextuels de Φ. Par exemple, l'union de deux ensembles couvrants contextuels de chaque formule de Φ couvre contextuellement Φ, selon la proposition 2.3.5. Un cas particulier à considérer est lorsque les ensembles couvrants contextuels non-vides de chaque formule de Φ sont stricts. Dans ce cas, leur union couvre contextuellement et strictement Φ.

Proposition 2.3.5 *Soient C un contexte, $\Phi = \{\phi_1, \ldots, \phi_n\}$ un ensemble de formules et Ψ_i un ensemble couvrant contextuel (stricte) de ϕ_i dans C, pour chaque $i \in [1..n]$. Alors $\cup_i \Psi_i$ couvre contextuellement (et strictement) Φ dans C.*

Preuve La preuve se construit à partir de l'observation que $\cup_i \Psi_i$ est un ensemble couvrant contextuel (stricte) de ϕ_j dans le contexte C, pour toute $\phi_j \in \Phi$.

Fin de preuve

De nouveaux ensembles couvrants contextuels peuvent être engendrés par l'application des propositions 2.3.3 et 2.3.5.

Exemple 2.3.3 *Soit une spécification conditionnelle contenant les symboles de fonction* 0, *s,* True, False *et aussi un ensemble d'équations conditionnelles Ax qui définissent les symboles de fonction* P : `nat` × `nat` → `bool` *et* $\leq$: `nat` × `nat` → `bool` :

$$0 \leq x = \text{True} \tag{2.1}$$
$$s(x) \leq 0 = \text{False} \tag{2.2}$$
$$s(x) \leq s(y) = x \leq y \tag{2.3}$$
$$P(0,\, x) = \text{True} \tag{2.4}$$
$$s(x) \leq y = \text{True} \Rightarrow P(s(x),\, y) = \text{False} \tag{2.5}$$

Soit R le système de réécriture conditionnelle obtenu par l'orientation des équations conditionnelles de Ax de gauche vers la droite. En outre, on suppose qu'un ensemble couvrant de R

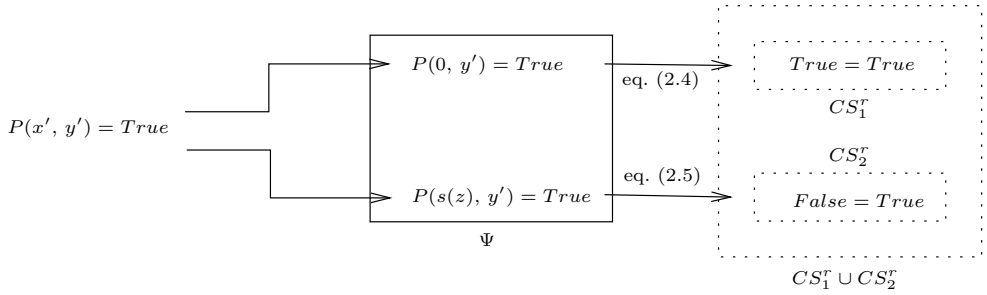

FIG. 2.1 – *Des ensembles couvrants contextuels de* $P(x', y') = \text{True}$

est $\{0, s(z), \text{True}, \text{False}\}$*, comme dans l'exemple 2.3.1. Le pré-ordre sur les termes* $\leq_t$ *est le pré-ordre MPO sur la suivante précédence ascendante sur les symboles de fonction :* True, False, 0, s, $\leq$, P*. Enfin, on suppose que* $\preceq_c$ *est un pré-ordre sur les termes dérivé de* $\leq_t$*, comme dans l'exemple 1.4.8.*

Soit C *une clause* $P(x', y') = \text{True}$*. Si on considère* $CS\Sigma(C) = \{\{x' \mapsto 0\}, \{x' \mapsto s(z)\}\}$ *l'ensemble de substitutions couvrantes, alors* $\Psi = \{P(0, y') = \text{True}, P(s(z), y') = \text{True}\}$ *est un ensemble couvrant de* C*. On peut construire un ensemble couvrant contextuel strict pour tout élément de* Ψ*. Premièrement,* $CS_1^r = \{\text{True} = \text{True}\}$ *est un ensemble couvrant strict de* $P(0, y') = \text{True}$ *car* $(\text{True} = \text{True}) \prec_c (P(0, y') = \text{True})$ *et* $Ax \cup \{\text{True} = \text{True}\} \models_{ini} P(0, y') = \text{True}$*, en utilisant l'équation (2.4). De façon similaire,* $CS_2^r = \{\text{False} = \text{True}\}$ *est un ensemble couvrant contextuel strict de* $P(s(z), y') = \text{True}$ *dans le contexte* $(\emptyset, \{s(z) \leq y' = \text{True}\})$ *car* $(\text{False} = \text{True}) \prec_c (P(s(z), y') = \text{True})$*,* $(s(z) \leq y' = \text{True}) \prec_c (P(s(z), y') = \text{True})$ *et* $Ax \cup \{s(z) \leq y' = \text{True}\} \cup \{\text{False} = \text{True}\} \models_{ini} P(s(z), y') = \text{True}$*, par l'équation (2.5).*

Tout ensemble couvrant strict est aussi un ensemble couvrant contextuel strict, dans tous les contextes. Par conséquent, CS_1^r *peut être considéré comme un ensemble couvrant contextuel strict de* $P(0, y') = \text{True}$ *dans le contexte* $(\emptyset, \{s(z) \leq y' = \text{True}\})$*. Par la proposition 2.3.5,* $CS_1^r \cup CS_2^r$ *est un ensemble couvrant contextuel strict de* Ψ *dans le contexte* $(\emptyset, \{s(z) \leq y' = \text{True}\})$*. Conformément à la proposition 2.3.4,* $CS_1^r \cup CS_2^r$ *est aussi un ensemble couvrant contextuel strict de* C*. Les ensembles couvrants contextuels rencontrés dans cet exemple sont représentés dans la figure 2.1 par des rectangles. Un rectangle en pointillé signifie un ensemble couvrant contextuel strict.*

2.4 Conclusions

Dans ce chapitre, nous avons proposé une généralisation de la notion d'ensemble couvrant utilisé dans les preuves par récurrence, nommée ensemble couvrant contextuel. Nous avons établi ses propriétés de compositionalité qui permettent de construire de nouveaux ensembles couvrant contextuels. Les exemples présentés portent sur le cas où les formules sont des clauses et les axiomes des équations conditionnelles.

3

Systèmes abstraits de déduction basés sur la récurrence implicite

Sommaire

3.1 Introduction

La similarité de la plupart des procédures de preuves basées sur la récurrence implicite suggère l'existence d'un cadre unificateur permettant leur comparaison et leurs extensions modulaires, généralisations et modifications faciles. Quelques systèmes d'inférence ont été proposés, comme I [Bouhoula, 1997], $\mathsf{I}(Ax,\ \geq)$ [Naidich, 1996], la procédure de récurrence implicite de [Bronsard *et al.*, 1996], dénotée dans la suite par B, ou le système d'inférence à cadre « commuté »† [Wirth, 1997]. Par exemple, B généralise la procédure de récurrence hiérarchique de [Reddy, 1990] et les procédures inductives pour des équations conditionnelles de [Kounalis et Rusinowitch, 1990; Bronsard et Reddy, 1991; Bouhoula et Rusinowitch, 1993].

Dans ce chapitre, nous proposons premièrement un nouveau système d'inférence, qui est spécifié uniformément en terme d'ensembles couvrants contextuels. Puis, nous montrons successivement ses propriétés de correction, de correction réfutationnelle et les conditions nécessaires pour garantir sa complétude réfutationnelle. Ensuite, nous faisons une étude comparative avec les procédures abstraites similaires mentionnées ci-dessus. Finalement, nous introduisons le concept de

† en anglais, « switched » frame inference system

AddPremise : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{A}} (E \cup \underbrace{\Phi_1 \cup \ldots \cup \Phi_p}_{\Phi},\ H \cup \{\phi\})$

si
(a) on a un ensemble couvrant contextuel Ψ de ϕ dans le contexte $(H,\ E \cup \Phi)$ **et**
(b) **soit** Ψ est vide. Alors Φ est vide
(b') **soit** $\Psi = \cup_{j=1}^{p}\{\psi_j\}$ **et** Φ_j est un ensemble couvrant contextuel strict de ψ_j dans le contexte $(H,\ E \cup \{\phi\} \cup (\Phi \backslash\backslash \Phi_j))$, pour chaque $j \in [1..p]$

Simplify : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{A}} (E \cup \underbrace{\Phi_1 \cup \ldots \cup \Phi_p}_{\Phi},\ H)$

si
(a) on a un ensemble couvrant contextuel Ψ de ϕ dans le contexte $(E \cup H \cup \Phi,\ \emptyset)$ **et**
(b) **soit** Ψ est vide. Alors Φ est vide
(b') **soit** $\Psi = \cup_{j=1}^{p}\{\psi_j\}$ **et** Φ_j est un ensemble couvrant contextuel de ψ_j dans le contexte $(E \cup H \cup (\Phi \backslash\backslash \Phi_j),\ \{\phi\})$, pour chaque $j \in [1..p]$

Fig. 3.1 – *Le système abstrait d'inférence* A

module de raisonnement, censé de construire les ensembles couvrants contextuels élémentaires, qui ne sont pas obtenus par les opérations de composition présentées dans le chapitre 2. Nous montrons que le schéma d'intégration d'un ensemble de modules de raisonnement arbitraires dans notre système préserve ses propriétés de correction et de complétude.

3.2 Le système abstrait d'inférence A

Soit $\leq$ un pré-ordre sur les formules qui est bien fondé et fortement stable par substitution. Notre système d'inférence abstrait A, présenté dans la figure 3.1, consiste en deux règles d'inférence : AddPremise et Simplify. On suppose qu'une de ces règles a été appliquée sur une conjecture ϕ de l'état de dérivation $(E \cup \{\phi\},\ H)$ [†]. Une conséquence de l'application de cette règle est le remplacement de ϕ par un ensemble couvrant contextuel (potentiellement vide) Φ de ce dernier, qui se construit de la même manière pour les deux règles. Premièrement, on engendre un ensemble couvrant contextuel Ψ de ϕ à l'étape (a). S'il est vide, alors Φ est un ensemble couvrant contextuel vide de ϕ, conformément à l'étape (b). Sinon, Φ est formé de l'union des ensembles couvrants contextuels construits pour chaque élément de Ψ, comme à l'étape (b'). Si la règle est AddPremise, ces ensembles couvrants contextuels sont stricts. Par conséquent, Φ couvre contextuellement et strictement Ψ, selon la proposition 2.3.5. Conformément à la proposition 2.3.2 et 2.3.4, Φ est donc un ensemble couvrant contextuel strict de ϕ. On procède de manière similaire pour la règle Simplify pour montrer que Φ est un ensemble couvrant contextuel de ϕ.

La distinction principale entre les deux règles d'inférence est l'addition de la conjecture traitée à l'ensemble des prémisses lorsqu'on applique AddPremise. Notons que ceci laisse la possibilité à ϕ de participer à des étapes d'inférence ultérieures de la dérivation. En contrepartie, Simplify

† L'ensemble courant de conjectures est supposé non-vide, sinon la dérivation finit avec succès.

a des conditions d'application moins contraignantes que AddPremise : si $\phi\tau$ est une instance close de ϕ, alors i) à l'étape (a) de AddPremise, le contexte ne doit pas inclure des instances de E et Φ équivalentes à $\phi\tau$, et ii) à l'étape (b'), l'ensemble couvrant contextuel Φ_j construit pour tout élément ψ_j de Ψ doit être strict et les instances de $\Phi \backslash\backslash \Phi_j$ et E du contexte doivent être inférieures à $\phi\tau$.

3.3 Propriétés de A

3.3.1 Correction et correction réfutationnelle

Dans la suite, nous montrons quelques propriétés importantes du système d'inférence A, comme la correction. Nous montrons aussi que A est réfutationnellement correct si certaines conditions sont satisfaites.

Définition 3.3.1 (théorème de A, A-preuve) *L'ensemble de propositions E^0 sont des* théorèmes par rapport à l'ensemble d'axiomes *Ax, dénoté par $Ax \vdash^{\mathsf{A}}_{ind^*} E^0$, s'il existe une* A*-dérivation finie $D = (E^0, \emptyset) \vdash^{\mathsf{A}} \cdots \vdash^{\mathsf{A}} (\emptyset, H^n)$, avec $n \geq 1$. La dérivation D est une* A-preuve *de E^0.*

Définition 3.3.2 (correction de A, correction réfutationnelle de A) *Voici quelques propriétés importantes du système d'inférence* A *:*

- ***correction****. Le système d'inférence* A *est* correct *si tout ensemble E de* A*-théorèmes par rapport à un ensemble d'axiomes Ax est leur conséquence inductive. Formellement, si $Ax \vdash^{\mathsf{A}}_{ind^*} E$ alors $Ax \models_{ind^*} E$.*
- ***correction réfutationnelle****. Le système d'inférence* A *est* réfutationnellement correct *si la réfutation d'une conjecture dans une étape d'inférence implique la réfutation des conjectures initiales. Formellement, si $(E, \emptyset) \vdash^{\mathsf{A}} \ldots \vdash^{\mathsf{A}} (E', H)$ et E' contient un contre-exemple, alors $Ax \not\models_{ind^*} E$.*

Par la proposition suivante, on introduit une propriété essentielle pour établir la correction de A.

Proposition 3.3.1 *Soit E^0 un ensemble de conjectures, H^0 un ensemble de prémisses et Ax un ensemble d'axiomes tels qu'il existe une* A*-dérivation commençant par l'état (E^0, H^0) et finissant avec succès. Alors pour toute conjecture $\phi \in E^0$ et pour toute substitution close τ, $Ax \cup H^0_{\leq\phi\tau} \models_{ind^*} \phi\tau$.*

Preuve On va raisonner par contradiction. D'une part, on suppose qu'il y a une formule $\phi_0 \in E^0$, telle qu'il existe une substitution close σ pour laquelle $Ax \cup H^0_{\leq\phi_0\sigma} \not\models_{ind^*} \phi_0\sigma$. D'autre part, on suppose qu'il existe une A-dérivation finie $(E^0, H^0) \vdash^{\mathsf{A}} \cdots \vdash^{\mathsf{A}} (\emptyset, H^m)$, avec $m \geq 1$.

On définit l'ensemble de contre-exemples de $Ax \cup H^0_{\leq\phi_0\sigma}$ inférieurs où équivalents à $\phi_0\sigma$, contenus par toutes les conjectures rencontrées pendant la dérivation :

$$CE = \{\psi\theta \mid \psi \in \bigcup_{i=0}^{m-1} E^i, \theta \text{ est une substitution close et } Ax \cup H^0_{\leq\phi_0\sigma} \not\models_{ind^*} \psi\theta \text{ tel que } \psi\theta \leq \phi_0\sigma\}$$

L'ensemble CE n'est pas vide ; il contient au moins $\phi_0\sigma$. Il y a aussi un élément minimal de CE car $\leq$ est bien fondé. Comme la dérivation se termine par un ensemble vide de conjectures, il existe une *dernière* étape $i \in [0..m-1]$ quand la conjecture ϕ, dont les instances closes contiennent un élément minimal de CE, est traitée. On suppose que l'étape i de la preuve est caractérisée par $E^i = E \cup \{\phi\}$ et $H^i = H$.

On va montrer qu'aucune règle d'inférence ne peut s'appliquer à ϕ. Dans ce cas, ϕ est persistante dans la dérivation, en contredisant l'hypothèse selon laquelle la dérivation se termine par un ensemble vide de conjectures. Soit $\phi\tau$ cet élément minimal de CE. On va faire une analyse par cas, conformément à la règle qu'on va appliquer à ϕ.

1. On suppose premièrement que AddPremise a été appliquée à ϕ et qu'un nouvel ensemble de conjectures Φ est engendré. Alors, il existe un ensemble couvrant contextuel Ψ de ϕ dans le contexte $(H,\ E \cup \Phi)$, conformément à l'étape (a). Puis, on a le choix entre i) l'étape (b) telle que Ψ et Φ soient vides, ou ii) l'étape (b') telle que $\Psi = \cup_{j=1}^{p}\{\psi_j\}$ et Φ_j est un ensemble couvrant contextuel de ψ_j dans le contexte $(H,\ E \cup \{\phi\} \cup (\Phi \setminus\!\setminus \Phi_j))$, pour tout $j \in [1..p]$, et avec $\Phi = \cup_{j=1}^{p}\Phi_j$. L'état de la preuve à l'étape $i+1$ sera caractérisé par $H^{i+1} = H \cup \{\phi\}$ et $E^{i+1} = E \cup \Phi$.
   Comme à l'étape (a) Ψ est un ensemble couvrant contextuel de ϕ dans le contexte $(H,\ E \cup \Phi)$, pour toute substitution close τ, on a $Ax \cup (H_{\leq\phi\tau} \cup (E \cup \Phi)_{<\phi\tau}) \cup \Psi_{\leq\phi\tau} \models_{ind^*} \phi\tau$. On va montrer que $Ax \cup H^0_{\leq\phi_0\sigma} \models_{ind^*} (H_{\leq\phi\tau} \cup (E \cup \Phi)_{<\phi\tau}) \cup \Psi_{\leq\phi\tau}$.

   - $Ax \cup H^0_{\leq\phi_0\sigma} \models_{ind^*} (E \cup \Phi \cup (H \setminus\!\setminus H^0))_{<\phi\tau}$ parce que $(E \cup \Phi \cup (H \setminus\!\setminus H^0)) \in \bigcup_{i=0}^{m-1} E^i$ et $\phi\tau$ est un élément minimal de CE.
   - $Ax \cup H^0_{\leq\phi_0\sigma} \models_{ind^*} H^0_{\leq\phi\tau}$ car $\phi\tau \leq \phi_0\sigma$, résultant $H^0_{\leq\phi\tau} \subseteq H^0_{\leq\phi_0\sigma}$.
   - On va prouver par contradiction que $Ax \cup H^0_{\leq\phi_0\sigma} \models_{ind^*} (H \setminus\!\setminus H^0)_{\sim\phi\tau}$. On suppose que $Ax \cup H^0_{\leq\phi_0\sigma} \not\models_{ind^*} (H \setminus\!\setminus H^0)_{\sim\phi\tau}$. Alors, il existe une prémisse $h^i_\alpha \in (H^i \setminus\!\setminus H^0)$ dont les instances closes contiennent au moins un élément de CE, et celui-ci est équivalent à $\phi\tau$, d'après le lemme 2.3.1.
     On va montrer qu'il y a une contradiction si on applique AddPremise dans une certaine étape précédente de la dérivation. Notons que, pendant la dérivation, les prémisses sont accumulées une après l'autre à l'ensemble H^0. Soit l'étape i' l'étape de la dérivation où la *première* prémisse, n'appartenant pas à H^0 et contenant un élément de CE équivalent à $\phi\tau$, a été accumulée. Cette prémisse est dénotée par h. Une telle étape existe et elle a lieu antérieurement ou à l'étape correspondant à l'addition de h^i_α à l'ensemble de prémisses. Soit $h\delta$ l'élément de CE équivalent à $\phi\tau$. L'étape i' de la dérivation est :

$$(E' \cup \{h\},\ H') \vdash^{\mathsf{A}} (E' \cup \underbrace{\Phi'_1 \cup \ldots \cup \Phi'_p}_{\Phi'},\ H' \cup \{h\})$$

     On va raisonner de manière similaire comme on a fait pour l'étape i de la dérivation. Il existe un ensemble couvrant contextuel $\Psi' = \cup_{j=1}^{p'}\{\psi'_j\}$ de h dans le contexte $(H',\ E' \cup \Phi')$ et, soit i) Ψ' est vide, soit ii) pour chaque $j \in [1..p']$, Φ'_j est un ensemble couvrant contextuel strict de ψ'_j dans le contexte $(H',\ E' \cup \{h\} \cup (\Phi' \setminus\!\setminus \Psi'_j))$. Dans le premier cas, on déduit que $Ax \cup H^0_{\leq\phi_0\sigma} \not\models_{ind^*} H'_{\leq h\delta} \cup (E' \cup \Phi')_{<h\delta}$ à partir de $Ax \cup H^0_{\leq\phi_0\sigma} \not\models_{ind^*} h\delta$ et $Ax \cup H'_{\leq h\delta} \cup (E' \cup \Phi')_{<h\delta} \models_{ind^*} h\delta$. D'autre part, pour les mêmes raisons que précédemment, on a $Ax \cup H^0_{\leq\phi_0\sigma} \models_{ind^*} (H' \cup E' \cup \Phi')_{<h\delta} \cup H^0_{\leq h\delta}$. En outre, $H' \setminus\!\setminus H^0$ ne contient que des prémisses accumulées pendant les étapes qui précédent l'étape i, donc

$Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H' \backslash\backslash H^0)_{\sim h\delta}$. Il résulte que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} H'_{\leq h\delta} \cup (E' \cup \Phi')_{<h\delta}$, donc contradiction.

Dans le deuxième cas, on déduit que $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} (\cup_{j=1}^{p'}\{\psi'_j\})_{\leq h\delta}$ à partir de $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} h\delta$, $Ax \cup H'_{\leq h\delta} \cup (E' \cup \Phi')_{<h\delta} \cup (\cup_{j=1}^{p'}\{\psi'_j\})_{\leq h\delta} \models_{ind^*} h\delta$ et $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} H'_{\leq h\delta} \cup (E' \cup \Phi')_{<h\delta}$. Alors, il existe $j' \in [1..p']$ et une substitution close β tels que $\psi'_{j'}\beta$ est un élément de CE, satisfaisant $\psi'_{j'}\beta \leq h\delta$. Comme $\Phi'_{j'}$ est un ensemble couvrant contextuel strict de $\psi'_{j'}$ dans le contexte $(H', E' \cup \{h\} \cup (\Phi' \backslash\backslash \Phi'_{j'}))$, on a $Ax \cup (H'_{\leq \psi'_{j'}\beta} \cup (E' \cup \{h\} \cup (\Phi' \backslash\backslash \Phi'_{j'}))_{<\psi'_{j'}\beta}) \cup (\Phi'_{j'})_{<\psi'_{j'}\beta} \models_{ind^*} \psi'_{j'}\beta$. De plus, on déduit $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} (H'_{\leq \psi'_{j'}\beta} \cup (E' \cup \{h\} \cup \Phi')_{<\psi'_{j'}\beta})$ à partir de $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} \psi'_{j'}\beta$ et $Ax \cup (H'_{\leq \psi'_{j'}\beta} \cup (E' \cup \{h\} \cup \Phi')_{<\psi'_{j'}\beta}) \models_{ind^*} \psi'_{j'}\beta$. D'autre part, on peut montrer comme précédemment que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (E' \cup \Phi' \cup (H' \backslash\backslash H^0) \cup \{h\})_{<\psi'_{j'}\beta} \cup H^0_{\leq \psi'_{j'}\beta}$ et que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H' \backslash\backslash H^0)_{\sim \psi'_{j'}\beta}$ car $\psi'_{j'}\beta \leq h\delta$ et, par conséquent, toute instance close des formules de $H'_{\sim \psi'_{j'}\beta}$ est inférieure ou équivalente à $h\delta$, selon le lemme 2.3.1. Donc, $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H'_{\leq \psi'_{j'}\beta} \cup (E' \cup \{h\} \cup \Phi')_{<\psi'_{j'}\beta})$. Contradiction.

Dans la suite, on va procéder comme pendant le traitement précédent de l'instance $h\delta$. Supposons que l'étape (b) est franchie, alors Ψ est vide, donc Φ l'est aussi. Alors, $Ax \cup (H_{\leq \phi\tau} \cup E_{<\phi\tau}) \models_{ind^*} \phi\tau$ et il résulte que $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} (H_{\leq \phi\tau} \cup E_{<\phi\tau})$. D'autre part, on obtient comme précédemment que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H_{\leq \phi\tau} \cup E_{<\phi\tau})$. Ceci est une contradiction.

Finalement, on considère l'étape (b'). Soit Ψ l'ensemble $\cup_{j=1}^{p}\{\psi_j\}$. On déduit que $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} \Psi_{\leq \phi\tau}$. Par conséquent, il faut exister un indice $j \in [1..p]$ et une substitution close γ tels que $\psi_j\gamma$ est un élément de CE, satisfaisant $\psi_j\gamma \leq \phi\tau$. Selon les hypothèses, Φ_j est un ensemble couvrant contextuel strict de ψ_j dans le contexte $(H, E \cup \{\phi\} \cup (\Phi \backslash\backslash \Phi_j))$. Donc, $Ax \cup (H_{\leq \psi_j\gamma} \cup (E \cup \{\phi\} \cup (\Phi \backslash\backslash \Phi_j))_{<\psi_j\gamma}) \cup (\Phi_j)_{<\psi_j\gamma} \models_{ind^*} \psi_j\gamma$. D'une part, on peut déduire que $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} (H_{\leq \psi_j\gamma} \cup (E \cup \{\phi\} \cup \Phi)_{<\psi_j\gamma})$. D'autre part, $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (E \cup (H \backslash\backslash H^0) \cup \Phi \cup \{\phi\})_{<\psi_j\gamma} \cup H^0_{\leq \psi_j\gamma}$ et $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H \backslash\backslash H^0)_{\sim \psi_j\gamma}$ car $\psi_j\gamma \leq \phi\tau$: i) soit les instances closes de formules de $(H \backslash\backslash H^0)_{\sim \psi_j\gamma}$ sont inférieures à $\phi\tau$, soit ii) elles sont équivalentes à $\phi\tau$, ce qui n'est pas tout à fait possible parce qu'on arrive à une contradiction, comme précédemment. Il résulte que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H_{\leq \psi_j\gamma} \cup (E \cup \{\phi\} \cup \Phi)_{<\psi_j\gamma})$, donc contradiction.

2. On suppose que la règle SIMPLIFY a été appliquée à ϕ et que le nouvel ensemble de conjectures Φ a été engendré. Alors, il existe un ensemble couvrant contextuel Ψ de ϕ dans le contexte $(E \cup H \cup \Phi, \emptyset)$ et i) soit Ψ et Φ sont vides, soit ii) $\Psi = \cup_{j=1}^{p}\{\psi_j\}$ tel que Φ_j est un ensemble couvrant contextuel de ψ_j dans le contexte $(E \cup H \cup (\Phi \backslash\backslash \Phi_j), \{\phi\})$, pour tout $j \in [1..p]$, et $\Phi = \cup_{j=1}^{p}\Phi_j$. La preuve à l'étape $i+1$ est caractérisée par $H^{i+1} = H$ et $E^{i+1} = E \cup \Phi$. Comme Ψ est un ensemble couvrant contextuel de ϕ dans le contexte $(E \cup H \cup \Phi, \emptyset)$, pour toutes les substitutions closes τ, on a $Ax \cup (E \cup H \cup \Phi)_{\leq \phi\tau} \cup \Psi_{\leq \phi\tau} \models_{ind^*} \phi\tau$. Comme dans le cas précédent qui analyse ADDPREMISE, on peut prouver que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (E \cup (H \backslash\backslash H^0) \cup \Phi)_{<\phi\tau} \cup H^0_{\leq \phi\tau}$ et $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H \backslash\backslash H^0)_{\sim \phi\tau}$. On a aussi $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (E \cup \Phi)_{\sim \phi\tau}$ car on a supposé que ϕ est la dernière conjecture de la dérivation contenant un élément minimal de CE.

Dans le premier cas, si Ψ est vide alors Φ l'est aussi et, par conséquent, $Ax \cup (E \cup H)_{\leq \phi\tau} \models_{ind^*} \phi\tau$. A partir de $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} \phi\tau$, on conclut que $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} (E \cup H)_{\leq \phi\tau}$, donc contradiction.

Si $\Psi = \cup_{j=1}^{p}\{\psi_j\}$, on obtient $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} \Psi_{\leq \phi\tau}$. Par conséquent, il faut exister un indice

$j \in [1..p]$ et une substitution γ tels que $\psi_j\gamma$ est un élément de CE, satisfaisant $\psi_j\gamma \leq \phi\tau$. Conformément aux hypothèses, Φ_j est un ensemble couvrant contextuel de ψ_j dans le contexte $(E \cup H \cup (\Phi \setminus\setminus \Phi_j), \{\phi\})$. Donc, $Ax \cup (E \cup H \cup \Phi)_{\leq \psi_j\gamma} \cup \{\phi\}_{<\psi_j\gamma} \models_{ind^*} \psi_j\gamma$, en déduisant $Ax \cup H^0_{\leq \phi_0\sigma} \not\models_{ind^*} (E \cup H \cup \Phi)_{\leq \psi_j\gamma} \cup \{\phi\}_{<\psi_j\gamma}$. D'autre part, on peut prouver comme précédemment, que $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (E \cup (H \setminus\setminus H^0) \cup \Phi \cup \{\phi\})_{<\psi_j\gamma} \cup H^0_{\leq \psi_j\gamma}$, $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (H \setminus\setminus H^0)_{\sim\psi_j\gamma}$ et $Ax \cup H^0_{\leq \phi_0\sigma} \models_{ind^*} (E \cup \Phi)_{\sim\psi_j\gamma}$. Ceci est une contradiction.

Fin de preuve

Théorème 3.3.1 (correction de A) *Soit E^0 un ensemble de conjectures et Ax un ensemble d'axiomes tels que $Ax \vdash^{\mathsf{A}}_{ind^*} E^0$. Alors $Ax \models_{ind^*} E^0$.*

Preuve Dans la proposition 3.3.1, on considère l'ensemble H^0 vide.

Fin de preuve

D'un autre point de vue, à chaque preuve, on construit des ensembles couvrants contextuels vides aux conjectures initiales.

Lemme 3.3.1 *Si $Ax \vdash^{\mathsf{A}}_{ind^*} E^0$, alors il existe un ensemble couvrant contextuel vide qui couvre contextuellement E^0.*

Preuve Par les propriétés de composition d'ensembles couvrants contextuels 2.3.3 et 2.3.5, chaque A-règle d'inférence construit un ensemble couvrant contextuel de la conjecture traitée. Ainsi, l'ensemble des conjectures de l'état final couvre contextuellement E^0. De plus, cet ensemble est vide car $Ax \vdash^{\mathsf{A}}_{ind^*} E^0$.

Fin de preuve

Maintenant, on va analyser la propriété de correction réfutationnelle du système A. Le système d'inférence A est réfutationnellement correct si toute application de ses règles d'inférence préserve la validité des prémisses et des conclusions.

Théorème 3.3.2 (correction réfutationnelle de A) *Le système* A *est* réfutationnellement correct *si on a $Ax \models_{ind^*} E^{i+1} \cup H^{i+1}$ lorsque $Ax \models_{ind^*} E^i \cup H^i$, pour toute étape i de toute* A*-dérivation $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$.*

Preuve Par la propriété de transitivité de la relation $\models_{ind^*}$ dans une A-dérivation arbitraire $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$, il résulte que si $Ax \models_{ind^*} E^0$, alors $Ax \models_{ind^*} E^i \cup H^i$, pour toute étape $i > 0$. En supposant qu'il existe une étape j telle que $Ax \not\models_{ind^*} E^j$, on déduit $Ax \not\models_{ind^*} E^0$.

Fin de preuve

Afin de certifier la propriété de correction réfutationnelle de A, il est suffisant de vérifier qu'il n'y a pas d'étape d'inférence où on ajoute de nouveaux contre-exemples aux conjectures. On appelle cette condition *le critère de la plus petite couverture*.

Définition 3.3.3 (le critère de la plus petite couverture) *On suppose qu'une* A*-règle de la figure 3.1 a été appliquée à la conjecture ϕ d'un état arbitraire $(E \cup \{\phi\}, H)$ d'une* A*-dérivation telle que l'ensemble de nouvelles conjectures Φ est non-vide. Alors le critère de la plus petite couverture est satisfait par cette règle si, pour chaque conjecture $\psi \in \Phi$ et chaque substitution close τ, il existe une substitution close θ telle que $Ax \cup E \cup H \cup \{\phi\theta\} \models_{ind^*} \psi\tau$.*

DELETE : $(E \cup \{\phi\},\, H) \vdash^{\mathsf{A}} (E,\, H)$
si
(a) il existe un ensemble couvrant contextuel Ψ de ϕ dans le contexte $(E \cup H,\, \emptyset)$ **et**
(b) **soit** Ψ est vide
(b') **soit** $\Psi = \cup_{j=1}^{p}\{\psi_j\}$ **et** Φ_j est un ensemble couvrant contextuel vide de ψ_j dans le contexte $(E \cup H,\, \{\phi\})$, pour tout $j \in [1..p]$

FIG. 3.2 – *La règle d'inférence* DELETE

Théorème 3.3.3 (correction réfutationnelle de A) *Le système d'inférence* A *est réfutationnellement correct si le critère de la plus petite couverture est satisfait par ses règles d'inférence.*

Preuve On suppose donnés une A-dérivation arbitraire et un état $(E^i,\, H^i)$ d'une étape arbitraire i tels que $Ax \models_{ind^*} E^i \cup H^i$. Soit ϕ la conjecture traitée de E^i et Φ les nouvelles conjectures obtenues après l'application d'une A-règle d'inférence satisfaisant le critère de la plus petite couverture. L'état à l'étape $i+1$ est $(E^{i+1},\, H^{i+1})$, tel que $E^{i+1} = (E^i \setminus\!\setminus \{\phi\}) \Cup \Phi$ et H^{i+1} est H^i auquel on ajoute $\{\phi\}$ si la règle appliquée est ADDPREMISE. Si Φ est vide, alors on peut facilement observer, par l'analyse de la définition de chaque A-règle de la figure 3.1, que $(E^{i+1} \cup H^{i+1}) \subseteq (E^i \cup H^i)$. Par conséquent, $Ax \models_{ind^*} E^{i+1} \cup H^{i+1}$.

Sinon, on suppose qu'il existe une formule $\psi \in E^{i+1} \cup H^{i+1}$ qui contient un contre-exemple $\psi\tau$. Comme $Ax \models_{ind^*} E^i \cup H^i$, on a $\psi \in \Phi$. Parce qu'on a supposé que le critère de la plus petite couverture est satisfait, on déduit qu'il y a une substitution close θ telle que $Ax \cup (E^i \setminus\!\setminus \{\phi\}) \cup H^i \cup \{\phi\theta\} \models_{ind^*} \psi\tau$. Par la proposition 2.2.2, on a $Ax \not\models_{ind^*} \phi\theta$. D'autre part, $Ax \models_{ind^*} \phi$ car $\phi \in E^i$. Contradiction.

Fin de preuve

Le système d'inférence A est non-déterministe. Par exemple, ses règles sont applicables si les conditions associées aux étapes (a) et (b) de ADDPREMISE sont satisfaites. Le non-déterministe peut être éliminé par la fixation d'une précédence globale sur les règles. Comme les dérivations finies avec succès se terminent par un ensemble fini de conjectures, une stratégie raisonnable d'application de ces règles sera d'essayer premièrement l'instance de la règle SIMPLIFY qui produit un ensemble couvrant contextuel vide de la conjecture traitée. Cette instance, appelée DELETE, est représentée dans la figure 3.2.

3.3.2 Discussions sur la complétude et la complétude réfutationnelle

Les méthodes de preuves par récurrence sont incomplètes. Cette conclusion est une conséquence du célèbre (premier) théorème d'incomplétude de Gödel, selon lequel pour tout système de preuve cohérente de l'arithmétique, il existe une affirmation arithmétique valide† qui n'est pas prouvable par le système.

Dans la suite de cette section, on va étudier la propriété de complétude réfutationnelle du système A. Pour cela, on suppose qu'il y a un prédicat d'échec, `Fail`(E), pour réfuter un ensemble de conjectures E, tel que si `Fail`(E), alors $Ax \not\models_{ind^*} E$.

† donnée sous la forme d'une formule de premier ordre

Définition 3.3.4 (complétude réfutationnelle de A) *On dit qu'un système d'inférence est* réfutationnellement complet *si tout ensemble de conjectures qui n'est pas une conséquence inductive des axiomes est réfuté. Rapporté au système* A*, si* $Ax \not\models_{ind^*} E$*, alors pour toute dérivation il existe un état* (E', H') *tel que la dérivation a la forme* $(E, \emptyset) \vdash^{\mathsf{A}} \ldots \vdash^{\mathsf{A}} (E', H)$ *et* $\texttt{Fail}(E')$ *est vrai.*

Intuitivement, le système A est réfutationnellement complet si les conditions suivantes sont satisfaites : i) l'existence d'une conjecture contenant un contre-exemple minimal de la dérivation implique l'existence d'une conjecture contenant un contre-exemple minimal auquelle aucune des règles de A n'est pas applicable. ii) les A-dérivations sont équitables.

Définition 3.3.5 (A-dérivation équitable) *Une* A*-dérivation* $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$ *est équitable si soit*

- *la dérivation est finie et elle termine dans l'état* (E^n, H^n) *tel que* $\texttt{Fail}(E^n)$ *ou* $E^n = \emptyset$*, soit*
- *la dérivation est infinie et l'ensemble de conjectures persistantes* $\bigcup_{i\geq 0} \bigcap_{j\geq i} E^j$ *est vide.*

Un système d'inférence est *équitable* s'il construit uniquement des dérivations équitables.

Théorème 3.3.4 (complétude réfutationnelle de A) *On suppose que* A *est équitable et que l'existence d'une conjecture contenant un contre-exemple minimal de la dérivation équitable* $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$ *implique i) l'existence d'un dernier état i dans la dérivation, et ii) l'existence d'une conjecture de* E^i*, contenant un contre-exemple minimal telle que* $\texttt{Fail}(E^i)$ *est vrai. Alors*

1. *si* $Ax \not\models_{ind^*} E^0$*, pour toute* A*-dérivation équitable* $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$ *il existe une étape* $n \geq 0$ *pour laquelle* $\texttt{Fail}(E^n)$ *est vrai, et*
2. A *est réfutationnellement complet.*

Preuve Soient $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$ une A-dérivation équitable telle que $Ax \not\models_{ind^*} E^0$. Donc, il existe une conjecture $\phi_0 \in E^0$ et une instance close $\phi_0\sigma$ de celle-ci. On construit l'ensemble CE de tous les contre-exemples inférieurs ou équivalents à $\phi_0\sigma$ rencontrés pendant la dérivation, comme dans la preuve de la proposition 3.3.1 pour laquelle $H^0 = \emptyset$. L'ensemble CE n'est pas vide ; il contient au moins $\phi_0\sigma$. De plus, il existe aussi un élément minimal de CE, noté par ψ, car $\leq$ est bien fondé. Comme la dérivation est équitable, il existe une étape i lorsqu'une conjecture ϕ contenant une instance close $\phi\tau$ équivalent à ψ est traitée. Par hypothèse, ceci implique l'existence d'un dernier état j de la dérivation qui contient une conjecture contenant un contre-exemple minimal telle que $\texttt{Fail}(E^j)$ est vrai.

On conclut que A est réfutationnellement complet.

Fin de preuve

3.4 Comparaison de A avec d'autres systèmes d'inférence

Dans cette section, nous comparons quelques systèmes d'inférence abstraits (dérivés de B [Bronsard *et al.*, 1996], le système à cadre « commuté » [Wirth, 1997], $\mathsf{I}(Ax,\ \geq)$ [Naidich, 1996] et I [Bouhoula, 1997], tous présentés dans la figure 3.3) par rapport au système A dans le cas où on a un même domaine d'interprétation et le pré-ordre utilisé, $\leq$, est bien fondé et fortement

stable par substitution. On va supposer que le domaine de raisonnement est constitué par des termes clos.

Dans la figure 3.3(a), on présente le système d'inférence B. La figure 3.3(b) illustre une variante du système à cadre « commuté » noté avec S. Sa version originale a été conçue pour manipuler des instances de formules (à la place de formules) afin de permettre des ordres de récurrence plus flexibles. Dans cette approche, une instance close d'une formule ϕ est une paire (ϕ, γ), où γ est une substitution close, et une instance close (ϕ, γ) représente la formule close $\phi\gamma$. Par conséquent, une formule close peut avoir plusieurs représentations comme instance de formule. Comme les ordres utilisés sur de telles paires sont aussi fortement stables par substitution, des systèmes d'inférence abstraits basés sur des instances de formules peuvent être déduits facilement à partir des systèmes présentés dans la figure 3.3, par le remplacement des ordres sur des formules par des ordres sur des instances de formules.

Une variante du $\mathsf{I}(Ax,\ \geq)$, adaptée au raisonnement sur des formules et dénoté par $\mathsf{I}'(Ax,\ \geq)$, est présenté dans la figure 3.3(c). Concernant le système I, il peut se reconstruire à partir du système d'inférence I′, défini dans la figure 3.3(d), en considérant les formules comme étant des clauses et $\leq$ comme le pré-ordre $\preceq_c$ sur des clauses de la définition 1.4.8. On suppose aussi que le critère de la plus petite couverture est satisfait par chaque I′-règle.

Dans les tableaux 3.1 et 3.2, on a représenté les règles d'inférence des systèmes de la figure 3.3[†] comme des instances respectivement de AddPremise et Simplify. Chaque tableau a quatre colonnes. Les deux premières colonnes donnent le nom de la règle et de la procédure abstraite auquelle elle appartient. La colonne suivante indique l'ensemble couvrant contextuel et son contexte maximal admis à l'étape (a) par rapport à la conjecture traitée ϕ. Dans la dernière colonne, on présente le contexte maximal admis à l'étape (b') par rapport à la formule ψ_j. Les endroits où les ensembles couvrants contextuels doivent être stricts sont marqués par « *(strict)* ». Par exemple, pour chaque règle du tableau 3.1, l'ensemble couvrant contextuel construit à l'étape (b') est strict.

		Etape (a)		**Etape (b')**
règle	**système**	$\cup_{j=1}^{p}\psi_j$	**contexte**	**contexte**
Expand	B	$\{\phi\}$	$(\emptyset, \emptyset)$	$(\emptyset, \emptyset)$ *(strict)*
M.S.T.	S	$\{\phi\}$	$(\emptyset, \emptyset)$	$(H, E \cup (\Phi \setminus\!\setminus \Phi_j))$ *(strict)*
Generate	$\mathsf{I}'(Ax,\ \geq)$	$\cup_{j=1}^{p}\psi_j$	$(\emptyset, \emptyset)$	$(\emptyset, E \cup H \cup (\Phi \setminus\!\setminus \Phi_j) \cup \{\phi\})$ *(strict)*
Generate	I′	$\cup_{j=1}^{p}\psi_j$	$(\emptyset, \emptyset)$	$(\emptyset, E \cup H \cup (\Phi \setminus\!\setminus \Phi_j) \cup \{\phi\})$ *(strict)*
AddPremise	A	$\cup_{j=1}^{p}\psi_j$	$(H, E \cup \Phi)$	$(H, E \cup (\Phi \setminus\!\setminus \Phi_j) \cup \{\phi\})$ *(strict)*

Tab. 3.1 – *Des règles qui instancient* AddPremise *de* A

Une simple analyse des ensembles couvrants contextuels et de ses contextes maximaux admis pour chaque règle aboutit à la conclusion que les A-règles, représentées dans la dernière ligne de chaque tableau, sont les plus générales. En comparant AddPremise aux autres règles du tableau 3.1, on observe qu'elle permet, à l'étape (a), des contextes non-vides pour engendrer $\cup_{j=1}^{n}\psi_j$. A l'étape (b'), AddPremise admet i) par rapport aux règles Generate, un contexte incluant des instances de H équivalentes à ψ_j, et ii) par rapport à la règle M.S.T., un contexte incluant des instances de $\{\phi\}$ inférieures à ψ_j.

En analysant le tableau 3.2, si on restreint la A-règle Simplify telle que i) à l'étape (a), $\cup_{j=1}^{n}\psi_j$ est remplacé par $\{\phi\}$ et le contexte est vide, et ii) à l'étape (b'), le contexte ne contient

† sauf la règle Lemma du système B

EXPAND : $(E \cup \{\phi\},\ H) \vdash^{B} (E \cup \Phi,\ H \cup \{\phi\})$
si Φ est un ensemble couvrant strict de ϕ

SIMPLIFY : $(E \cup \{\phi\},\ H) \vdash^{B} (E \cup \Phi,\ H)$
si pour toute substitution close τ,
$Ax \cup (E \cup H \cup \Phi)_{\leq \phi\tau} \models_{ind^*} \phi\tau$

LEMMA : $(E,\ H) \vdash^{B} (E \cup \Phi,\ H)$

(a) Le système B

MEMORIZING SWITCHED TRANSFORMATION (M.S.T.) : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{S}} (E \cup \Phi,\ H \cup \{\phi\})$
si pour toute substitution close τ,
$Ax \cup (E \cup \Phi)_{<\phi\tau} \cup H_{\leq \phi\tau} \models_{ind^*} \phi\tau$

SIMPLE SWITCHED TRANSFORMATION (S.S.T.) : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{S}} (E \cup \Phi,\ H)$
si pour toute substitution close τ,
$Ax \cup (E \cup H \cup \Phi)_{\leq \phi\tau} \models_{ind^*} \phi\tau$

(b) Le système S

GENERATE : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{I}'(Ax,\ \geq)} (E \cup \Phi,\ H \cup \{\phi\})$
si il existe un ensemble couvrant Ψ de ϕ tel que, pour toute substitution close τ et toute $\psi \in \Psi$,
$Ax \cup (E \cup H \cup \{\phi\} \cup \Phi)_{<\psi\tau} \models_{ind^*} \psi\tau$

SIMPLIFY : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{I}'(Ax,\ \geq)} (E \cup \Phi,\ H)$
si pour toute substitution close τ,
$Ax \cup (E \cup H \cup \Phi)_{\leq \phi\tau} \models_{ind^*} \phi\tau$

(c) Le système $\mathsf{I}'(Ax,\ \geq)$

GENERATE : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{I}'} (E \cup \Phi,\ H \cup \{\phi\})$
si il existe un ensemble couvrant Ψ de ϕ tel que pour toute substitution close τ et toute $\psi \in \Psi$,
$Ax \cup (E \cup H \cup \{\phi\} \cup \Phi)_{<\psi\tau} \models_{ind^*} \psi\tau$

SIMPLIFY : $(E \cup \{\phi\},\ H) \vdash^{\mathsf{I}'} (E \cup \Phi,\ H)$
si pour toute substitution close τ,
$Ax \cup (E \cup \Phi)_{<\phi\tau} \cup H_{\leq \phi\tau} \models_{ind^*} \phi\tau$

(d) Le système I'

FIG. 3.3 – *Quelques systèmes d'inférence abstraits*

règle	système	Etape (a) $\cup_{j=1}^{p}\psi_j$	Etape (a) contexte	Etape (b') contexte
SIMPLIFY	B	$\{\phi\}$	$(\emptyset, \emptyset)$	$(E \cup H \cup (\Phi \setminus\!\setminus \Phi_j), \emptyset)$
S.S.T.	S	$\{\phi\}$	$(\emptyset, \emptyset)$	$(E \cup H \cup (\Phi \setminus\!\setminus \Phi_j), \emptyset)$
SIMPLIFY	I′(Ax, $\geq$)	$\{\phi\}$	$(\emptyset, \emptyset)$	$(E \cup H \cup (\Phi \setminus\!\setminus \Phi_j), \emptyset)$
SIMPLIFY	I′	$\{\phi\}$	$(\emptyset, \emptyset)$	$(H, E \cup (\Phi \setminus\!\setminus \Phi_j))$ *(strict)*
SIMPLIFY	A	$\cup_{j=1}^{p}\psi_j$	$(E \cup H \cup \Phi, \emptyset)$	$(E \cup H \cup (\Phi \setminus\!\setminus \Phi_j), \{\phi\})$

TAB. 3.2 – *Des règles qui instancient* SIMPLIFY *de* A

pas des instances de ϕ plus petites que ψ_j, on obtient les définitions des autres règles SIMPLIFY et de S.S.T.. La règle I′-SIMPLIFY constitue une exception, dans la mesure où à l'étape (b') on n'utilise pas des instances de H équivalentes à ϕ et l'ensemble couvrant contextuel est strict.

On conclut dans un premier temps que A est une généralisation du S, I′(Ax, $\geq$) et I′. Le système B instancie aussi A si A est étendu avec la règle LEMMA contenue par B :

$$\text{LEMMA} : (E, H) \vdash (E \cup E', H)$$

Il ressort clairement que la version étendue de A reste correcte, car LEMMA ne traite pas de formules de l'ensemble courant de conjectures et, par conséquent, il ne peut pas éliminer la dernière formule de la dérivation qui contient un élément minimal de CE (voir la preuve de correction de A et de la proposition 3.3.1). Malheureusement, la propriété de correction réfutationnelle est perdue si les instances de E' contiennent de nouveaux contre-exemples tels que le critère de la plus petite couverture n'est plus satisfait.

Dans [Bouhoula, 1997], le système I est formé par trois règles : GENERATE, SIMPLIFY et DELETE, où DELETE est une instance de SIMPLIFY. Donc, A est une généralisation de I.

3.5 Schéma générique pour l'intégration des modules de raisonnement dans A

Les règles d'inférence du système d'inférence abstrait A, présenté dans la section 3.2, sont essentiellement descriptives et spécifient, par le biais des contextes associés aux ensembles couvrants contextuels, *quelle* information peut être utilisée afin d'obtenir des preuves correctes. Dans cette section, on va détailler *comment* les ensembles couvrants contextuels sont mis en œuvre à l'aide de *modules de raisonnement*.

3.5.1 Modules de raisonnement

Habituellement, un module de raisonnement est supposé implémenter une technique de raisonnement particulière qu'on veut rendre accessible au prouveur. Dans notre cadre, il représente l'entité qui permet de créer des ensembles couvrants contextuels élémentaires. Selon la technique de raisonnement, il est possible que la génération des ensembles couvrants contextuels dépende de conditions qui se vérifient par récurrence.

Définition 3.5.1 (modules de raisonnement conditionnels et inconditionnels) *Soient Ax un ensemble d'axiomes, $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ une paire d'ensembles de formules et ϕ une formule. On suppose que $\mathbb{P}(S)$ est l'ensemble des parties d'un ensemble S.*

Un module de raisonnement conditionnel M *est caractérisé par deux fonctions partielles* c_M, $g_M : \mathcal{L} \times (\mathbb{P}(\mathcal{L}) * \mathbb{P}(\mathcal{L})) \to \mathbb{P}(\mathcal{L})$ *nommées respectivement la* fonction de condition *et la* fonction de génération. *Elles prennent à l'entrée une formule et un contexte et retournent un ensemble de formules. On dit que* M *est* applicable *à* ϕ *dans le contexte* Cxt *si pour toute substitution close* τ, $Ax \cup \mathsf{C}^1_{\leq\phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \cup g_M(\phi, Cxt)_{\leq\phi\tau} \models_{ind^*} \phi\tau$ *est satisfaite sous l'hypothèse que* $Ax \cup \mathsf{C}^1_{\leq\phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \models_{ind^*} c_M(\phi, Cxt)\tau$.

Un module de raisonnement inconditionnel M' *n'est caractérisé que par la fonction de génération. Dans ce cas,* M' *est* applicable *à* ϕ *dans le contexte* Cxt *ssi* $\{\phi\} \sqsubset^{Ax}_{Cxt} g_{M'}(\phi, Cxt)$.

Lors de l'application d'un module de raisonnement conditionnel M, on note qu'il suffit de montrer que pour toute substitution close τ on a $Ax \cup \mathsf{C}^1_{\leq\phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \models_{ind^*} c_M(\phi, Cxt)\tau$ pour obtenir $\{\phi\} \sqsubset^{Ax}_{Cxt} g_M(\phi, Cxt)$.

Dans la suite, on va présenter quelques exemples de modules de raisonnement. Dans le premier, on définit un module de raisonnement conditionnel basé sur une opération d'analyse par cas sur des clauses équationnelles, dans le modèle initial.

Exemple 3.5.1 *Soient* $C_1, \ldots, C_n$ *des clauses équationnelles,* $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ *un contexte,* $\leqslant_c$ *l'extension multiensemble de l'ordre* $\preceq_c$ *sur les clauses de la définition 1.4.8, et* C *une clause équationnelle tels que* $C_i \prec_c C$, *pour tout* $i \in [1..n]$. *Les deux fonctions caractérisant le module de raisonnement conditionnel* CA, *sont* $c_{CA}(C, Cxt) = \{\bigvee_{i=1}^n C_i\}$ *et* $g_{CA}(C, Cxt) = \cup_{i=1}^n \{C \vee C_i\}$. *On va montrer que* CA *est applicable à* C *pour obtenir* $\{C\} \sqsubset^{Ax}_{Cxt} g_{CA}(C, Cxt)$.

A partir de la définition de $\preceq_c$, *on peut observer que pour tout* $i \in [1..n]$, *on obtient les faits suivants i)* $C_i \vee C \preceq_c C$, *et ii)* $\{\bigvee_{i=1}^n C_i\} \leqslant_c \{C\}$, *puisque* $C_i \prec_c C$. *Pour toute instance close* $C\tau$, *il résulte* $C_i\tau \vee C\tau \preceq_c C\tau$, *par le premier fait et la propriété de stabilité forte par substitution de* $\preceq_c$. *Selon la définition 3.5.1, on suppose que pour toute substitution close* τ, $Ax \cup \mathsf{C}^1_{\leq C\tau} \cup \mathsf{C}^2_{<C\tau} \models_{ini} \{\bigvee_{i=1}^n C_i\}\tau$. *Donc, il existe* $j \in [1..n]$ *tel que* $Ax \cup \mathsf{C}^1_{\leq C\tau} \cup \mathsf{C}^2_{<C\tau} \models_{ini} C_j\tau$. *Par conséquent,* $Ax \cup \mathsf{C}^1_{\leq C\tau} \cup \mathsf{C}^2_{<C\tau} \cup \{C \vee C_j\}_{\preceq_c C\tau} \models_{ini} C\tau$, *car* $(C \vee C_j)\tau \preceq_c C\tau$. *Donc,* $Ax \cup \mathsf{C}^1_{\leq C\tau} \cup \mathsf{C}^2_{<C\tau} \cup g_{CA}(C, Cxt)_{\preceq_c C\tau} \models_{ini} C\tau$.

Dans l'exemple suivant, on va montrer comment des modules de raisonnement inconditionnels basés sur des techniques de réécriture sont construits pour raisonner sur des clauses équationnelles dans le modèle initial.

Exemple 3.5.2 *Soient* Ax *un ensemble d'équations conditionnelles,* R *un système de réécriture conditionnelle obtenu à partir de* Ax *et* $\preceq_c$ *le pré-ordre sur les clauses de la définition 1.4.8. La relation de réécriture conditionnelle* $\to_R$ *de la définition 1.5.3 peut être étendue sur des clauses comme suit. Une clause* $C'[s[a]_u = t]$, *contenant une équation* $s[a]_u = t$, *est* réécrite conditionnellement *à la position* u *de* s *et marqué par* $C'[s[a]_u = t] \xrightarrow{C}_R C'[s[a']_u = t]$, *si* $a \to_R a'$. *Notons que la relation* $C'[s[a']_u = t] \preceq_c C'[s[a]_u = t]$ *est satisfaite, par la définition de* $\preceq_c$, *et que* $\xrightarrow{C}_R$ *est stable par substitution.*

Soient $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ *une paire d'ensembles de clauses équationnelles, et* C_1, C_2 *deux clauses équationnelles telles que* $C_1 \xrightarrow{C}_{R \cup \mathsf{C}^1_{\preceq_c C_1} \cup \mathsf{C}^2_{\prec_c C_1}} C_2$. *On définit la fonction de génération* g_{CR} *du module de raisonnement inconditionnel* CR *comme* $g_{CR}(C_1, Cxt) = \{C_2\}$. *Puisque* $C_2 \preceq_c C_1$, $\preceq_c$ *est fortement stable par substitution et* $\xrightarrow{C}_R$ *est stable par substitution, pour toute substitution close* τ, *on a* $Ax \cup \mathsf{C}^1_{\leq C_1\tau} \cup \mathsf{C}^2_{<C_1\tau} \cup \{C_2\}_{\preceq_c C_1\tau} \models_{ini} C_1\tau$. *Donc,* $\{C_2\}$ *est un ensemble couvrant contextuel de* C_1 *dans le contexte* Cxt.

En raison de l'incomplétude inhérente des méthodes de preuve par récurrence, pendant une preuve il est parfois nécessaire d'utiliser d'autres techniques de raisonnement sur des domaines de raisonnement particuliers. Un dernier exemple illustre l'utilisation des techniques basées sur l'arithmétique linéaire dans le raisonnement sur des clauses équationnelles.

Exemple 3.5.3 (module de raisonnement basé sur l'arithmétique linéaire) *On va définir le module de raisonnement inconditionnel PA pour l'arithmétique linéaire, présentée dans l'exemple 1.6.1.*

Soient C une clause équationnelle, $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ une paire d'ensembles de clauses équationnelles et $\preceq_c$ le pré-ordre sur les clauses de la définition 1.4.8. Alors $g_{PA}(C, Cxt)$ est un ensemble couvrant contextuel vide si $\models_{PA} \mathcal{TR}(\neg C) \cup \mathcal{TR}(Ax \cup \mathsf{C}^1_{\preceq_c C} \cup \mathsf{C}^2_{\prec_c C})$.

3.5.2 Intégration des modules de raisonnement dans A. Le système $\mathsf{A}(\mathcal{RM})$.

Le système d'inférence $\mathsf{A}(\mathcal{RM})$ est une instanciation de A qu'on définit en termes d'un ensemble de modules de raisonnement $\mathcal{RM}$. A partir de maintenant, on va supposer que $\mathcal{RM}$ contient au moins un module de raisonnement inconditionnel. En outre, on va supposer que le test de conséquence des conditions de tout module de raisonnement conditionnel de $\mathcal{RM}$ est fait récursivement en appelant $\mathsf{A}(\mathcal{RM})$. Dans ce cas, on dit que $\mathsf{A}(\mathcal{RM})$ est un système d'inférence *récursif* et que les modules de raisonnement sont *intégrés* dans le système d'inférence A : d'une part, l'ensemble couvrant contextuel employé au moment de l'application d'une règle d'inférence est engendré par des modules de raisonnement et, d'autre part, le système $\mathsf{A}(\mathcal{RM})$ est employé pour vérifier les conditions des modules de raisonnement conditionnels. Par conséquent, le système d'inférence $\mathsf{A}(\mathcal{RM})$ et les modules de raisonnement conditionnels sont *mutuellement dépendants*.

Définition 3.5.2 (schéma d'intégration des modules de raisonnement conditionnels) *Soient Ax un ensemble d'axiomes, $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ un contexte et $M \in \mathcal{RM}$ un module de raisonnement conditionnel. Alors M est applicable à la formule ϕ dans le contexte Cxt si $\{\phi\} \sqsubset^{Ax}_{Cxt} g_M(\phi, Cxt)$ et $c_M(\phi, Cxt)$ est prouvée par un appel récursif à $\mathsf{A}(\mathcal{RM})$. M est aussi applicable à ϕ dans Cxt si*

- *$c_M(\phi, Cxt) \lessdot\!\lessdot \{\phi\}$ et la $\mathsf{A}(\mathcal{RM})$-dérivation $(c_M(\phi, Cxt), \mathsf{C}^1) \vdash^{\mathsf{A}(\mathcal{RM})} \ldots$ finit avec succès, ou*
- *$c_M(\phi, Cxt) \ll \{\phi\}$ et la $\mathsf{A}(\mathcal{RM})$-dérivation $(c_M(\phi, Cxt), \mathsf{C}^1 \cup \mathsf{C}^2) \vdash^{\mathsf{A}(\mathcal{RM})} \ldots$ finit avec succès.*

Notons que si les conditions d'application du module M sont inférieures ou équivalentes à la conjecture courante, on peut utiliser des éléments du contexte en tant que prémisses lors de leur vérification avec $\mathsf{A}(\mathcal{RM})$.

L'exemple suivant montre la dérivation d'une preuve avec le système $\mathsf{A}(\mathcal{RM})$.

Exemple 3.5.4 *Soit $\mathcal{RM}$ un ensemble contenant les modules de raisonnement $\{CA, CR, PA\}$, définis respectivement dans les exemples 3.5.1, 3.5.2 et 3.5.3. Reprenons la spécification conditionnelle de l'exemple 1.5.1 d'où on peut dériver les deux règles de réécriture*

$$x \leq y = True \Rightarrow min(x, y) \rightarrow x \tag{3.1}$$

$$x \leq y = False \Rightarrow min(x, y) \rightarrow y \tag{3.2}$$

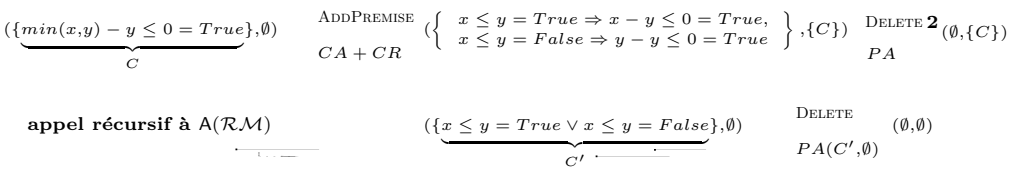

FIG. 3.4 – *Une* $\mathsf{A}(\mathcal{RM})$*-preuve de* $min(x, y) - y \leq 0 = True$

qui définissent la fonction min et qui sont obtenues par l'orientation des équations conditionnelles correspondantes en utilisant l'ordre MPO de la définition 1.4.7 avec la priorité suivante sur les symboles de fonctions

$$\text{True} \prec_{\mathsf{F}} \text{False} \prec_{\mathsf{F}} 0 \prec_{\mathsf{F}} s \prec_{\mathsf{F}} \leq \prec_{\mathsf{F}} min$$

Une $\mathsf{A}(\mathcal{RM})$*-preuve de la conjecture C, représentant l'équation* $min(x, y) - y \leq 0 = True$*, est illustrée dans la figure 3.4.*

La $\mathsf{A}(\mathcal{RM})$*-dérivation commence par l'application à C de la règle* AddPremise *dont les ensembles couvrants contextuels sont mis en œuvre par les modules de raisonnement* CA *et* CR*. Dans un premier temps,* CA *construit l'ensemble couvrant* $\overline{C} = \{x \leq y = True \Rightarrow min(x, y) - y \leq 0 = True,\ x \leq y = False \Rightarrow min(x, y) - y \leq 0 = True\}$ *avec la condition que* $x \leq y = True \vee x \leq y = False$*. Ceci correspond à l'étape (a) de la définition de* AddPremise*, montrée dans la figure 3.1. A l'étape (b'), pour chaque élément de* $\overline{C}$*,* CR *construit un ensemble couvrant strict en réécrivant le terme* $\leq_t$*-maximal* $min(x, y) - y \leq 0$ *de C à son sous-terme* $min(x, y)$ *avec les règles de réécriture conditionnelles (3.1) et (3.2).*

La condition $x \leq y = True \vee x \leq y = False$ *est testée par* $\mathsf{A}(\mathcal{RM})$*. Dans la figure 3.4, la* $\mathsf{A}(\mathcal{RM})$*-preuve due à l'appel récursive à* $\mathsf{A}(\mathcal{RM})$ *est indiquée en pointillé. La dérivation consiste dans une seule étape d'inférence correspondant à l'application de la règle* Delete*. Les ensembles couvrants contextuels sont implémentés avec le module de raisonnement* PA *qui transforme la négation de la clause* $x \leq y = True \vee x \leq y = False$ *dans la conjonction des inégalités linéaires* $-x + y + 1 \leq 0$ *et* $x - y \leq 0$*. Par leur addition, l'inégalité impossible* $1 \leq 0$ *est obtenue. Ceci nous permet de conclure que la clause est une conséquence initiale des axiomes.*

Après l'application de AddPremise*,* $(E^1 = \{x \leq y = True \Rightarrow x - y \leq 0 = True,\ x \leq y = False \Rightarrow y - y \leq 0 = True\},\ H^1 = \{C\})$ *est le nouveau état de la dérivation. Pendant les dernières deux étapes, les conjectures de* E^1 *sont éliminées successivement par l'application de la règle* Delete *comme précédemment. A chaque étape, l'inégalité impossible* $1 \leq 0$ *a été engendrée. Enfin, la dérivation finit par un ensemble vide de conjectures.*

Définition 3.5.3 (dérivation linéaire et arborescente) *Une* $\mathsf{A}(\mathcal{RM})$*-dérivation qui ne contient pas d'appels récursifs à* $\mathsf{A}(\mathcal{RM})$ *est* linéaire*, sinon elle est* arborescente *et est formée d'une ensemble de* $\mathsf{A}(\mathcal{RM})$*-dérivations linéaires stratifiées.*

Par exemple, la dérivation arborescente de l'exemple 3.5.4 est constituée par deux dérivations linéaires.

La dimension *verticale* d'une $\mathsf{A}(\mathcal{RM})$-dérivation arborescente est introduite par les appels récursifs à $\mathsf{A}(\mathcal{RM})$. A chaque appel récursif, une nouvelle $\mathsf{A}(\mathcal{RM})$-dérivation est ajoutée comme « fils » de la dérivation linéaire courante à l'arbre de la preuve globale. Ainsi, la « racine » de la

preuve globale est une dérivation linéaire qu'on va considérer comme *principale* parmi les autres dérivations linéaires, considérées comme *secondaires*.

Une mesure de la dimension verticale d'une $\mathsf{A}(\mathcal{RM})$-dérivation arborescente est sa *profondeur*.

Définition 3.5.4 (profondeurs de M-étapes, $\mathsf{A}(\mathcal{RM})$-étapes et $\mathsf{A}(\mathcal{RM})$-dérivations) *On suppose que les conditions d'un module de raisonnement conditionnel $M \in \mathcal{RM}$ sont vérifiées pendant une M-*étape*. La profondeur d'une M-étape, d'une $\mathsf{A}(\mathcal{RM})$-étape et d'une $\mathsf{A}(\mathcal{RM})$-dérivation sont définies récursivement comme suit.*

Si ϕ est une formule pour laquelle le module de raisonnement M_i engendre un ensemble couvrant contextuel dans le contexte Cxt pendant une étape S d'une dérivation linéaire D, alors la profondeur de la M_i-étape est $depth^m(M_i, S, D) =$

$$\begin{cases} 0 & \text{si } M_i \text{ est un module de raisonnement inconditionnel} \\ 1 + depth^d(D_{c_{M_i}(\phi,\, Cxt)}) & \text{sinon} \end{cases}$$

où $D_{c_{M_i}(\phi,\, Cxt)}$ est la $\mathsf{A}(\mathcal{RM})$-preuve de $c_{M_i}(\phi, Cxt)$.

La profondeur de l'étape d'inférence S d'une dérivation linéaire D est

$$depth^s(S, D) = max(\{depth^m(M_i) \mid i \in [1..p]\}),$$

où $\{M_1, \ldots, M_p\} \subseteq \mathcal{RM}$ est un ensemble de modules de raisonnement qui participent à la génération des ensembles couvrants contextuels impliqués à l'étape S. Enfin, la profondeur de la dérivation D est

$$depth^d(D) = max(\{depth^s(S_i) \mid S_i \text{ est une étape de la dérivation linéaire principale de } D\})$$

Exemple 3.5.5 *Dans la figure 3.4, la profondeur de la $\mathsf{A}(\mathcal{RM})$-preuve est 1.*

Définition 3.5.5 ($\mathsf{A}(\mathcal{RM})$-preuve, $\vdash_{ind*}^{\mathsf{A}(\mathcal{RM})}$) *Etant donnés un ensemble d'axiomes Ax et un ensemble de conjectures E, une $\mathsf{A}(\mathcal{RM})$-dérivation stratifiée est une* $\mathsf{A}(\mathcal{RM})$-preuve *de E, dénotée par $Ax \vdash_{ind*}^{\mathsf{A}(\mathcal{RM})} E$, si*

- *elle est de profondeur finie,*
- *chacune de ses $\mathsf{A}(\mathcal{RM})$-dérivations secondaires sont finies et terminent avec succès, et*
- *la $\mathsf{A}(\mathcal{RM})$-dérivation principale est une preuve.*

Selon la définition 3.5.2, on peut noter que les modules de raisonnement conditionnels impliqués dans la dérivation arborescente ne sont applicables qu'*après* ses conditions soient prouvées. Sous cette hypothèse, la définition 3.5.4 est bien fondée. De plus, on peut conclure que les ensembles couvrants contextuels contenus par les dérivations les plus profondes sont engendrés seulement par des modules de raisonnement inconditionnels.

La certitude qu'un ensemble de formules, calculé avec la fonction de génération d'un module de raisonnement conditionnel, est un ensemble couvrant contextuel dépend non seulement de la correction des techniques de raisonnement, mais aussi de la correction de la preuve de ses conditions.

Dans la section suivante, on va montrer les propriétés de correction et de correction réfutationnelle de $\mathsf{A}(\mathcal{RM})$.

3.5.3 Propriétés de $\mathsf{A}(\mathcal{RM})$

Le théorème suivant établit la correction du $\mathsf{A}(\mathcal{RM})$. Pour cela, il suffit de montrer que le schéma d'intégration des modules de raisonnement conditionnels de $\mathcal{RM}$ dans le système d'inférence A permet à ces modules d'engendrer des ensembles couvrants contextuels.

Théorème 3.5.1 (correction de $\mathsf{A}(\mathcal{RM})$) *Soient E un ensemble de formules et Ax un ensemble d'axiomes tels que $Ax \vdash_{ind^*}^{\mathsf{A}(\mathcal{RM})} E$. Alors $Ax \models_{ind^*} E$.*

Preuve Par récurrence sur la profondeur d'une $\mathsf{A}(\mathcal{RM})$-preuve arbitraire, on va montrer la propriété P suivante : pour chacune de ses dérivations arborescentes dont la dérivation principale est de la forme $(E^0, H^0) \vdash^{\mathsf{A}(\mathcal{RM})} \ldots \vdash^{\mathsf{A}(\mathcal{RM})} (\emptyset, H^m)$, on a $Ax \cup H^0_{\leq \phi\tau} \models_{ind^*} \phi\tau$, pour toute formule $\phi \in E^0$ et substitution close τ.

1. *le cas de base.* On suppose que la profondeur de la $\mathsf{A}(\mathcal{RM})$-preuve est 0. A partir de la définition 3.5.4, il résulte que la preuve est linéaire, que H^0 est vide et que des modules de raisonnement inconditionnels ont été utilisés pendant la dérivation. Par la définition 3.5.1, tout module de raisonnement employé construit un ensemble couvrant contextuel. Conformément au théorème 3.3.1, on a $Ax \models_{ind^*} E$.
2. *le pas de récurrence.* Par hypothèse de récurrence, on suppose que toute $\mathsf{A}(\mathcal{RM})$-dérivation arborescente d'une profondeur inférieure ou égale à une valeur naturelle arbitraire n ($n \geq 1$) satisfait la propriété P. On va montrer que toute $\mathsf{A}(\mathcal{RM})$-dérivation arborescente D de profondeur $n+1$ satisfait la propriété P pour conclure qu'elle est vraie pour toute $\mathsf{A}(\mathcal{RM})$-dérivation arborescente de la $\mathsf{A}(\mathcal{RM})$-preuve.

   Par la définition 3.5.4, on peut déduire que toute dérivation secondaire de D est d'une profondeur inférieure ou égale à n. Elles satisfont la propriété P, par hypothèse de récurrence.

   On suppose que M est un module de raisonnement conditionnel arbitraire de la dérivation principale de D, censé de produire un ensemble couvrant contextuel dans le contexte $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ pour une formule ϕ. On va analyser les conditions d'applicabilité de M, selon la définition 3.5.2.
   - Si $Ax \vdash_{ind^*}^{\mathsf{A}(\mathcal{RM})} c_M(\phi, Cxt)$ alors $Ax \models_{ind^*} c_M(\phi, Cxt)$. Donc, pour toute substitution close τ et formule $\psi \in c_M(\phi, Cxt)$, on a $Ax \cup \mathsf{C}^1_{\leq \phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \models_{ind^*} \psi\tau$.
   - Si $c_M(\phi, Cxt) \lll \{\phi\}$ et la $\mathsf{A}(\mathcal{RM})$-dérivation secondaire $(c_M(\phi, Cxt), \mathsf{C}^1) \vdash^{\mathsf{A}(\mathcal{RM})} \ldots$ finit avec succès, alors pour toute formule $\psi \in c_M(\phi, Cxt)$ et toute substitution close τ, on a $Ax \cup \mathsf{C}^1_{\leq \psi\tau} \models_{ind^*} \psi\tau$. Par ailleurs, $\psi\tau \leq \phi\tau$, donc $\mathsf{C}^1_{\leq \psi\tau} \subseteq \mathsf{C}^1_{\leq \phi\tau} \subseteq (\mathsf{C}^1_{\leq \phi\tau} \cup \mathsf{C}^2_{<\phi\tau})$. Par conséquent, $Ax \cup \mathsf{C}^1_{\leq \phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \models_{ind^*} \psi\tau$.
   - Si $c_M(\phi, Cxt) \ll \{\phi\}$ et la $\mathsf{A}(\mathcal{RM})$-dérivation secondaire $(c_M(\phi, Cxt), \mathsf{C}^1 \cup \mathsf{C}^2) \vdash^{\mathsf{A}(\mathcal{RM})} \ldots$ finit avec succès, alors pour toute formule $\psi \in c_M(\phi, Cxt)$ et toute substitution close τ, on a $Ax \cup (\mathsf{C}^1 \cup \mathsf{C}^2)_{\leq \psi\tau} \models_{ind^*} \psi\tau$. D'autre part, $\psi\tau < \phi\tau$, donc $(\mathsf{C}^1_{\leq \psi\tau} \cup \mathsf{C}^2_{\leq \psi\tau}) \subseteq \mathsf{C}^1_{\leq \phi\tau} \cup \mathsf{C}^2_{<\phi\tau}$. Par conséquent, $Ax \cup \mathsf{C}^1_{\leq \phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \models_{ind^*} \psi\tau$.

   Dans tous les cas, la relation $Ax \cup \mathsf{C}^1_{\leq \phi\tau} \cup \mathsf{C}^2_{<\phi\tau} \models_{ind^*} c_M(\phi, Cxt)\tau$ est satisfaite pour toute substitution close τ. Selon la définition 3.5.1, M calcule un ensemble couvrant contextuel de ϕ.

   Par la proposition 3.3.1, il résulte que D satisfait la propriété P, pour laquelle H^0 est vide.

Fin de preuve

Par exemple, la conjecture C de l'exemple 3.5.4 a été prouvée avec $\mathsf{A}(\mathcal{RM})$, où $\mathcal{RM} = \{\mathcal{CA}, \mathcal{CR}, \mathcal{PA}\}$. Donc, elle est une conséquence initiale des axiomes (3.1) et (3.2), par le théorème 3.5.1.

Théorème 3.5.2 (correction et complétude réfutationnelle de $\mathsf{A}(\mathcal{RM})$) *Le système d'inférence $\mathsf{A}(\mathcal{RM})$ est réfutationnellement correct si le critère de la plus petite couverture est satisfait.*

Le système d'inférence $\mathsf{A}(\mathcal{RM})$ est réfutationnellement complet si $\mathsf{A}(\mathcal{RM})$ est équitable et l'existence d'une conjecture contenant un contre-exemple minimal de la dérivation équitable $(E^0, \emptyset) \vdash^{\mathsf{A}} (E^1, H^1) \vdash^{\mathsf{A}} \ldots$ implique i) l'existence d'un dernier état i dans la dérivation, et ii) l'existence d'une conjecture de cet état, contenant un contre-exemple minimal telle que $\mathtt{Fail}(E^i)$ est vrai.

Preuve Tout module de raisonnement appelé dans une $\mathsf{A}(\mathcal{RM})$-dérivation linéaire principale retourne un ensemble couvrant contextuel, comme cela a déjà été démontré dans la preuve du théorème 3.5.1. Par le théorème 3.3.3, le système d'inférence $\mathsf{A}(\mathcal{RM})$ est réfutationnellement correct. Par le théorème 3.3.4, $\mathsf{A}(\mathcal{RM})$ est réfutationnellement complet.

Fin de preuve

3.6 Conclusions

Les ensembles couvrants contextuels sont utilisés pour définir uniformément un système abstrait d'inférence, basé sur la récurrence implicite. Celui-ci permet d'établir le contenu du contexte maximal des ensembles couvrants contextuels employés. Nous avons montré qu'il est plus général que d'autres procédures similaires.

Les ensembles couvrants contextuels sont mis en œuvre par des modules de raisonnement conditionnels et inconditionnels. La vérification des conditions d'un module de raisonnement conditionnel peut se faire par des appels récursifs au système abstrait d'inférence selon un schéma d'intégration qui ne perturbe pas les propriétés de correction, correction réfutationnelle et complétude réfutationnelle du système abstrait. Quelques avantages de cette approche sont i) la simplicité du schéma d'intégration des modules de raisonnement conditionnel dans le système abstrait, et ii) l'utilisation des éléments du contexte de l'ensemble couvrant contextuel implémenté par le module de raisonnement comme prémisses initiales dans la preuve des conditions.

4

Une instance de $\mathsf{A}(\mathcal{RM})$: le démonstrateur automatique SPIKE

Sommaire

4.1 Introduction

Les implantations du système d'inférence $\mathsf{A}(\mathcal{RM})$, décrit dans le chapitre 3, correspondent à des procédures exécutables qui peuvent s'obtenir en spécifiant

- le domaine de raisonnement,
- la classe de formules du langage $\mathcal{L}$ et l'ensemble d'axiomes,
- le pré-ordre global (bien fondé et fortement stable par substitution) $\leq$ sur les formules,
- l'ensemble de techniques de raisonnement à mettre en œuvre et les modules de raisonnement élémentaires associés,
- les règles d'inférence définies en termes d'ensembles couvrants contextuels, et
- la stratégie d'application des règles pendant les dérivations.

Dans ce chapitre, on se focalise sur l'analyse d'une instance du système d'inférence J, présentée dans [Bouhoula, 1997] comme une généralisation des procédures antérieurement définies dans [Bouhoula et Rusinowitch, 1995a] qui décrivent le système d'inférence du démonstrateur automatique SPIKE [Bouhoula et Rusinowitch, 1995b], ainsi qu'une de ses variantes adaptée aux spécifications paramétrées [Bouhoula, 1996].

SPIKE est capable de vérifier des relations de conséquences dans le modèle initial des spécifications algébriques multi-sortées dont les axiomes sont des équations conditionnelles et les conjectures (formules) sont des clauses équationnelles. Le pré-ordre global sur les clauses équationnelles est $\preceq_c$ de la définition 1.4.8 † et l'ordre sur les termes est $<_{MPO}$ de la définition 1.4.7.

Dans la suite, on va introduire les techniques de raisonnement utilisées par SPIKE et on va montrer qu'elles permettent de construire des modules de raisonnement. Puis, on va présenter chacune des règles d'inférence de SPIKE en tant qu'instance des règles du système abstrait $\mathsf{A}(\mathcal{RM})$ dont les ensembles couvrants sont des (compositions d') ensembles couvrants contextuels engendrés par ces modules de raisonnement. A part la validation des propriétés de correction et de complétude de SPIKE, ce résultat d'instanciation apporte des améliorations qu'on va illustrer à la fin du chapitre.

4.2 Un système d'inférence récursif de SPIKE

4.2.1 Techniques de raisonnement

Les principales techniques de raisonnement utilisées par SPIKE sont *la réécriture inductive*, *l'analyse par cas* et l'élimination des *tautologies* et des *clauses subsumées*. Comme d'habitude, Ax va dénoter l'ensemble d'axiomes et R le système de réécriture conditionnelle obtenu par l'orientation des axiomes de Ax.

Définition 4.2.1 (réécriture inductive [Bouhoula, 1997]) *Soient* H *un ensemble d'équations conditionnelles et* $C[l\sigma]$ *une clause équationnelle. Alors on écrit* $l\sigma \xrightarrow{C}_{R\langle\mathsf{H}\rangle} r\sigma$ *s'il existe une équation conditionnelle* $e \in R \cup \mathsf{H}$ *de la forme* $\wedge_{i=1}^{n} a_i = b_i \Rightarrow l = r$ *telle que*

1. *si* $e \in \mathsf{H}$, *alors* $e\sigma \prec_c C$ *et* $(\cup_{i=1}^{n}\{a_i\sigma,\, b_i\sigma\}) \ll_t \{l\sigma\}$, *et*
2. *pour tout* $i \in [1..n]$, *il existe un terme* c_i *tel que* $a_i\sigma \xrightarrow{C}{}^{*}_{R\langle\mathsf{H}\rangle} c_i$ *et* $b_i\sigma \xrightarrow{C}{}^{*}_{R\langle\mathsf{H}\rangle} c_i$.

La relation de réécriture inductive $\rightarrow_{R\langle\mathsf{H}\rangle}$ *est définie comme suit. Une clause* $C[a]_u$ *est* inductivement réécrite *à la position* u *ssi* $a \xrightarrow{C}_{R\langle\mathsf{H}\rangle} a'$ *et* $C[a']_u \prec_c C[a]_u$. *Alors on écrit* $C[a]_u \rightarrow_{R\langle\mathsf{H}\rangle} C[a']_u$.

Observons que la relation $C[a']_u \prec_c C[a]_u$ est satisfaite si a est un sous-terme d'un terme $\preceq_t$-maximal de C.

La technique de raisonnement suivante combine les opérations d'analyse par cas et de réécriture conditionnelle, définies respectivement dans les exemples 3.5.1 et 3.5.2, afin de simplifier une clause avec des règles de réécriture conditionnelles.

Définition 4.2.2 (analyse par cas) *Soient* $C[s]$ *une clause équationnelle et* $\cup_{i=1}^{n}\{P_i \Rightarrow l_i \rightarrow r_i\}$ *un ensemble de règles de réécriture conditionnelles de* R *tels que pour chaque* $i \in [1..n]$, *il existe une position* u_i *et une substitution* σ_i *pour laquelle* $s/u_i = l_i\sigma_i$. *Alors* CaseAnalysis$(C[s])$ *retourne l'ensemble* $\overline{C} = \{P_1\sigma_1 \Rightarrow C[s_1], \ldots, P_n\sigma_n \Rightarrow C[s_n]\}$ *si* $Ax \models_{ini} [\bigvee_{i=1}^{n} P_i\sigma_i]^{cnf}$, *où* $s_i = s[r_i\sigma_i]_{u_i}$.

Définition 4.2.3 (tautologie) *Une* tautologie *est une clause équationnelle contenant dans le conséquent une équation de la forme* $t = t$.

† qui est moins restrictif que l'ordre décroissant utilisé dans [Bouhoula, 1997]

GENERATE : $(E \cup \{C\},\ H) \vdash^{J'} (E \cup (\cup_\sigma E_\sigma),\ H \cup \{C\})$
si pour toute substitution couvrante $\sigma \in CS\Sigma(C)$,
- soit $C\sigma$ est une tautologie et $E_\sigma = \emptyset$,
- soit $C\sigma \rightarrow_{R\langle H \cup E \cup \{C\}\rangle} C'$ et $E_\sigma = \{C'\}$
- sinon, $E_\sigma = RecursiveCaseAnalysis'(C\sigma)$

CASE SIMPLIFY : $(E \cup \{C\},\ H) \vdash^{J'} (E \cup E',\ H)$
si $E' = RecursiveCaseAnalysis(C)$

SIMPLIFY : $(E \cup \{C\},\ H) \vdash^{J'} (E \cup \{C'\},\ H)$
si $C \rightarrow_{R\langle H \cup E\rangle} C'$

SUBSUME : $(E \cup \{C\},\ H) \vdash^{J'} (E,\ H)$
si C est subsumée par une autre clause de $R \cup H \cup E$

TAUTOLOGY : $(E \cup \{C\},\ H) \vdash^{J'} (E,\ H)$
si C est une tautologie

FIG. 4.1 – *Le système d'inférence J'*

Afin d'introduire la notion de subsomption, on précise qu'une clause $C_1 \equiv a_1 \wedge \ldots \wedge a_n \Rightarrow b_1 \vee \ldots \vee b_m$ est une *sous-clause* de la clause $C_2 \equiv a'_1 \wedge \ldots \wedge a'_r \Rightarrow b'_1 \vee \ldots \vee b'_s$ ssi $\{a_1, \ldots, a_n\} \setminus\setminus \{a'_1, \ldots, a'_r\} = \emptyset$ et $\{b_1, \ldots, b_m\} \setminus\setminus \{b'_1, \ldots, b'_s\} = \emptyset$.

Définition 4.2.4 (subsomption clausale syntaxique) *Soient C_1 et C_2 deux clauses. On dit que C_1* subsume *C_2 si $C_1\sigma$ est une sous-clause de C_2 pour une substitution σ.*

Dans cette définition, notons que $C_1\sigma \preceq_c C_2$.

4.2.2 Le système d'inférence J'

Dans la définition 4.2.2, on peut observer qu'on n'a pas spécifié la manière dont le test de conséquence initiale de $\bigvee_{i=1}^{n} P_i\sigma_i$ est fait. Soit la fonction *RecursiveCaseAnalysis* une variante d'implantation de *CaseAnalysis* pour laquelle la vérification se fait par un appel récursif au système d'inférence, comme décrit dans la section 3.5.2.

Dans la suite, on analyse une implantation récursive de J, notée par J' et présentée dans la figure 4.1. Elle dérive de J en remplaçant *CaseAnalysis* par *RecursiveCaseAnalysis*. De plus, si dans la définition 4.2.2 on limite s d'être un sous-terme d'un terme $\preceq_t$-maximal de C, on obtient la règle *RecursiveCaseAnalysis'*. Cette restriction garantit que toute clause de $\overline{C}$ est plus petite que C. Notons que celle-ci est implicite dans la définition de la fonction *CaseAnalysis* de [Bouhoula, 1997] parce que l'ordre décroissant permet de dériver $C[a]_p \prec_c C[a']_p$ lorsque $a \prec_t a'$, pour toute clause C, position p et tous les termes a et a'.

La règle GENERATE construit d'abord un ensemble couvrant pour la conjecture traitée par son instanciation successive avec des substitutions couvrantes de $CS\Sigma(C)$, comme dans l'exemple 2.3.1. Puis, chaque instance est soit éliminée, si elle est une tautologie, soit simplifiée par une réécriture inductive ou bien par une opération d'analyse par cas. La conjecture traitée est remplacée dans un premier temps dans l'ensemble courant de conjectures par l'ensemble de

nouvelles conjectures qui représentent le résultat des opérations de simplification. Puis elle est ajoutée à l'ensemble de prémisses. La règle CASE SIMPLIFY (respectivement SIMPLIFY) permet la simplification de la conjecture traitée par une opération d'analyse par cas (respectivement, de réécriture inductive). Les règles SUBSUME et TAUTOLOGY éliminent respectivement des clauses redondantes triviales comme les clauses subsumées et les tautologies.

Modules de raisonnement

Maintenant, on va définir un ensemble de modules de raisonnement à partir des techniques de raisonnement décrites dans la section 4.2.1.

Le module de raisonnement *IR* On dénote par IR un module de raisonnement inconditionnel basé sur la réécriture inductive qu'on va spécifier dans la suite. Pour cela, on suppose que H est un ensemble d'équations conditionnelles et C, C' deux clauses équationnelles telles que $C \rightarrow_{R\langle \mathsf{H}\rangle} C'$. La fonction de génération de IR est définie comme $g_{IR}(C, (\emptyset, \mathsf{H})) = \{C'\}$ si $C \rightarrow_{R\langle \mathsf{H}\rangle} C'$. Alors, $\{C'\}$ est un ensemble couvrant contextuel strict de C, comme on le montre par la proposition suivante.

Proposition 4.2.1 *Soient C et C' deux clauses équationnelles et H un ensemble d'équations conditionnelles tel que $C \rightarrow_{Ax\langle \mathsf{H}\rangle} C'$. Alors i) pour toute substitution close τ, $Ax \cup \mathsf{H}_{\prec_c C\tau} \cup \{C\tau\} \models_{ini} C'\tau$, et ii) C' est un ensemble contextuel strict de C dans le contexte $(\emptyset, \mathsf{H})$.*

Preuve On suppose que $C \rightarrow_{R\langle \mathsf{H}\rangle} C'$. Alors, il existe un terme a de C qui se réécrit en a' pour produire la clause C'. Conformément à la définition 4.2.1, pendant le processus de réécriture, on n'a utilisé que des équations conditionnelles de $R \cup \mathsf{H}_{\prec_c C}$. Donc, $Ax \cup \mathsf{H}_{\prec_c C\tau} \models_{ini} (a = a')\tau$ pour toute substitution close τ. Alors i) $Ax \cup \mathsf{H}_{\prec_c C\tau} \cup \{C[a]_p\tau\} \models_{ini} C[a']_p\tau$, et aussi ii) $Ax \cup \mathsf{H}_{\prec_c C\tau} \cup \{C[a']_p\tau\} \models_{ini} C[a]_p\tau$. D'une part, le critère de couverture minimale est satisfait, car $Ax \cup \mathsf{H}_{\prec_c C\tau} \cup \{C\tau\} \models_{ini} C'\tau$ en utilisant le premier fait.

D'autre part, selon la définition 4.2.1, on a $C' \prec_c C$. Puisque $\prec_c$ est stable par substitution, on a $C'\theta \prec_c C\theta$ pour toute substitution θ et en particulier pour τ. Par conséquent, le deuxième fait ii) devient $Ax \cup \mathsf{H}_{\prec_c C\tau} \cup C'_{\prec_c C\tau} \models_{ini} C\tau$. Il résulte que C' est un ensemble couvrant contextuel strict de C dans le contexte $(\emptyset, \mathsf{H})$.

Fin de preuve

Les modules de raisonnement *RCA* et *RCA′* Le module de raisonnement conditionnel RCA met en œuvre l'opération d'analyse par cas de la définition 4.2.2 afin de construire un ensemble couvrant contextuel strict. Etant donnés une clause C et un contexte $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$, la fonction de génération associée à RCA, $g_{RCA}(C, Cxt)$, retourne un ensemble de clauses $\overline{C} = \{P_1\sigma_1 \Rightarrow C[s_1], \ldots, P_n\sigma_n \Rightarrow C[s_n]\}$ s'il existe une équation de C de la forme $s = t$ ou $t = s$ telle qu'il existe un ensemble de règles $\cup_{i=1}^{n}\{P_i \Rightarrow l_i \rightarrow r_i\}$ de $R \cup \mathsf{C}^1_{\preceq_c C} \cup \mathsf{C}^2_{\prec_c C}$ tel que pour chaque $i \in [1..n]$ on a $s/u_i = l_i\sigma_i$ et $s_i = s[r_i\sigma_i]_{u_i}$. La fonction de condition de RCA est $c_{RCA}(C, Cxt) = \{\bigvee_{i=1}^{n} P_i\sigma_i\}$. Le module de raisonnement RCA' est identique à RCA, sauf que s doit être un sous-terme d'un terme $\preceq_t$-maximal de C.

L'ensemble couvrant contextuel se construit en deux étapes, comme dans l'exemple 2.3.3. D'abord, $\overline{C'} = \{P_1\sigma_1 \Rightarrow C[s\sigma_1], \ldots, P_n\sigma_n \Rightarrow C[s\sigma_n]\}$ est un ensemble couvrant contextuel de C dans le contexte Cxt, comme dans l'exemple 3.5.1. Puis, pour chaque $i \in [1..n]$, $\{P_i\sigma_i \Rightarrow C[s_i]\}$ est un ensemble couvrant (strict pour RCA') de $P_i\sigma_i \Rightarrow C[s\sigma_i]$ car $Ax \cup \mathsf{C}^1_{\preceq_c C} \cup \mathsf{C}^2_{\prec_c C} \models_{ini} P_i \Rightarrow s_i = s$ et $s_i \prec_t s$. Par conséquent, $\overline{C}$ est un ensemble couvrant contextuel (strict pour

RCA') de C dans le contexte Cxt. De plus, pour chaque $\phi \in \overline{C}$ et toute substitution close τ, $Ax \cup \mathsf{C}^1_{\preceq_c C} \cup \mathsf{C}^2_{\prec_c C} \cup \{C\tau\} \models_{ini} \phi\tau$. Donc, le critère de la couverture minimale est satisfait par toute règle d'inférence qui remplacerait C par $\overline{C}$.

Les modules de raisonnement $CS\Sigma$, T et CS Enfin, on va définir les modules de raisonnement inconditionnels $CS\Sigma$, T et CS dont les fonctions de génération sont, respectivement :

- $g_{CS\Sigma}(C, (\mathsf{C}^1, \mathsf{C}^2))$, qui retourne l'ensemble $\overline{C} = \{C\sigma \mid \sigma \in CS\Sigma(C)\}$, défini dans l'exemple 2.3.1. D'une part, $\overline{C}$ est un ensemble couvrant de C, comme il est déjà mentionné dans cet exemple. D'autre part, pour chaque $C' \in \overline{C}$ et toute substitution close τ, il existe une substitution close τ' telle que $Ax \cup \mathsf{C}^1_{\preceq_c C} \cup \mathsf{C}^2_{\prec_c C} \cup \{C\tau'\} \models_{ini} C'\tau$. Si C' est l'instance $C\sigma$, où $\sigma \in CS\Sigma(C)$, il suffit de prendre $\tau' = \sigma\tau$. Par conséquent, le critère de la plus petite couverture est satisfait par toute règle d'inférence qui remplacerait C par $\overline{C}$.
- $g_T(C, Cxt)$ est un ensemble couvrant contextuel vide de C pour tout contexte Cxt. Ceci est dû au fait que C est une tautologie et on a $\emptyset \models_{ini} C'$, pour toute tautologie C'.
- $g_{CS}(C, (\mathsf{C}^1, \mathsf{C}^2))$ est un ensemble couvrant contextuel vide de C dans le contexte $(\mathsf{C}^1, \mathsf{C}^2)$ sous la supposition que C est subsumée par une des clauses de $Ax \cup \mathsf{C}^1_{\preceq_c C} \cup \mathsf{C}^2_{\prec_c C}$. Par conséquent, $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \models_{ini} C\tau$, pour toute substitution close τ.

4.2.3 Correction et correction réfutationnelle de J'

Pour montrer la correction et correction réfutationnelle de J', il suffit de justifier que J' est une instance de $\mathsf{A}(\mathcal{RM})$ en utilisant les modules de raisonnement IR, RCA, $CS\Sigma$, T et CS.

Théorème 4.2.1 (correction et correction réfutationnelle de J') *On suppose que $\mathcal{RM}$ est l'ensemble de modules de raisonnement $\{IR, RCA, RCA', CS\Sigma, T, CS\}$. Alors, le système d'inférence J' est une instance correcte et réfutationnellement correcte de $\mathsf{A}(\mathcal{RM})$.*

Preuve On va analyser l'une après l'autre les règles d'inférence de J'. D'abord, on identifie la A-règle qui est instanciée par une J'-règle. Puis, on construit l'ensemble couvrant contextuel approprié à chaque étape de la A-règle, en définissant le module de raisonnement qui l'engendre et le contexte. On suppose que la J'-règle est appliquée à la clause C à l'état arbitraire $(E \cup \{C\}, H)$ d'une J'-dérivation.

- Generate est une instance de A-AddPremise, définie dans la figure 3.1. L'ensemble couvrant contextuel de C dont on a besoin à l'étape (a) de A-AddPremise est $\overline{C} = g_{CS\Sigma}(C, (\emptyset, \emptyset))$. Comme il n'est pas vide, pour chaque élément $C\sigma$ de $\overline{C}$, E_σ est soit un ensemble couvrant contextuel strict de $C\sigma$ engendré à l'étape (b') par $g_{RCA'}(C\sigma, (\emptyset, \emptyset))$ ou $g_{IR}(C\sigma, (\emptyset, H \cup E \cup \{C\})$, soit un ensemble couvrant contextuel vide de l'étape (b') construit par $g_T(C\sigma, (\emptyset, \emptyset))$. D'une part, à chacune des étapes, le contexte utilisé par Generate est plus petit que le contexte maximal permis par A-AddPremise et, d'autre part, le critère de la plus petite couverture est satisfait au moment du passage de $\{C\}$ à $\overline{C}$ et de tout $C\sigma$ de $\overline{C}$ à E_σ. Par conséquence, le critère de la plus petite couverture est aussi satisfait par Generate.
- Case Simplify est une instance de A-Simplify. A l'étape (a), l'ensemble couvrant de C est $\{C\}$. A l'étape (b'), l'ensemble E' est $g_{RCA}(C, (\emptyset, \emptyset))$ qui est un ensemble couvrant de C. Le critère de la plus petite couverture est satisfait par Case Simplify, pour des raisons similaires au cas précédent.

POSITIVE DECOMPOSITION : $(E \cup \{f(\overrightarrow{s}) = f(\overrightarrow{t}) \vee r\},\ H) \vdash^{J'} (E \cup (\cup_{i=1}^{n}\{s_i = t_i \vee r\}),\ H)$
si f est un symbole de constructeur libre.

NEGATIVE DECOMPOSITION : $(E \cup \{\neg(f(\overrightarrow{s}) = f(\overrightarrow{t})) \vee r\},\ H) \vdash^{J'} (E \cup \{\vee_{i=1}^{n}\neg(s_i = t_i) \vee r\},\ H)$
si f est un symbole de constructeur libre.

POSITIVE CLASH : $(E \cup \{f(\overrightarrow{s}) = g(\overrightarrow{t}) \vee r)\},\ H) \vdash^{J'} (E \cup \{r\},\ H)$
si f et g sont deux symboles de constructeurs libres distincts.

ELIMINATE TRIVIAL EQUATION : $(E \cup \{\neg(s = s) \vee r\},\ H) \vdash^{J'} (E \cup \{r\},\ H)$

DELETE : $(E \cup \{\bigvee_{i=1}^{n} \neg(x_i = t_i) \vee r\},\ H) \vdash^{J'} (E,\ H)$
si pour tout $i : x_i \notin Var(t_i)$ et $r\rho$ est une tautologie, où $\rho = \{x_i \leftarrow t_i \mid i \in [1..n]\}$.

OCCUR CHECK : $(E \cup \{\bigvee_{i=1}^{n} \neg(x_i = t_i) \vee r\},\ H) \vdash^{J'} (E,\ H)$
s'il existe $i \in [1..n]$ tel que x_i est différente de t_i et $x_i \in Var(t_i)$
et t_i est un terme constructeur qui est inductivement R-irréductible.

NEGATIVE CLASH : $(E \cup \{\neg(f(\overrightarrow{s}) = g(\overrightarrow{t})) \vee r\},\ H) \vdash^{J'} (E,\ H)$
si f et g sont deux symboles de constructeurs libres distincts.

FIG. 4.2 – *Le système d'inférence* J' *(suite)*

- SIMPLIFY est une instance de A-SIMPLIFY. L'ensemble couvrant contextuel de C à l'étape (a) est $\{C\}$. L'ensemble couvrant contextuel strict de C à l'étape (b') est $\{C'\} = g_{IR}(C, (\emptyset, H \cup E))$, conformément à la proposition 4.2.1. De plus, le contexte $(\emptyset, H \cup E)$ est plus petit que le contexte maximal permis par A-SIMPLIFY, c'est-à-dire $(H \cup E, \emptyset)$. Par la même proposition, on établit que la règle satisfait le critère de la plus petite couverture.
- SUBSUME est une instance de A-DELETE. L'ensemble couvrant contextuel à l'étape (a) est $\{C\}$ et l'ensemble couvrant contextuel vide à l'étape (b') est $g_{CS}(C, (E \cup H, \emptyset))$. Le contexte $(E \cup H, \emptyset)$ est égal au contexte maximal permis par A-DELETE à cette étape.
- TAUTOLOGY est aussi une instance de A-DELETE. L'ensemble couvrant contextuel à l'étape (a) est $\{C\}$ et l'ensemble couvrant contextuel vide de C à l'étape de (b') est $g_T(C, (\emptyset, \emptyset))$.

Par le théorème 3.5.1, le système J' est correct et, par le théorème 3.5.2, il est réfutationnellement correct.

Fin de preuve

4.2.4 Extension de J' par des règles d'inférence structurelles

L'application des règles d'inférence GENERATE, CASE SIMPLIFY et SIMPLIFY aide à simplifier les conjectures d'une dérivation. Si le système de réécriture conditionnelle R est suffisamment complet, alors ces règles peuvent s'appliquer tant qu'il y a des symboles de fonctions définies dans les conjectures. Afin de décider sur la validité de toute clause ne contenant que des termes constructeurs, on va étendre l'ensemble des règles d'inférence de J' avec les règles d'inférence structurelles suivantes [Bouhoula, 1997] : POSITIVE DECOMPOSITION, NEGATIVE DECOMPOSITION, POSITIVE CLASH, ELIMINATE TRIVIAL EQUATIONS, DELETE, OCCUR CHECK et NEGATIVE CLASH, présentées dans la figure 4.2.

Dans la suite, on va étudier les propriétés du système J' ainsi étendu. La propriété de convergence sur les termes clos de R doit être satisfaite afin obtenir la correction et la correction réfutationnelle de l'extension du système J'.

Théorème 4.2.2 *L'extension du système d'inférence J' est correcte et réfutationnellement correcte si le système de réécriture R est convergent sur les termes clos.*

Preuve On définit les modules de raisonnement inconditionnels PD, ND, PC, ETE, D, OC et NC, dont les fonctions de génération sont, respectivement :

- $g_{PD}(C, Cxt)$ retourne l'ensemble $\overline{C} \equiv \cup_{i=1}^{n}\{s_i = t_i \vee r\}$ si C est la clause $f(\vec{s}) = f(\vec{t}) \vee r$ et f est un symbole de constructeur libre. $\overline{C}$ est un ensemble couvrant de C si pour toute substitution close τ on a $Ax \cup \overline{C}_{\preceq_c C\tau} \models_{ini} C\tau$. Cette relation est satisfaite car i) par la propriété de sous-terme, pour tout $i \in [1..n]$ le terme s_i (resp. t_i) est plus petit que $f(\vec{s})$ (resp. $f(\vec{t})$), ce qui conclut que $(s_i = t_i \vee r)\tau \preceq_c (f(\vec{s}) = f(\vec{t}) \vee r)\tau$, et ii) $Ax \models_{ini} (f(\vec{s}) = f(\vec{t}) \vee r)\tau$ si, pour tout $i \in [1..n]$, $Ax \models_{ini} (s_i = t_i \vee r)\tau$.
- $g_{ND}(C, Cxt)$ est $\{C' \equiv \vee_{i=1}^{n}\neg(s_i = t_i) \vee r\}$ si C est la clause $\neg(f(\vec{s}) = f(\vec{t})) \vee r$ et f est un symbole de constructeur libre. On va montrer que $\{C'\}$ est un ensemble couvrant de C. Soit τ une substitution close. D'une part, on a $Ax \cup \{C'\tau\} \models_{ini} C\tau$. Ceci résulte du fait que $\models_{ini} \vee_{i=1}^{n}\neg(s_i = t_i)\tau$ implique $\models_{ini} \neg(f(\vec{s}) = f(\vec{t}))\tau$ puisque R est convergent sur les termes clos et f est un symbole de constructeur libre. D'autre part, $C'\tau \preceq_c C\tau$. Par conséquent, $Ax \cup \{C'\}_{\preceq_c C\tau} \models_{ini} C\tau$.
- $g_{PC}(C, Cxt) = \{r\}$ si C est de la forme $f(\vec{s}) = g(\vec{t}) \vee r$ et f, g sont deux symboles de constructeurs libres distincts. Pour toute substitution close τ, on a d'une part $Ax \cup \{r\}_{\preceq_c C\tau} \models_{ini} (f(\vec{s}) = g(\vec{t}) \vee r)\tau$ car $r \preceq_c C$ et d'autre part $\not\models_{ind} f(\vec{s}) = g(\vec{t})$ parce que f et g sont deux symboles de constructeurs libres distincts. Donc, $\{r\}$ est un ensemble couvrant de C.
- $g_{ETE}(C, Cxt)$ retourne $\{r\}$ comme ensemble couvrant de toute clause C de la forme $\neg(s = s) \vee r$. Pour toute substitution close τ, on a $Ax \cup \{r\}_{\preceq_c C\tau} \models_{ini} (\neg(s = s) \vee r)\tau$.
- $g_D(C, Cxt)$ est un ensemble couvrant vide de C si C est une clause de la forme $\bigvee_{i=1}^{n}\neg(x_i = t_i) \vee r$ telle que pour chaque $i \in [1..n]$, $x_i \notin Var(t_i)$ et $r\rho$ est une tautologie, où $\rho = \{x_i \leftarrow t_i \mid i \in [1..n]\}$. Par conséquent, $Ax \models_{ini} \bigvee_{i=1}^{n}\neg(x_i = t_i) \vee r$.
- $g_{OC}(C, Cxt)$ est aussi un ensemble couvrant vide de $C \equiv \bigvee_{i=1}^{n}\neg(x_i = t_i) \vee r$ s'il existe $i \in [1..n]$ tel que x_i est différente de t_i et $x_i \in Var(t_i)$, où t_i est un terme constructeur qui est inductivement R-irréductible. On a $Ax \models_{ini} \bigvee_{i=1}^{n}\neg(x_i = t_i) \vee r$.
- $g_{NC}(C, Cxt)$ est un ensemble couvrant vide de $C \equiv \neg(f(\vec{s}) = g(\vec{t})) \vee r$ si f et g sont deux symboles de constructeurs libres distincts. Comme $f(\vec{s})$ et $g(\vec{t})$ ne sont pas joignables et R est un système de réécriture convergent, on déduit que $Ax \models_{ini} \neg(f(\vec{s}) = g(\vec{t})) \vee r$.

On ajoute à $\mathcal{RM}$ les modules de raisonnement PD, ND, PC, ETE, D, OC et NC ainsi définis. On suppose que la conjecture traitée est la clause C de l'étape courante $(E \cup \{C\}, H)$ d'une J'-dérivation arbitraire. Comme précédemment, on va définir chaque nouvelle règle d'inférence de J' en tant qu'une instance d'une A-règle dont $\{C\}$ est l'ensemble couvrant contextuel de C qu'on a besoin à l'étape (a). Pour chacun d'entre eux, on va seulement spécifier l'ensemble couvrant contextuel à l'étape (b') :

- Positive Decomposition est une instance de A-Simplify. L'ensemble couvrant contextuel de l'étape (b') est $g_{PD}(f(\vec{s}) = f(\vec{t}) \vee r, (\emptyset, \emptyset))$. Pour chaque substitution close τ, on

a $Ax \cup \{(f(\overrightarrow{s}) = f(\overrightarrow{t}))\tau\} \models_{ini} [(\wedge_{i=1}^{n} s_i = t_i)]^{\mathtt{cnf}}\tau$ puisque R est convergent sur les termes clos et f est un symbole de constructeur libre. Donc, le critère de la plus petite couverture est satisfait.

- Negative Decomposition est une instance de A-Simplify. L'ensemble couvrant contextuel est $g_{ND}(\neg(f(\overrightarrow{s}) = f(\overrightarrow{t}) \vee r, (\emptyset, \emptyset))$. Comme pour la règle précédente, on peut montrer que le critère de la plus petite couverture est aussi satisfait.
- Positive Clash est une instance de A-Simplify. L'ensemble couvrant contextuel est $g_{PC}(f(\overrightarrow{s}) = g(\overrightarrow{t}) \vee r, (\emptyset, \emptyset))$. Le critère de la plus petite couverture est satisfait car, pour toute substitution close τ, on a $Ax \cup \{(f(\overrightarrow{s}) = g(\overrightarrow{t}) \vee r)\tau\} \models_{ini} r\tau$ puisque R est convergent sur les termes clos et f et g sont deux symboles de constructeurs libres distincts.
- Eliminate Trivial Equation est une instance de A-Simplify et l'ensemble couvrant contextuel est $g_{ETE}(\neg(s = s) \vee r, (\emptyset, \emptyset))$. Le critère de la plus petite couverture est aussi satisfait.
- Delete, Occur Check et Negative Clash sont des instances de A-Delete. L'ensemble couvrant contextuel vide est respectivement $g_D(\bigvee_{i=1}^{n} \neg(x_i = t_i) \vee r, (\emptyset, \emptyset))$, $g_{OC}(\bigvee_{i=1}^{n} \neg(x_i = t_i) \vee r, (\emptyset, \emptyset))$ et $g_{NC}(\neg(f(\overrightarrow{s}) = g(\overrightarrow{t})) \vee r, (\emptyset, \emptyset))$.

La correction et la correction réfutationnelle de J' résulte du théorème 3.5.1 et 3.5.2, respectivement.

Fin de preuve

Notons que, dans le cas où R n'est pas convergent sur les termes clos, alors le système J' privé des règles Negative Decomposition et Negative Clash reste quand même correct, tandis que le système J' sans les règles Positive Decomposition et Positive Clash est réfutationnellement correct. La propriété de convergence sur les termes clos évite d'avoir des relations d'égalité entre les termes constructeurs qui ne peuvent pas être déduites à partir de leur structure syntaxique. Par exemple, étant donné un système de réécriture conditionnelle R qui ne satisfait pas cette propriété et qui est suffisamment complet par rapport aux constructeurs, il existe un terme clos t qui a au moins deux termes constructeurs distincts et clos, c_1 et c_2, comme formes normales. Dans ce cas, $Ax \models_{ini} c_1 = c_2$, même si c_1 et c_2 sont syntaxiquement différents.

Le prédicat `Fail`, nécessaire pour réfuter des conjectures au long d'une dérivation, va être introduit dans la section 4.2.5. Sa définition ajoutera de nouvelles conditions sur R.

Dans le tableau 4.2.4, on présente des règles d'inférence de J', figurant dans la première colonne, comme des instances des A-règles de la deuxième colonne. Pour chaque règle de J', les deux dernières colonnes contiennent respectivement les noms des modules de raisonnement et les contextes utilisés aux étapes (a) et (b').

Le lemme suivant montre que toute application d'une J'-règle qui instancie A-Simplify réduit (par rapport à $\preceq_c$) la conjecture traitée.

Lemme 4.2.1 *Soit $\overline{C}$ l'ensemble de nouvelles conjectures obtenues par l'application à la conjecture C d'une J'-règle d'inférence instanciant* A-Simplify. *Alors $C' \preceq_c C$, pour toute $C' \in \overline{C}$.*

Preuve On va analyser chaque J'-règle qui instancie A-Simplify

- Toute nouvelle conjecture $C' \in \overline{C}$ est obtenue par Case Simplify en remplaçant dans C le terme auquel elle s'applique par un terme inférieur (par rapport à $\preceq_t$) et en ajoutant des conditions inférieures à lui. Selon la définition 1.4.8 de $\preceq_c$, on a $C' \preceq_c C$.

J'-règle	A-règle	g_{RM}, Cxt (étape a)	RM, Cxt (étape b')
GENERATE	ADDPREMISE	$CS\Sigma(C)$, $(\emptyset, \emptyset)$	RCA', $(\emptyset, \emptyset)$ (strict) IR, $(\emptyset, E \cup H \cup \{C\})$ (strict) T, $(\emptyset, \emptyset)$
CASE SIMPLIFY	SIMPLIFY	$\{C\}$, $(\emptyset, \emptyset)$	RCA, $(\emptyset, \emptyset)$ [a]
SIMPLIFY	SIMPLIFY	$\{C\}$, $(\emptyset, \emptyset)$	IR, $(\emptyset, E \cup H)$ (strict)
SUBSUME	DELETE	$\{C\}$, $(\emptyset, \emptyset)$	CS, $(E \cup H, \emptyset)$
TAUTOLOGY	DELETE	$\{C\}$, $(\emptyset, \emptyset)$	T, $(\emptyset, \emptyset)$
POSITIVE DECOMPOSITION	SIMPLIFY	$\{C\}$, $(\emptyset, \emptyset)$	PD, $(\emptyset, \emptyset)$
NEGATIVE DECOMPOSITION	SIMPLIFY	$\{C\}$, $(\emptyset, \emptyset)$	ND, $(\emptyset, \emptyset)$
POSITIVE CLASH	SIMPLIFY	$\{C\}$, $(\emptyset, \emptyset)$	PC, $(\emptyset, \emptyset)$
ELIMINATE TRIVIAL EQUATION	SIMPLIFY	$\{C\}$, $(\emptyset, \emptyset)$	ETE, $(\emptyset, \emptyset)$
DELETE	DELETE	$\{C\}$, $(\emptyset, \emptyset)$	D, $(\emptyset, \emptyset)$
OCCUR CHECK	DELETE	$\{C\}$, $(\emptyset, \emptyset)$	OC, $(\emptyset, \emptyset)$
NEGATIVE CLASH	DELETE	$\{C\}$, $(\emptyset, \emptyset)$	NC, $(\emptyset, \emptyset)$

TAB. 4.1 – *Les J'-règles d'inférence en tant qu'instances de* A*-règles*

[a] CASE SIMPLIFY calcule un ensemble couvrant contextuel strict dans [Bouhoula, 1997].

– L'application de SIMPLIFY à C engendre C' en remplaçant dans C le terme où elle s'applique par un terme inférieur. Pour des raisons similaires au cas précédent, $C' \preceq_c C$.
– Les nouvelles clauses dérivées par l'application de POSITIVE DECOMPOSITION, NEGATIVE DECOMPOSITION, POSITIVE CLASH et ELIMINATE TRIVIAL EQUATION sont obtenues par le remplacement d'un littéral de C par un ensemble (potentiellement vide) de littéraux inférieurs à lui.

Fin de preuve

SPIKE est capable de prouver des conjectures de manière automatique. Dans la suite, on présente la priorité sur l'application de ses règles d'inférence selon une stratégie qui essaie d'abord d'éliminer les clauses triviales ou redondantes, puis la normalisation des clauses et, finalement, la génération de nouveaux lemmes :

1. des règles structurelles qui instancient A-DELETE : TAUTOLOGY, DELETE, OCCUR CHECK et NEGATIVE CLASH ;
2. les autres règles qui instancient A-DELETE : SUBSUME ;
3. des règles structurelles qui instancient A-SIMPLIFY : POSITIVE DECOMPOSITION, NEGATIVE DECOMPOSITION, POSITIVE CLASH et ELIMINATE TRIVIAL EQUATION ;
4. les autres règles qui instancient A-SIMPLIFY : SIMPLIFY et CASE SIMPLIFY ;
5. des règles qui instancient A-ADDPREMISE : GENERATE.

4.2.5 Complétude réfutationnelle de J'

Dans cette section, on va étudier les conditions d'application du théorème 3.3.4 pour déduire la propriété de complétude réfutationnelle de J'.

Dans un premier temps, on va établir les conditions à satisfaire par le système de réécriture tel qu'au moins une des règles GENERATE, CASE SIMPLIFY et SIMPLIFY soit applicable à toute conjecture contenant au moins un symbole de fonction définie. Puis, si la conjecture est une *clause constructeur*, c'est-à-dire qui ne contient que des termes de $\mathcal{T}(\mathsf{CT}, \mathsf{V})$, on va utiliser les règles d'inférence structurelles pour décider de son incohérence, si les constructeurs sont libres

et le système de réécriture est convergent sur les termes clos. Dans ce cas, le prédicat d'échec `Fail`($\{C\}\cup E$) sera vrai s'il n'y a pas de J'-règle d'inférence qui s'applique à la clause constructeur C.

La notion de substitution couvrante n'est pas suffisante pour simplifier chaque instance couvrante avec des règles de réécriture lors de l'application de GENERATE.

Exemple 4.2.1 *On considère la spécification conditionnelle de l'exemple 1.5.1 et soit R le système de réécriture obtenu par l'orientation des équations conditionnelles de gauche à droite. Notons que la spécification est suffisamment complète. Soit* $C \equiv x \leq y = \text{True}$ *une clause sur laquelle on veut appliquer* GENERATE. *A l'étape (a) on utilise l'ensemble de substitutions couvrantes* $\{\{x \mapsto 0\}, \{x \mapsto s(x')\}\}$ *pour obtenir les instances* $0 \leq y = \text{True}$ *et* $s(x') \leq y = \text{True}$ *de C. Notons que la deuxième instance est R-irréductible, donc l'étape (b') n'est pas possible, impliquant l'échec de* GENERATE.

En général, ceci est dû i) au mauvais choix des variables qu'on instancie, et/ou ii) à l'ensemble couvrant utilisé qui n'est pas suffisamment «profond» pour que toutes les instances de la clauses soient filtrées par les parties gauches des règles de R. Une solution est de choisir l'ensemble de variables $\{x, y\}$ *et les substitutions couvrantes* $\{\{x \mapsto 0, y \mapsto 0\},\{\{x \mapsto 0, y \mapsto s(y')\},\{\{x \mapsto s(x'), y \mapsto 0\},\{\{x \mapsto s(x'), y \mapsto s(y')\}\}$. *Alors, les instances* $0 \leq 0 = \text{True}$ *et* $0 \leq s(y') = \text{True}$ *sont réduites par 1.1,* $s(x') \leq 0$ *par 1.2 et* $s(x') \leq s(y') = \text{True}$ *par 1.3.*

Pour éviter ce type d'échec, il suffit d'utiliser un schéma de récurrence particulier qui instancie (un sous-ensemble de) $Var(C)$ de la clause C, appelées des *variables de récurrence*, par des éléments d'un ensemble couvrant particulier de R, qu'on va appeler *ensemble test*. Dans ce cas, les substitutions couvrantes sont des *substitutions test*.

L'ensemble de variables de récurrence d'une clause ou terme se définit récursivement comme suit :

Définition 4.2.5 (variable de récurrence [Bouhoula, 1997]) *Soient R un système de réécriture et C un terme ou une clause. L'ensemble de variables de récurrence de C, noté par* $ind\_var(C)$*, est le plus petit ensemble tel que si x est une variable de sorte finitaire ou si elle apparaît dans un sous-terme non-variable t de C à la position u (t(u) = x) et il existe une règle* $\wedge_{i=1}^{n} l_i = r_i \Rightarrow g \rightarrow d$ *de R telle que g unifie avec t et :*

1. *u est une position non-variable de g, ou*
2. $g(u)$ *est une variable non-linéaire de g, ou*
3. $g(u) \in \cup_{i=1}^{n}\{ind\_var(l_i) \cup ind\_var(r_i)\}$,

alors x est une variable de récurrence *de C.*

Exemple 4.2.2 *Les variables de récurrence de la clause* $x \leq y = \text{True}$ *de l'exemple 4.2.1 sont x et y.*

Cette définition est plus générale que celle proposée dans [Bouhoula et Rusinowitch, 1995a] dans la mesure où elle permet de réfuter encore plus de conjectures, en particulier lorsque la spécification n'est pas suffisamment complète [Bouhoula, 1997].

Avant de définir la notion d'ensemble test, on doit introduire la notion de terme infinitaire.

Définition 4.2.6 (terme infinitaire) *Un terme t est* infinitaire *si pour toute position u de t pour laquelle t/u n'est pas clos, il existe un nombre infini d'instances closes R-irréductibles de t dont les sous-termes à la position u sont distincts.*

Définition 4.2.7 (ensemble test) *Un ensemble de termes TS est un* ensemble test d'un système de réécriture conditionnelle *R si TS est un ensemble couvrant de R satisfaisant la propriété suivante : toute instance d'un terme inductivement R-réductible par une substitution test est filtrée par la partie gauche d'une règle de R.*

La construction des ensembles test est illustrée par la proposition suivante. On introduit la notation $D(R)$ pour représenter la valeur $depth(R) - 1$ si R est linéaire gauche, sinon $depth(R)$.

Proposition 4.2.2 ([Bouhoula, 1997]) *Soit R un système de réécriture conditionnelle convergent sur les termes clos et suffisamment complet. Un ensemble test se calcule de la manière suivante :*

1. *$TS_1 = \{t \mid t$ est un terme constructeur linéaire de profondeur $\leq D(R)$ et $\forall p \in pos(t)$, $t(p) \in \mathsf{V}$ ssi $|p| = D(R)\}$.*
2. *$TS_2 = \bigcup_{t \in TS_1} expand(t)$, où $expand(t)$ est obtenu à partir de t par l'instanciation de ses variables de sorte finitaire s dans tous les cas possibles par des termes constructeurs clos de sorte s.*
3. *soit TS le sous-ensemble de TS_2 qui contient les termes qui ne sont pas inductivement R-réductibles.*
4. *si les termes de TS sont infinitaires, alors TS est un ensemble test de R.*

Exemple 4.2.3 *L'ensemble test calculé pour le système de réécriture R défini dans l'exemple 4.2.1 est* $\{0, s(x), \text{True}, \text{False}\}$.

Le calcul des ensembles test pour des spécifications équationnelles est décidable [Kapur *et al.*, 1987; Bündgen et Küchlin, 1989; Kounalis, 1992; Bündgen et Eckhardt, 1992; Hofbauer et Huber, 1994; Schmid et Fettig, 1995] et indécidable dans le cas général des spécifications conditionnelles. Pourtant, si le système de réécriture associé à la spécification conditionnelle est convergent sur les termes clos, suffisamment complet et les constructeurs ne sont spécifiés que par des équations inconditionnelles, leur calcul reste décidable [Bouhoula, 1997].

Maintenant, on introduit une propriété associée au système de réécriture, suffisante pour éviter les échecs de l'application de la règle Case Simplify : la *complétude forte*.

Définition 4.2.8 (complétude forte) *Soit $f \in \mathsf{F}$ un symbole de fonction suffisamment complet et R un système de réécriture déduit à partir des axiomes Ax. On dit que f est* fortement complet par rapport à *R si, pour toutes les règles $P_i \Rightarrow f(\overrightarrow{t_i}) \rightarrow d_i$ dont les membres gauches sont identiques à un renommage près μ_i, on a $Ax \models_{ini} \vee_i (P_i\mu_i)$. On dit aussi que R est un* système fortement complet *si tous les symboles de fonction de F sont fortement complets.*

Exemple 4.2.4 *Considérons la spécification de l'exemple 1.5.1 et la clause $min(x, y) = 0$. La règle* Case Simplify *est applicable au terme $min(x, y)$ tel que le résultat de son application est l'ensemble de clauses $\{x \leq y = \text{True} \Rightarrow x = 0,\ x \leq y = \text{False} \Rightarrow y = 0\}$. Maintenant, on remplace l'équation conditionnelle 1.4 par les équations conditionnelles 4.1 et 4.2. La nouvelle définition du symbole min devient :*

$$min(0, y) = 0 \tag{4.1}$$

$$s(x) \leq y = \text{True} \Rightarrow min(s(x), y) = s(x) \tag{4.2}$$

$$x \leq y = \text{False} \Rightarrow min(x, y) = y \tag{4.3}$$

Notons que la nouvelle spécification est aussi suffisamment complète. Pourtant, il y aura un échec sur l'application de la règle Case Simplify *à* $min(x, y) = 0$*, car la précondition* $x \leq y =$ False *n'est pas initialement valide.*

Proposition 4.2.3 *Soit R une spécification conditionnelle fortement complète et convergente sur les termes clos. Si C est une clause qui contient un terme inductivement R-réductible, alors au moins une des règles* Generate, Case Simplify *ou* Simplify *est applicable à C.*

Preuve Soit t le terme inductivement R-réductible contenu par la clause. Si t est R-réductible, alors on peut réduire t soit par Simplify si t est filtré par une règle inconditionnelle de R, soit par Case Simplify si elle est filtrée par une règle conditionnelle, car la spécification est fortement complète. D'une autre part, si t est R-irréductible mais inductivement R-réductible, soit σ une substitution test de C. Par la définition 4.2.7, $t\sigma$ est une instance de la partie gauche d'une règle de R, donc R-réductible comme dans le cas précédent. Par conséquent, on peut appliquer Generate à C.

Fin de preuve

Remarquons qu'un terme est inductivement R-réductible ssi il contient au moins un symbole de fonction définie et R est suffisamment complet.

Théorème 4.2.3 *Si R est fortement complet et convergent sur les termes clos, toute conjecture contenant un contre-exemple minimal d'une dérivation est une clause constructeur.*

Preuve Soit C une clause constructeur de l'ensemble courant de conjectures, contenant un contre-exemple minimal $C\tau$ d'une dérivation. On suppose, par absurde, que C est une conjecture contenant au moins un symbole de fonction définie. Selon la proposition 4.2.3, au moins une des règles Generate, Case Simplify ou Simplify est applicable à C. On va analyser ces règles, une par une. Notons que les éléments des contextes utilisés, qui sont inférieures (ou équivalentes) à $C\tau$, sont initialement valides. Ceci se montre de manière similaire comme dans la preuve du théorème 3.5.1.

- Si Generate s'applique à C, il existe un contre-exemple inférieure à $C\tau$ dans les nouvelles conjectures car l'ensemble couvrant engendré à l'étape (b') est strict. Contradiction.
- Si Simplify s'applique à C, alors on aboutit à une contradiction similaire.

Si Case Simplify s'applique à un terme $\preceq_t$-maximal t de C, l'ensemble couvrant engendré à l'étape (b') est aussi strict. Donc, contradiction. Si t n'est pas $\preceq_t$-maximal, l'ensemble couvrant engendré n'est pas forcément strict. Pourtant, il existe parmi les nouvelles conjectures une conjecture qui contient un contre-exemple équivalent à $C\tau$. Soit C' cette conjecture. Dans ce cas, C' doit contenir un autre terme, cette fois-ci $\preceq_t$-maximal, tel que $t \prec_t t'$. Comme t contient un symbole de fonction définie f_t et l'ordre sur les termes est $<_{MPO}$, il existe un symbole de fonction $f_{t'}$ satisfaisant $f_t <_{\mathsf{F}} f_{t'}$, conformément au lemme 1.4.1. D'une part, R est structuré, par la définition 1.5.9. Donc, il résulte que $f_{t'}$ est obligatoirement un symbole de fonction définie. D'autre part, l'application à l'infini de Case Simplify sur des termes qui ne sont pas $\preceq_t$-maximaux n'est pas possible. Ceci est dû au fait qu'à chaque application de la règle, on remplace dans la conjecture courante un terme par un ensemble fini de termes inférieurs à lui, qui ne sont pas $\preceq_t$-maximaux. Or, $\prec_t$ est bien fondé. Par conséquent, il existe une conjecture ultérieure dans la dérivation auquelle on ne peut plus appliquer aucune des règles Case Simplify, Simplify ou Generate, malgré l'existence d'un symbole de fonction définie. Contradiction avec la proposition 4.2.3.

Fin de preuve

Une procédure de décision pour la validité initiale des clauses constructeurs

Le système d'inférence composé par les règles structurelles de J', dénoté par J'_c, permet de décider sur la validité initiale d'une clause constructeur si les constructeurs sont libres.

Proposition 4.2.4 (terminaison de J'_c) *Soit C une clause constructeur. Si les constructeurs sont libres, toute J'_c-dérivation commençant par $(\{C\}, \emptyset)$ est finie.*

Preuve On va construire un ordre particulier sur les clauses constructeur comme étant une extension multiensemble de $<$, où $<$ est la relation « inférieur » sur les naturels.

La profondeur d'une équation, $|s = t|$, est la profondeur maximale des termes s et t. Formellement, $|s = t| = max(|s|, |t|)$. A partir de celle-ci, on définit la mesure d'une clause équationnelle $C \equiv \neg e_1 \vee \ldots \vee \neg e_n \vee e'_1 \vee \ldots \vee e'_m$ comme étant

$$|C| = \uplus_{i=1}^{n} |e_i| \uplus \uplus_{j=1}^{m} |e'_j|$$

Pour prouver la terminaison de toute J'_c-dérivation, il suffit de montrer que, par l'application d'une J'_c-règle, soit

1. il n'y a pas de nouvelles clauses engendrées, soit
2. les nouvelles clauses obtenues ont une mesure obtenue en remplaçant une des valeurs naturelles, contenues par la mesure de la clause constructeur courante, par un multiensemble (potentiellement vide) fini de valeurs inférieures à elle. Les nouvelles clauses seront aussi des clauses constructeurs.

On suppose que la règle appliquée a été

- une des règles Delete, Occur Check ou Negative Clash. Alors il n'y a pas de nouvelles clauses engendrées ;
- Positive Decomposition sur la clause $C \equiv f(\vec{s}) = f(\vec{t}) \vee r$. Soit m la valeur $|f(\vec{s}) = f(\vec{t})|$. Pour tout $i \in [1..n]$, la nouvelle clause engendrée $s_i = t_i \vee r$ a la mesure $(|C| \setminus\setminus \{m\}) \uplus \{|s_i = t_i|\}$. Celle-ci est inférieure à $|C|$, car $|s_i = t_i| < m$.
- Negative Decomposition sur la clause $C \equiv \neg(f(\vec{s}) = f(\vec{t})) \vee r$. Soit m la valeur $|f(\vec{s}) = f(\vec{t})|$. La nouvelle clause engendrée $\vee_{i=1}^{n} \neg(s_i = t_i) \vee r$ a la mesure $(|C| \setminus\setminus \{m\}) \uplus \{|s_1 = t_1|, \ldots, |s_n = t_n|\}$. Celle-ci est inférieure à $|C|$, car $|s_i = t_i| < m$, pour tout $i \in [1..n]$.
- Positive Clash sur la clause $C \equiv f(\vec{s}) = g(\vec{t}) \vee r$. Soit m la valeur $|f(\vec{s}) = g(\vec{t})|$. La nouvelle clause engendrée, r, a la mesure $|C| \setminus\setminus \{m\}$.
- Eliminate Trivial Equation sur la clause $C \equiv \neg(s = s) \vee r$. Soit m la valeur $|\neg(s = s)|$. La nouvelle clause engendrée, r, a la mesure $|C| \setminus\setminus \{m\}$.

Fin de preuve

Le prédicat d'échec `Fail` est vrai si l'ensemble de conjecture du dernier état de la J'_c-dérivation n'est pas vide. Alors il en existe une clause constructeur C sur laquelle aucune J'_c-règle n'est applicable. Si `Fail(C)` est vrai, C n'est pas une conséquence initiale des axiomes, selon le lemme suivant.

Lemme 4.2.2 *Soient C une clause constructeur, R un système de réécriture convergent sur les termes clos, basé sur des constructeurs libres. Les affirmations suivantes sont correctes :*

1. *Si `Fail`(C) est vrai alors $Ax \not\models_{ini} C$;*

2. *Si* $Ax \not\models_{ini} C$ *alors il existe une clause* C' *dans toute* J'*-dérivation commençant par* $(\{C\}, \emptyset)$ *telle que* $\texttt{Fail}(C')$ *est vrai.*

Preuve

1. On montre dans un premier temps que $Ax \not\models_{ini} C$ si $\texttt{Fail}(C)$ est vrai.
   Puisque les constructeurs sont libres et il n'y a pas de J'_c-règle applicable sur la clause constructeur C, C doit être forcement de la forme $\bigwedge_{i=1}^{n} x_i = t_i \Rightarrow \bigvee_{j=1}^{m} y_j = s_j$, satisfaisant pour tout $i \in [1..n]$, $x_i \in \mathsf{V}$, $t_i \in \mathcal{T}(\mathsf{CT}, \mathsf{V})$ et $x_i \not\equiv t_i$, et pour tout $j \in [1..m]$, $x_j \in \mathsf{V}$ et $s_j \in \mathcal{T}(\mathsf{CT}, \mathsf{V})$. De plus, $x_i \notin Var(t_i)$, sinon on aurait pu appliquer Occur Check sur C. Si on considère la substitution $\sigma = \{x_i \mapsto t_i | i \in [1..n]\}$, alors $C\sigma$ n'est pas une tautologie, sinon la règle Delete aurait été appliquée. Alors, on peut construire une substitution close τ telle que $C\sigma\tau$ n'est pas valide. Par exemple, pour chaque $j \in [1..n]$, on instancie les variables de $Var(s_j)$ par des termes constructeurs clos arbitraires et puis la variable y_j par un terme constructeur clos différent de l'instance close de s_j ainsi construite. Cette condition est suffisante car le système de réécriture est convergent sur les termes clos et les constructeurs sont libres.
   Comme $C\sigma$ n'est pas une conséquence initiale des axiomes, C ne l'est non plus.
2. Maintenant, on va montrer que si $Ax \not\models_{ini} C$, alors il existe une clause C' dans toute J'-dérivation commençant par $(\{C\}, \emptyset)$ telle que $\texttt{Fail}(C')$ est vrai. Comme sur C on applique seulement des règles de J'_c, par la proposition 4.2.4 et le théorème 4.2.1, toute J'-dérivation commençant par $(\{C\}, \emptyset)$ termine par un état ayant un ensemble non-vide de conjectures. Par conséquent, pour toute telle J'-dérivation, il doit exister une conjecture C' dans l'état final de la dérivation sur laquelle on ne peut plus appliquer aucune règle. Donc, $\texttt{Fail}(C')$ est forcement vrai.

Fin de preuve

Le théorème suivant est une conséquence du théorème 3.3.4.

Théorème 4.2.4 (complétude réfutationnelle de J') *Soient* E_0 *un ensemble de clauses et* R *un système de réécriture basé sur des constructeurs libres qui est convergent sur les termes clos et fortement complet sur les constructeurs. Si* $Ax \not\models_{ini} E_0$*, alors pour toutes les dérivations équitables* $(E^0, \emptyset) \vdash^{J'} \ldots$*, il existe un état auquel le prédicat* `Fail` *est appliqué.*

Preuve Comme $Ax \not\models_{ini} E_0$, il existe une clause $C_0 \in E_0$ et une substitution close σ telles que $Ax \not\models C_0\sigma$. Soit CE l'ensemble de tous les contre-exemples inférieurs ou équivalents à $C_0\sigma$ rencontrés pendant la dérivation, comme dans la preuve de la proposition 3.3.1, mais en considérant un l'ensemble initial des prémisses vide. L'ensemble CE n'est pas vide ; il contient au moins $C_0\sigma$. Il existe aussi un élément minimal de CE, noté par ψ, car $\leq$ est bien fondé. De plus, la dérivation est équitable, donc il existe une étape i lorsqu'une conjecture C contenant une instance close $C\tau$ équivalente à ψ est traitée.

Tautology ne peut pas s'appliquer à C car elle contient un contre-exemple. D'autre part, la subsomption de C par des instances des prémisses, des règles de réécriture ou des conjectures plus petites aboutira à une contradiction, comme dans la preuve de la proposition 3.3.1. Donc, C ne peut être subsumée que par des instances d'autres conjectures équivalentes à C. L'élimination de C par subsomption implique l'existence d'une autre conjecture ayant une instance équivalente à C. Comme le nombre de conjectures contenant des instances équivalentes à C décroît par l'application de Subsume, il existe un état de la dérivation où il y a une conjecture C' ayant une instance équivalente à C auquelle on ne peut plus appliquer Subsume.

Soit $C'\tau'$ le contre-exemple de C' équivalente à ψ. Selon la proposition 4.2.3, C' est une clause constructeur auquelle on peut appliquer seulement des règles de J'_c. La suite de J'_c-règles appliquées à C' et aux sous-conjectures dérivées à partir d'elle dans la J'-dérivation forment une J'_c-dérivation commençant par l'état $(\{C'\}, \emptyset)$. Cette dérivation va finir par `Fail`, selon le lemme 4.2.2, donc la J'-dérivation va finir aussi par `Fail`.

Fin de preuve

4.2.6 Améliorations de J'

Quelques améliorations du système d'inférence J', en tant qu'instance de A, peuvent être engendrées directement. Notons que ces optimisations n'affectent pas les propriétés de complétude et correction de J'.

Utilisation du contexte maximal admis par les A*-règles.* Quelques modules de raisonnement, comme IR et CS, dépendent du contenu du contexte lors de la génération des ensembles couvrants contextuels. Dans ces cas, le contexte peut être étendu au contexte maximal permis à l'étape correspondante des A-règles qu'elles instancient. Par exemple, le contexte de l'ensemble couvrant contextuel produit par IR à l'étape (b') de GENERATE, à savoir $(\emptyset, H \cup E \cup \{C\})$, peut s'étendre à $(H, E \cup \{C\})$, qui correspond au contexte maximal de l'étape (b') de ADDPREMISE.

Utilisation des ensembles couvrants contextuels, qui ne sont pas nécessairement stricts, dans des instances de A-SIMPLIFY. Par exemple, l'ensemble couvrant contextuel construit par IR à l'étape (b') de SIMPLIFY est strict. Comme ceci n'est pas imposé par la règle A-SIMPLIFY, la condition $C[a']_u \prec_c C[a]_u$ de la définition 4.2.1 peut être affaiblie à $C[a']_u \preceq_c C[a]_u$ afin de construire un ensemble couvrant contextuel plus général. Ceci enlèvera la restriction de réécrire seulement des sous-termes des termes $\preceq_t$-maximaux de la conjecture à traiter.

Extension des contextes maximaux par l'augmentation de l'ensemble de prémisses. Toute conjecture traitée par la règle SIMPLIFY peut être ajoutée à l'ensemble de prémisses. Notons que, dans ce cas, SIMPLIFY serait une instance de A-ADDPREMISE. Une remarque similaire pour la règle CASE SIMPLIFY si on emploie le module de raisonnement RCA' à la place de RCA.

Construction des ensembles couvrants contextuels plus puissants par la composition d'ensembles couvrants contextuels. Grâce aux propriétés de composition des ensembles couvrants contextuels, décrites en particulier par les propositions 2.3.3 et 2.3.5, il est possible de construire des ensembles couvrants contextuels plus complexes. Par exemple, on pourrait aussi employer la technique de subsomption à l'intérieur de la règle GENERATE. Pour cela, il suffit d'utiliser le module de raisonnement CS à l'étape (b') de GENERATE, pour engendrer des ensembles couvrants contextuels de $g_{CS}(C\sigma, (H, E \cup \{C\}))$. Dans ce cas, $\cup_\sigma E_\sigma$ serait un ensemble couvrant contextuel construit à l'aide des modules de raisonnement RCA', IR, T et CS.

Extension modulaire et incrémentale du système d'inférence par de nouvelles techniques de raisonnement. On suppose qu'on veut incorporer de nouvelles techniques de raisonnement dans le démonstrateur. Des modules de raisonnement élémentaires peuvent être conçus pour mettre en œuvre ces techniques et/ou de nouveaux modules de raisonnement peuvent être créés par leur composition avec ceux déjà existants. Le type de l'ensemble couvrant contextuel qu'ils

construisent et son contexte maximal admis à une étape donnée d'une règle d'inférence doivent être compatibles avec ceux correspondant à la A-règle instanciée par la règle d'inférence. Des exemples sont donnés dans la section 5.2.

Utilisation des prémisses et des conjectures de l'état courant comme des prémisses dans les dérivations secondaires. Notons que les conditions des modules de raisonnement RCA et RCA' sont inférieures à la conjecture C de l'étape courante ($E \cup \{C\}$, H), selon la définition 4.2.2. Par la définition 3.5.2, l'ensemble initial de prémisses de la dérivation secondaire, nécessaire pour vérifier les conditions, est autorisé à contenir les deux composantes du contexte courant. Comme le tableau 4.2.4 le montre, RCA' et RCA sont utilisés respectivement à l'étape (b') de GENERATE et CASE SIMPLIFY. Sachant que GENERATE instancie A-ADDPREMISE, et CASE SIMPLIFY la règle A-SIMPLIFY, en analysant les contextes maximaux appropriés dans les tableaux 3.1 et 3.2, on observe que l'ensemble initial de prémisses dans la dérivation secondaire peut être $H \cup E \cup \{C\}$.

4.3 Conclusions

Dans ce chapitre, nous avons obtenu comme résultat majeur la description d'un système d'inférence de SPIKE, présentée dans [Bouhoula, 1997], en tant qu'instance du système abstrait d'inférence A($\mathcal{RM}$) que nous avons proposé dans le chapitre 3.

A part une nouvelle description modulaire et uniforme de SPIKE vis-à-vis des techniques de raisonnement employés, une utilisation d'un ordre sur les clauses plus général et une solution proposée pour vérifier les conditions d'application de la règle CASE SIMPLIFY par des appels récursifs au prouveur, ce résultat lui apporte les améliorations suivantes :

- utilisation du contexte maximal admis par les A-règles ;
- utilisation des ensembles couvrants contextuels, qui ne sont pas nécessairement stricts, dans des instances de A-SIMPLIFY ;
- extension des contextes maximaux par l'augmentation de l'ensemble de prémisses ;
- construction des ensembles couvrants contextuels plus puissants par la composition d'ensembles couvrants contextuels ;
- extension modulaire et incrémentale du système d'inférence par de nouvelles techniques de réécriture ;
- utilisation des prémisses et des conjectures de l'état courant comme des prémisses dans les dérivations secondaires.

5

Extensions et améliorations de SPIKE

Sommaire

5.1 Introduction

Différentes procédures de preuve basées sur la récurrence implicite [Gramlich, 1989; Bevers et Lewi, 1990; Bouhoula et Rusinowitch, 1995a; Naidich, 1996] utilisent intensivement (des variantes de) la technique de subsomption clausale afin d'éliminer des conjectures redondantes. Son utilisation est essentielle pour éviter la divergence des dérivations. Dans la section 5.2, nous proposons dans un premier temps une nouvelle technique de subsomption, adaptée au raisonnement inductif. Puis, on montre comment une extension de SPIKE avec de nouvelles règles d'inférence, basées sur cette technique et sur des procédures de décision de l'arithmétique linéaire, est utilisée pour montrer la correction de l'algorithme MJRTY [Boyer et Moore, 1991] par une combinaison de raisonnement inductif et arithmétique.

Une amélioration importante apportée à une preuve est la réduction de son temps d'exécution par l'identification de ses parties indépendantes qui peuvent ainsi s'exécuter en même temps. Dans la section 5.3, nous proposons une interface parallèle pour le système d'inférence de SPIKE. Elle met en œuvre un schéma de parallélisation au niveau des conjectures qui principalement distribue les conjectures courantes aux processeurs, crée un processus SPIKE pour traiter chacune d'entre elles, analyse leurs traces de preuves et interprète leurs résultats.

5.2 Extension de SPIKE par la technique de subsomption sémantique inductive

Il existe plusieurs définitions de la subsomption clausale. Une des plus générales est la *subsomption clausale sémantique*. On dit qu'une clause C_1 *subsume sémantiquement* une autre clause C_2 s'il existe une substitution σ telle que $Ax \models_{ini} C_1\sigma \Rightarrow C_2$, où Ax est l'ensemble d'axiomes. Du point de vue théorique, trouver la « bonne » substitution qui satisfait l'implication logique est un problème indécidable. En pratique, la substitution est calculée par des techniques syntaxiques (décidables) comme le filtrage. Toute opération de subsomption basée sur des critères syntaxiques est généralement appelée θ-subsomption [Loveland, 1978]. Un exemple de θ-subsomption est la subsomption clausale de la définition 4.2.4. D'autres définitions de θ-subsomption se trouvent dans [Rusinowitch, 1989; Naidich, 1996].

Définition 5.2.1 (θ-subsomption sur des clauses équationnelles [Naidich, 1996]) *Une clause équationnelle C est* θ-subsumée *par une autre clause équationnelle C' s'il existe une substitution θ telle que i) pour toute équation $a = b$ de C' il existe un terme t satisfaisant la relation $t[a\theta] = t[b\theta] \in C$, et ii) pour toute $\neg(a = b) \in C'$, on a $\neg(a\theta = b\theta) \in C$.*

Notons que l'instance de la clause qui subsume est toujours inférieure ou équivalente (par rapport à $\preceq_c$) à la clause subsumée.

Exemple 5.2.1 *Considérons l'ensemble d'axiomes de l'exemple 2.3.3 et deux clauses équationnelles :*

$$P(s(x),\, y) = True \;\vee\; 0 \leq 0 = True \tag{5.1}$$

$$x_1 \leq x_2 = True \;\vee\; 0 \leq 0 = True \tag{5.2}$$

La clause (5.1) est θ-subsumée par la clause (5.2) instanciée avec la substitution $\{x_1 \leftarrow 0,\, x_2 \leftarrow 0\}$. Pourtant, si on utilise la définition 4.2.4 de subsomption clausale, la relation entre les deux clauses, tout en appliquant la même substitution, disparaît parce que $0 \leq 0 = True \vee 0 \leq 0 = True$ n'est pas une sous-clause de $P(s(x),\, y) = True \;\vee\; 0 \leq 0 = True$.

On peut simplement imaginer des situations où on a besoin que la clause qui subsume soit strictement inférieure.† Naidich propose une relation de θ-subsomption qui partage l'ensemble d'instances des clauses subsumées en deux sous-ensembles, W et V, contenant respectivement des instances inférieures ou équivalentes et strictement inférieures.

Définition 5.2.2 (θ-subsomption inductive [Naidich, 1996]) *Etant donnée une clause C et deux ensembles de clauses V et W, soit C' une clause de $W \cup V$ telle que C est θ-subsumée par C' avec la substitution θ. Alors, on écrit $C \supseteq_{W\cup V} C'$. On écrit aussi $C \supset_{W[V]} C'$ si soit $C' \in W$, soit $C' \in V$ et $C'\theta \not\preceq_c C$.*

Notons dans cette définition que l'instance de la clause qui subsume reste toujours inférieure ou équivalente à la clause subsumée. Pourtant, cette condition n'est pas nécessaire si la clause subsumée est une instance d'un axiome car il est toujours correct d'utiliser n'importe quelle

† Par exemple, si elle est une conjecture utilisée à l'étape (b') d'une règle A-AddPremise où la θ-subsomption est utilisée.

GENERATE : $(E \cup \{C\},\ H) \vdash^{J'} (E \cup (\cup_\sigma E_\sigma),\ H \cup \{C\})$
si pour toute substitution couvrante $\sigma \in CS\Sigma(C)$,
- soit $C\sigma$ est une tautologie et $E_\sigma = \emptyset$,
- soit $C\sigma \rightarrow_{R\langle H \cup E \cup \{C\}\rangle} C'$ et $E_\sigma = \{C'\}$,
- soit $E_\sigma = RecursiveCaseAnalysis'(C\sigma)$,
- sinon, il existe C' telle que $C\sigma \supset^s_{Ax[H]E\cup\{C\}} C'$ et $E_\sigma = \{C'\}$

FIG. 5.1 – *La nouvelle règle* GENERATE

instance d'un axiome. Cette restriction peut être levée en généralisant la θ-subsomption inductive à une variante de subsomption sémantique adaptée au raisonnement inductif.

Définition 5.2.3 (subsomption sémantique inductive) *Etant donnés une clause C et trois ensembles de clauses T, W et V. On suppose que C' est une clause et σ une substitution telles que $Ax \models_{ind} [C'\sigma \Rightarrow C]^{\mathtt{cnf}}$. Alors, on écrit*

$$C \supseteq^s_{T[W]} C'$$

si soit i) $C' \in T$, soit ii) $C' \in W$ et $C'\sigma \preceq_c C$. On écrit

$$C \supset^s_{T[W]V} C'$$

si soit i) $C' \in T$, soit ii) $C' \in W$ et $C'\sigma \preceq_c C$, soit iii) $C' \in V$ et $C'\sigma \prec_c C$.

Théorème 5.2.1 *La notion de subsomption sémantique inductive est une généralisation de la θ-subsomption inductive.*

Preuve Soient C, C' deux clauses et V, W deux ensembles de clauses tels que $C \supseteq_{W\cup V} C'$. Comme C est θ-subsumée par C' avec la substitution θ, il résulte que $C'\theta \preceq_c C$ et $Ax \models_{ind} [C'\theta \Rightarrow C]^{\mathtt{cnf}}$. Par conséquent, $C \supseteq^s_{\emptyset[W\cup V]} C'$.

D'autre part, on suppose que $C \supset_{W[V]} C'$. De manière similaire, $Ax \models_{ind} [C'\theta \Rightarrow C]^{\mathtt{cnf}}$ et soit i) $C'\theta \preceq_c C$, si $C' \in W$, soit ii) $C'\theta \prec_c C$, si $C' \in V$, parce que les conditions $C'\theta \not\sim_c C$ et $C'\theta \prec_c C$ sont équivalentes sous les hypothèses courantes. Par conséquent, $C \supset^s_{\emptyset[W]V} C'$.

Fin de preuve

L'ensemble $\mathcal{RM}$ de modules de raisonnement associé au système d'inférence J' peut s'étendre par de nouveaux modules de raisonnement basés sur la subsomption sémantique inductive. Pour cela, on va définir un module de raisonnement conditionnel ISS^s qui met en œuvre la relation $\supset^s$, telle que : i) sa fonction de génération retourne un ensemble couvrant contextuel vide dans le contexte $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$, et ii) sa fonction de condition est $c_{ISS^s}(C, Cxt) = [C'\theta \Rightarrow C]^{\mathtt{cnf}}$, où θ est une substitution, et soit a) $C' \in Ax$, soit b) $C' \in \mathsf{C}^1$ tel que $C'\theta \preceq_c C$, soit c) $C' \in \mathsf{C}^2$ tel que $C'\theta \prec_c C$. Une application pratique de ISS^s est la génération des ensembles couvrants contextuels vides de l'étape (b') de GENERATE† lorsque $Ax \models_{ind} c_{ISS^s}(C\sigma, (H, E \cup \{C\}))$ à l'étape $(E \cup \{C\}, H)$ d'une J'-dérivation. La règle GENERATE devient (voir la figure 5.1) :

De la même façon, on peut définir un module de raisonnement ISS qui implante la relation $\supseteq^s$: i) sa fonction de génération retourne un ensemble couvrant contextuel vide dans le contexte

† Voir la figure 4.1.

Cxt caractérisé par (C^1, C^2), et ii) sa fonction de condition est $c_{ISS}(C, Cxt) = [C'\theta \Rightarrow C]^{\mathtt{cnf}}$, où θ est une substitution, et soit a) $C' \in Ax$, soit b) $C' \in \mathsf{C}^1$ tel que $C'\theta \preceq_c C$. Il peut être utilisé dans de nouvelles règles d'inférence qui instancient DELETE, par exemple pour définir la règle INDUCTIVE SEMANTIC SUBSUMPTION :

INDUCTIVE SEMANTIC SUBSUMPTION : $(E \cup \{C\}, H) \vdash^{J'} (E, H)$ si ISS construit un ensemble couvrant contextuel vide dans le contexte $(C, (E \cup H, \emptyset))$

La règle INDUCTIVE SEMANTIC SUBSUMPTION, en tant qu'instance de A-DELETE, a $\{C\}$ comme ensemble couvrant contextuel demandé à l'étape (a), et l'ensemble couvrant contextuel vide engendré par ISS, à l'étape (b').

L'extension ainsi proposée du système d'inférence J' est une instance correcte et réfutationnellement correcte du $\mathsf{A}(\mathcal{RM})$.

5.2.1 Un exemple : la preuve de correction de l'algorithme MJRTY

La version étendue du système d'inférence J', présentée dans la section 5.2, a été utilisée pour montrer la correction de l'algorithme MJRTY [Boyer et Moore, 1991]. MJRTY calcule d'une manière efficace l'élément *majoritaire* (s'il en existe un) d'un multiensemble, c'est-à-dire l'élément qui y apparaît un nombre de fois supérieur à la moitié de la taille du multiensemble. L'algorithme vérifie les éléments en temps réel (c'est-à-dire qu'il n'y a pas de stockage d'éléments afin d'exécuter des opérations ultérieures) et élimine la phase de comptage spécifique aux algorithmes triviaux similaires lorsque l'existence de l'élément majoritaire est supposée. Sinon, un deuxième parcours est nécessaire afin de certifier que l'élément calculé est vraiment majoritaire. Un autre avantage de l'algorithme est sa complexité linéaire en temps par rapport à la cardinalité du multiensemble. Comme application typique de MJRTY, on peut citer la désignation du candidat majoritaire parmi les candidats inscrits à des élections.

MJRTY a été inventé et prouvé correct avec le démonstrateur automatique NQTHM [Boyer et Moore, 1979] en 1980 par Boyer et Moore, mais publié seulement onze ans plus tard dans [Boyer et Moore, 1991]. Codé dans le langage de programmation Fortran, l'algorithme dispose d'une preuve de correction assez difficile qui demande l'utilisation de cinq lemmes pour vérifier les 61 conditions de preuve engendrées par un outil de génération des conditions de preuve à partir des programmes en Fortran. A part NQTHM, plusieurs démonstrateurs de preuve interactifs ont réussi ultérieurement à prouver sa correction, par exemple PVS [Owre *et al.*, 1992] (selon [Howe, 1993]), Nuprl [Jackson, 1994] dans [Howe, 1993], et STeP [Bjørner et others, 1995] dans [Bjørner, 1998].

L'idée de l'algorithme est de construire des paires d'éléments et d'éliminer les paires entre des éléments différents de manière à ce que l'élément retourné à la fin du processus d'élimination soit l'élément potentiellement majoritaire. MJRTY peut être facilement converti d'un code écrit dans un langage impératif, comme Fortran, à une fonction récursive. Cette fonction retourne une paire *(mcv, mlv)*, où *mcv* est le candidat majoritaire et *ml* son avance par rapport aux autres candidats sachant que les votes se trouvent dans une urne p à i votes donnés comme arguments à l'entrée (voir l'algorithme 1 qui la met en œuvre).

La spécification de MJRTY pour SPIKE est présentée dans la figure 5.2. Elle comprend quatre parties principales. Les sortes sont déclarées dans la première partie : `nat` représente les naturels, `bool` les booléens, `list` les listes de candidats et `cand` les candidats. La deuxième partie contient la déclaration des symboles de fonction. Dans une première étape, on déclare les

Algorithme 1 m(p, i) : l'algorithme MJRTY

Entrée : une urne p de taille i

Sortie : le candidat majoritaire et son avance par rapport aux autres candidats

```
1: si i > 0 alors
2:    (mcv, mlv) ← m(p, i - 1)
3:    si p[i] = mcv alors
4:       retourne (mcv, mlv + 1)
5:    sinon si mlv > 0 alors
6:       retourne (mcv, mlv - 1)
7:    sinon
8:       retourne (p[i], 1)
9:    fin si
10: sinon
11:    (Noname, 1)
12: fin si
```

symboles constructeurs `0` et `s` pour les naturels, `True` et `False` pour les booléens, `Nil` et `Cons` pour les listes de candidats et, enfin, `c` et `Noname` pour les candidats. `Noname` est considéré comme un candidat spécial qui est retourné lorsqu'il n'y a pas de candidat majoritaire. Puis, on déclare les symboles de fonctions définies. La fonction m a été divisée en deux fonctions mutuellement récursives, `mc` et `ml`, qui calculent respectivement le candidat majoritaire et son avance par rapport aux autres candidats. `count(p, i, a)` compte le nombre de votes pour un candidat donné `a` d'une urne `p` contenant `i` votes. Les autres fonctions définies sont : i) `access(p, n)`, qui retourne le `n`-ième élément d'une liste `p`, ii) `eqc` qui est l'opérateur d'égalité associé à la sorte `cand`, iii) la fonction conditionnelle `if` à quatre arguments, iv) la soustraction Peano par 1, notée par `minus1`, v) l'opérateur « inférieur » $<$ sur les naturels, et vi) l'opérateur d'addition $+$. La troisième partie contient les définitions axiomatiques de chaque symbole de fonction définie. Dans la dernière partie, on établit les précédences sur les symboles de fonction qui seront utilisées pour définir le pré-ordre $\preceq_c$ sur les clauses, selon les définitions 1.4.7 et 1.4.8.

La conjecture principale spécifie que $mc(p,\, i)$ retourne toujours le candidat majoritaire lorsqu'il existe un tel candidat dans l'urne p à i votes[†] :

$$\forall p : \texttt{list},\, \forall i : \texttt{nat},\, \forall a : \texttt{cand},\; i < 2 * count(p,\, i,\, a) = True \Rightarrow a = mc(p,\, i)$$

Un lemme important, qui simplifie beaucoup la preuve de la conjecture principale, est un invariant proposé par N. Shankar (selon [Howe, 1993]) :

$$2 * (if(x_1,\, mc(x_2,\, x_3),\, 0,\, ml(x_2,\, x_3)) + count(x_2,\, x_3,\, x_1)) < s(x_3 + ml(x_2,\, x_3)) = True$$

La preuve de l'invariant n'est pas facile. Avec SPIKE, elle demande l'utilisation de la subsomption sémantique inductive dans 17 de ses sous-buts non-triviaux. On va analyser dans la

† Pour une meilleure présentation, $2 * x$ va dénoter $x + x$.

suite un de ces sous-buts. Soit C_1 la conjecture normalisée suivante :

$$eqc(c(j),\, access(p,i)) = True \;\vee\; eqc(access(p,\, i),\, mc(p,\, i)) = True \;\vee\; 0 < ml(p,\, i) = False \vee$$
$$eqc(c(j),\, mc(p,\, i)) = True \vee 2 * (minus1(ml(p,\, i)) + count(p,\, i,\, c(j))) < s(s(i + minus1(ml(p,\, i)))) = True$$

et C_2 la conjecture dérivée de l'invariant par l'application de la règle Case Analysis qui élimine le symbole if du terme $if(x_1,\, mc(x_2,\, x_3),\, 0,\, ml(x_2,\, x_3))$:

$$eqc(x_1,\, mc(x_2,\, x_3)) = True \vee 2 * (ml(x_2,\, x_3) + count(x_2,\, x_3,\, x_1)) < s(x_3 + ml(x_2,\, x_3)) = True$$

La conjecture C_1 est un des sous-buts obtenus après l'application de Generate sur C_2, par l'instanciation des variables de récurrence en utilisant la substitution couvrante $\theta = \{x_1 \leftarrow c(j),\, x_2 \leftarrow p,\, x_3 \leftarrow s(i)\}$. On va montrer que C_1 est subsumée par C_2 en utilisant la règle Inductive Semantic Subsumption, définie dans la section 5.2.

Soit $\sigma = \{x_1 \leftarrow c(j),\, x_2 \leftarrow p,\, x_3 \leftarrow i\}$. Notons que la condition $C_2\sigma \preceq_c C_1$ est accomplie. D'autre part, la condition d'applicabilité du module de raisonnement ISS, à savoir $Ax \models_{ind} [C_2\sigma \Rightarrow C_1]^{\mathtt{cnf}}$, revient à montrer que les deux clauses suivantes, obtenues par la normalisation de $C_2\sigma \Rightarrow C_1$ en forme normale conjonctive, sont des conséquences initiales des axiomes :

1. $2 * (ml(p,\, i) + count(p,\, i,\, c(j))) < s(i + ml(p,\, i)) = True \Rightarrow eqc(c(j),\, access(p,\, i)) = True \vee eqc(access(p,\, i),\, mc(p,\, i)) = True \vee 0 < ml(p,\, i) = False \vee eqc(c(j),\, mc(p,\, i)) = True \vee 2 * (minus1(ml(p,\, i)) + count(p,\, i,\, c(j))) < s(s(i + minus1(ml(p,\, i)))) = True$, et
2. $eqc(c(j),\, mc(p,\, i)) = True \Rightarrow eqc(c(j),\, access(p,\, i)) = True \vee eqc(access(p,\, i),\, mc(p,\, i)) = True \vee 0 < ml(p,\, i) = False \vee eqc(c(j),\, mc(p,\, i)) = True \vee 2 * (minus1(ml(p,\, i)) + count(p,\, i,\, c(j))) < s(s(i + minus1(ml(p,\, i)))) = True$,

Elles ont été vérifiées par des appels récursifs au prouveur. La deuxième conjecture est triviale car $eqc(c(j),\, mc(p,\, i)) = True$ apparaît des deux côtés de $\Rightarrow$. La première conjecture est prouvée correcte par l'utilisation du raisonnement arithmétique. On abstrait les sous-termes non-arithmétiques suivants avec des variables : $ml(p,\, i)$ avec x et $count(p,\, i,\, c(j))$ avec y, pour obtenir

$$\underline{2 * (x + y) < s(i + x) = True} \Rightarrow$$
$$eqc(c(j),\, access(p,\, i)) = True,\; eqc(access(p,\, i),\, mc(p,\, i)) = True,\; \underline{0 < x = False},$$
$$eqc(c(j),\, mc(p,\, i)) = True,\; \underline{2 * (minus1(x) + y) < s(s(i + minus1(x))) = True}$$

Les formules atomiques soulignées sont identifiées comme arithmétiques. Une solution pour prouver leur correction est d'appliquer une procédure de décision pour l'arithmétique linéaire. Dans l'exemple 1.6.1, on a montré comment prouver la clause C' :

$$2 * (x + y) < (i + x + 1) = True \Rightarrow 2 * (x - 1 + y) < (1 + i + x) = True$$

par raisonnement arithmétique. Pour cela, nous introduisons la règle d'inférence suivante, de type A-Delete, basée sur le module de raisonnement PA défini dans l'exemple 3.5.3. Elle est définie de manière similaire à Inductive Semantic Subsumption.

LINEAR ARITHMETIC : $(E \cup \{C\},\, H) \vdash^{J'} (E,\, H)$ si PA construit un ensemble couvrant contextuel vide dans le contexte $(C,\, (E \cup H,\, \emptyset))$

On conclut que l'opération de subsomption sémantique inductive est applicable afin de montrer la validité initiale de C_1.

Une autre solution est de simuler le raisonnement arithmétique par l'utilisation des lemmes arithmétiques. Pour l'exemple précédent, un tel lemme peut être

$$2 * (u + v) < s(z + u) = False,\ 2 * (minus1(u) + v) < s(s(z + minus1(u))) = True$$

Les autres sous-buts non-triviaux issus de l'invariant sont prouvés de manière similaire.

5.2.2 Heuristiques pour le choix des substitutions

Malgré l'indécidabilité de la relation de subsomption sémantique inductive, nous avons conçu une heuristique qui propose des substitutions lorsque les clauses qui subsument sont des prémisses. Cette heuristique a été utilisée avec succès dans la preuve de correction de l'algorithme MJRTY, présenté dans la section 5.2.1.

Selon la définition 5.2.3, si H est un ensemble de prémisses, $C \in H$ une prémisse et C_1 une conjecture de l'état courant de la dérivation $(E \cup \{C_1\},\, H)$, on a $C_1 \supseteq^{s}_{Ax[H]} C$ si σ est une substitution telle que i) (relation d'ordre) $C\sigma \preceq_c C_1$ et ii) (relation de conséquence initiale) $Ax \models_{ind} [C\sigma \Rightarrow C_1]^{\mathtt{cnf}}$. L'idée principale de l'heuristique est de réduire le choix des substitutions à celles qui vérifient la contrainte introduite par la relation d'ordre. La construction de ces substitutions se fait à partir de la trace de la preuve, en analysant les rapports d'héritage entre C et C_1. Pour cela, on va associer à chaque clause son *historique*, qui permettra d'identifier les étapes importantes pendant la transformation des clauses, commençant par les clauses initiales et jusqu'à sa génération. L'historique d'une clause C_1, notée par $hist(C_1)$, est une liste de paires de la forme (*clause*, *substitution*) qu'on calcule au moment de la création de C_1. On va noter par [] et @, respectivement, la *liste vide* et l'*opération de concaténation* de deux listes. L'historique d'une conjecture se calcule récursivement comme suit :

Définition 5.2.4 (historique d'une clause, parent direct d'une clause) *On considère* $(E \cup \{C\},\, H) \vdash^{\mathtt{J}'} (E \cup \overline{C},\, H)$ *une étape arbitraire d'une* J'*-dérivation. Alors*

$$hist(C_1) = \begin{cases} [] & \text{si } (E \cup \{C\},\, H) \text{ est l'état initial de la dérivation} \\ & \text{et } C_1 \in E \cup \{C\}, \\ hist(C) & \text{si la } J'\text{-règle appliquée à cette étape n'est pas GENERATE et} \\ & C_1 \in \overline{C}, \\ hist(C)@[(C,\, \sigma)] & \text{si GENERATE a été appliquée à } C \text{ et } C_1 \in \overline{C} \text{ est une clause} \\ & \text{obtenue en simplifiant l'instance } C\sigma, \\ & \text{où } \sigma \text{ est une substitution couvrante de } CS\Sigma(C). \end{cases}$$

Dans le dernier cas, on appelle C *le* parent direct *de* C_1.

Lemme 5.2.1 *Dans la définition 5.2.4, les clauses qui apparaissent dans* $hist(C_1)$ *font partie de l'ensemble courant de prémisses* H.

```
specification : MJRTY

sorts : nat, bool, list, cand ;

constructors :
0 : -> nat ;
s_ : nat ->nat ;
True : ->bool ;
False : ->bool ;
Nil : ->list ;
Cons__ : cand list ->list ;
c_ :cand ->cand ;
Noname : ->cand ;

defined functions :

mc__ : list nat ->cand ;
ml__ : list nat ->nat ;
count___ : list nat cand ->nat ;
access__ : list nat ->cand ;
eqc__ : cand cand ->bool ;
if____ : cand cand nat nat ->nat ;
minus1_ : nat ->nat ;
_<_ : nat nat ->bool ;
_+_ : nat nat ->nat ;

axioms :

% mc calcule le candidat majoritaire d'une liste de candidats
eqc(access(p, i), mc(p, i))=True =>mc(p, s(i))=mc(p, i) ;
eqc(access(p, i), mc(p, i))=False, 0<ml(p, i)=True =>mc(p, s(i))=mc(p, i) ;
eqc(access(p, i), mc(p, i))=False, 0<ml(p, i)=False =>mc(p, s(i))=access(p, i) ;
mc(p, 0)=Noname ;

% ml retourne l'avance du candidat majoritaire par rapport aux autres candidats
eqc(access(p, i), mc(p, i))=True =>ml(p, s(i))=s(ml(p, i)) ;
eqc(access(p, i), mc(p, i))=False, 0<ml(p, i)=True =>ml(p, s(i))=minus1(ml(p, i)) ;
eqc(access(p, i), mc(p, i))=False, 0<ml(p, i)=False =>ml(p, s(i))=s(0) ;
ml(p, 0)=s(0) ;

% count calcule le nombre d'occurrences d'un élément dans une liste
eqc(a, access(p, i))=True =>count(p, s(i), a)=s(count(p, i, a)) ;
eqc(a, access(p, i))=False =>count(p, s(i), a)=count(p, i, a) ;
count(p, 0, y)= 0 ;

% access retourne le n-ième élément d'une liste
access(Nil, x)=Noname ;
access(Cons(x, l), 0)=x ;
access(Cons(x, l), s(y))=access(l, y) ;

% eqc définit l'opérateur de l'égalité de la sorte cand
eqc(Noname, Noname)=True ;
eqc(Noname, c(x))=False ;
eqc(c(x), Noname)=False ;
eqc(c(x), c(y))=eqc(x, y) ;

% la définition de if à quatre arguments
eqc(a, b)=True =>if(a, b, x, y)=x ;
eqc(a, b)=False =>if(a, b, x, y)=y ;

% minus1 définit la soustraction Peano par 1
minus1(s(x))=x ;
minus1(0)=0 ;

% la définition de l'opérateur ‘‘plus petit’’sur les naturels
x<0=False ;
0<s(x)=True ;
s(x)<s(y)=x<y ;
```


FIG. 5.2 – *La spécification de MJRTY*

```
% l'addition
0+x=x ;
s(x)+y=s(x+y) ;
% la précédence sur les symboles de fonction

less :
0 s True False Noname Nil Cons c minus1 eqc if + access < ml count ;

equiv :
mc ml ;
```

FIG. 5.3 – *La spécification de MJRTY (suite)*

Preuve Par la construction de $hist(C_1)$. Sur les clauses qui apparaissent dans $hist(C_1)$ on a appliqué GENERATE.

Fin de preuve

On dit qu'une clause C est un *parent d'une clause* C_1 s'il existe une paire (C_2, σ) dans $hist(C_1)$ telle que $C \equiv C_2$. Notons qu'un parent direct est aussi un parent. La notion de *substitution cumulative* entre C et C_1 représente la composition de gauche à droite de toutes les substitutions de l'historique de C_1 enregistrées à partir de C.

Définition 5.2.5 (substitution cumulative entre deux clauses) *Soit C et C_1 deux clauses telles que C est un parent de C_1 et la dernière paire enregistrée dans $hist(C_1)$ est (C_2, σ). La* substitution cumulative $\theta_{C \frown C_1}$ *entre C et C_1 est définie récursivement comme suit :*

1. *si $C \equiv C_2$, alors $\theta_{C \frown C_1} = \sigma$.*
2. *sinon, si $\theta_{C \frown C_2}$ est la substitution cumulative entre C et la clause C_2, alors $\theta_{C \frown C_1} = (\theta_{C \frown C_2})\sigma$.*

Exemple 5.2.2 *Si $hist(C) = [(C_4, \sigma_4); (C_3, \sigma_3); (C_2, \sigma_2); (C_1, \sigma_1)]$, alors la substitution cumulative $\theta_{C_3 \frown C}$ entre C_3 et C est la substitution composée $\sigma_3\sigma_2\sigma_1$.*

Lemme 5.2.2 *Soient C et C_1 deux clauses et $\theta_{C \frown C_1}$ la substitution cumulative entre C et C_1. Alors on peut construire une J'-dérivation qui ne contient pas d'applications de* GENERATE *et qui simplifie $C\theta_{C \frown C_1}$ en C_1.*

Preuve Par construction, à partir de la (sous-)dérivation qui transforme C en C_1. Considérons qu'à une étape de cette dérivation on a appliqué la règle GENERATE à la clause C_2 d'une paire (C_2, σ) de $hist(C_1)$ qui n'est pas dans $hist(C)$. A cette étape, à la place de GENERATE on pourra forcément appliquer CASE SIMPLIFY (resp. SIMPLIFY) si l'opération de simplification de l'instance $C_2\sigma$ a été *RecursiveCaseAnalysis'* (resp. la réécriture inductive). Les étapes de la dérivation qui impliquent le traitement des instances de C_2 avec d'autres substitutions couvrantes seront ignorées.

Fin de preuve

Le parent d'une clause, instancié avec la substitution cumulative entre lui et la clause, est supérieur ou équivalent à la clause, selon la proposition suivante.

Proposition 5.2.1 *Soient C et C_1 deux clauses et $\theta_{C \frown C_1}$ la substitution cumulative entre C et C_1. Alors $C_1 \preceq_c C\theta_{C \frown C_1}$.*

Preuve Selon le lemme 5.2.2, on peut construire une J-dérivation qui transforme $C\theta_{C \curvearrowright C_1}$ en C_1 sans utiliser GENERATE. Selon le lemme 4.2.1, chaque J'-règle qui est différente de GENERATE et qui produit un ensemble non-vide de clauses $\overline{C_3}$ une fois appliquée sur une clause C_2, respecte la relation $C_3 \preceq_c C_2$, pour toute clause $C_3 \in \overline{C_3}$. Par la propriété de transitivité de $\preceq_c$, on obtient $C_1 \preceq_c C\theta_{C \curvearrowright C_1}$.

Fin de preuve

Pour appliquer la règle de subsomption sémantique inductive à une clause C_1 en utilisant une prémisse C de l'historique de C_1, il faut trouver une substitution σ telle que $C\sigma \preceq_c C_1$. Si $\theta_{C \curvearrowright C_1}$ est la substitution cumulative entre C et C_1, alors $C_1 \preceq_c C\theta_{C \curvearrowright C_1}$, selon la proposition 5.2.1. Donc, la substitution candidat σ doit aussi satisfaire la condition (nécessaire, mais pas suffisante)

$$C\sigma \preceq_c C\theta_{C \curvearrowright C_1}$$

A partir des substitutions cumulatives, on va construire des substitutions candidates qui vont respecter cette condition.

Définition 5.2.6 (application de type récursif) *Une application d'une variable dans un terme, de la forme $x \mapsto t$, est de* type récursif *si le terme non-variable t contient au moins une variable de même sorte que la variable x.*

Exemple 5.2.3 *L'application $x \mapsto s(x')$ est de type récursif car les variables x et x' sont de sorte* `nat`*.*

Les substitutions couvrantes peuvent contenir des applications de type récursif lorsque les variables du domaine d'application sont de sorte infinitaire. L'image de ces substitutions est formée par des éléments de l'ensemble couvrant $CS(R)$ associés au système de réécriture R, souvent des termes R-irréductibles contenant des variables de même sorte que ces derniers si la sorte est infinitaire.

Exemple 5.2.4 *L'image d'une application d'une substitution couvrante, de sorte* `nat`*, est un terme qui peut avoir la forme $s(\cdots(s(x))\cdots)$. Notons que x est de sorte* `nat`*. De même, pour les termes de la forme $Cons(x_1, Cons(\cdots Cons(x_n, y)\cdots))$, de sorte* `list`*. Dans ce cas, la variable y est de sorte* `list`*.*

Il suffit d'éliminer au moins une application de type récursif d'une substitution cumulative afin d'obtenir une substitution candidate.

Proposition 5.2.2 *Soient θ une substitution contenant l'application de type récursif $x \mapsto t[y]$, C une clause telle que le domaine de θ est inclus dans $Var(C)$, et θ' la substitution $(\theta \backslash \{x \mapsto t[y]\}) \cup \{x \mapsto y\}$. Alors $C\theta' \preceq_c C\theta$.*

Preuve Grâce à la propriété de sous-terme de $\leq_t$, on a $y <_t t[y]$. Comme $\leq_t$ est stable par contexte, pour tout terme $s[y]_p$, on a $s[y]_p \leq_t s[t[y]]_p$. Par la construction de $\preceq_c$, on obtient $C\theta' \preceq_c C\theta$.

Fin de preuve

Dans l'algorithme 2, on calcule les substitutions candidates prêtes à vérifier la relation d'ordre et, en cas de succès, la relation de conséquence initiale afin de montrer que la prémisse C subsume

la conjecture C_1. Pour calculer les substitutions candidates qui serviront ultérieurement à la vérification de la relation de conséquence initiale $Ax \models C\sigma \Rightarrow C_1$, on part de la substitution cumulative $\theta_{C \curvearrowright C_1}$ entre C et C_1. Selon la proposition 5.2.2, une substitution candidate peut se déduire à partir de la substitution cumulative $\theta_{C \curvearrowright C_1}$ en modifiant au moins une application de type récursif. Afin de calculer toutes ces substitutions, on va extraire dans un premier temps l'ensemble d'applications de type récursif de $\theta_{C \curvearrowright C_1}$, noté par $\theta^r_{C \curvearrowright C_1}$, et on va construire toutes les substitutions possibles $\mathbb{P}(\theta^r_{C \curvearrowright C_1})$ à partir de celles-ci. Puis, pour chaque substitution σ^r de $\mathbb{P}(\theta^r_{C \curvearrowright C_1})$, on va construire une substitution candidate en modifiant dans (une copie de) $\theta_{C \curvearrowright C_1}$ chaque application de σ^r de la forme $x \mapsto t[y]$ par $x \mapsto y$. Notons que $\mathbb{P}(\theta^r_{C \curvearrowright C_1})$ ne doit pas contenir la substitution vide puisqu'elle ne permet pas de modifier au moins une application de type récursif.

Algorithme 2 Le calcul des substitutions candidates

Entrée : une clause C_1 et une clause C de $hist(C_1)$

Sortie : un ensemble de substitutions candidates $\mathcal{S}$

1: calculer la substitution cumulative $\theta_{C \curvearrowright C_1}$ entre C et C_1.
2: **soit** $\theta^r_{C \curvearrowright C_1}$ l'ensemble d'applications de type récursif de $\theta_{C \curvearrowright C_1}$
3: $\mathcal{S} \leftarrow \emptyset$
4: **soit** $\mathbb{P}(\theta^r_{C \curvearrowright C_1})$ l'ensemble des parties de $\theta^r_{C \curvearrowright C_1}$ dont on élimine la substitution vide
5: **pour toute** substitution σ^r de $\mathbb{P}(\theta^r_{C \curvearrowright C_1})$ **exécute**
6: $\quad\gamma \leftarrow \theta_{C \curvearrowright C_1}$
7: $\quad$**pour toute** application $x \mapsto t[y]$ de σ^r **exécute**
8: $\quad\quad$**soit** $\gamma \leftarrow (\gamma \backslash \{x \mapsto t[y]\}) \cup \{x \mapsto y\}$
9: $\quad$**fin pour**
10: $\quad\mathcal{S} \leftarrow \mathcal{S} \cup \gamma$
11: **fin pour**
12: **retourne** $\mathcal{S}$

Théorème 5.2.2 *L'algorithme 2 termine.*

Preuve L'historique de C_1 est une liste finie, ainsi que le nombre d'applications contenues par les substitutions couvrantes.

Fin de preuve

Comme on a déjà mentionné au début de la section, cette heuristique a été utilisée avec succès lors de la preuve de l'algorithme MJRTY. Pendant le traitement du sous-but non-trivial de l'invariant présenté dans la section 5.2.1, la substitution σ a été ainsi calculée à partir de la substitution (cumulative) θ, en remplaçant l'application de type récursif $x_3 \mapsto s(i)$ par $x_3 \mapsto i$.

Normalement, l'application de la règle Inductive Semantic Subsumption n'est pas efficace dans des preuves complètement automatiques en raison du nombre relativement élevé de substitutions candidates à tester, exponentiel avec le nombre d'applications de type récursif de la substitution cumulative, d'une part, et du test (toujours indécidable) de la relation de conséquence initiale, d'autre part. On envisage plutôt son utilité dans des environnements de preuves interactifs, où l'utilisateur serait guidé par cette heuristique dans son choix des substitutions.

5.3 SPIKEpar : une interface parallèle de SPIKE

Dans cette section, nous décrivons une interface parallèle du démonstrateur automatique SPIKE telle qu'elle est mise en œuvre sur un réseau de stations de travail monoprocesseur. En fait, l'utilisation des environnements parallèles et distribués est une solution générale pour améliorer le temps d'exécution du calcul symbolique. Pourtant, les preuves par récurrence basées sur la réécriture sont connues comme étant difficiles à paralléliser. Ceci est dû en partie à la forte dépendance de la stratégie de preuve choisie, elle-même étant sensible aux décisions de programmation dynamique des tâches entre processeurs. La stratégie utilisée au cours d'une preuve a une grande influence sur les performances du démonstrateur. Elle se répercute sur les tests finaux de performance et rend difficile l'analyse de l'efficacité du travail en parallèle.

On dispose de deux approches pour paralléliser des programmes liés au calcul symbolique. La première utilise le *parallélisme de l'espace de recherche*, aussi nommé *parallélisme de type OU*, qui consiste à trouver parmi plusieurs stratégies celle qui mène à une solution. Le gain en performance peut être obtenu par l'exécution parallèle de différentes stratégies jusqu'à l'obtention d'une première solution. D'autre part, si une stratégie optimale est employée, les avantages d'une utilisation de cette approche disparaissent.

La deuxième approche exploite le *parallélisme de travail*, aussi connu sous le nom de *parallélisme de type ET*, et s'appuie sur le partage du travail global entre plusieurs processeurs. Dans sa version pure, l'algorithme parallèle s'exécute chaque fois avec la même stratégie ; on parle alors d'une *parallélisation conforme à la stratégie* [Bündgen *et al.*, 1996]. Une telle approche a de nombreux avantages, comme la possibilité d'effectuer des calculs déterminés et prévus, ainsi que des expériences reproductibles.

La grande complexité d'une preuve par récurrence basée sur la réécriture permet des schémas de parallélisation à plusieurs niveaux de granularité, selon la dimension de la tâche. La plupart des travaux théoriques sont consacrés à l'étude de la parallélisation à grain fin (par exemple, le traitement du filtrage parallèle), tandis que pendant les expérimentations on utilise des schémas de parallélisation à gros grain, où l'efficacité du calcul est facile à obtenir mais les effets de la stratégie utilisée peuvent intervenir.

Dans la suite, on va détailler la description de SPIKEpar [Stratulat, 1998b], une interface parallèle de SPIKE [Bouhoula et Rusinowitch, 1995a] qui combine le schéma maître-esclave avec un schéma de parallélisation de travail à gros grain, conforme à la stratégie.

5.3.1 Schéma de parallélisation

L'algorithme utilisé dans la mise en œuvre de SPIKEpar a été conçu pour exploiter le parallélisme à gros grain sur une architecture de processeurs à mémoire distribuée en suivant le paradigme *diviser pour régner* lors de l'attribution du travail aux processeurs. Les rôles de l'interface sont de gérer l'état global de la preuve, de lancer des processus SPIKE et d'analyser leur état courant, par l'observation de leur trace, pour décider si les processus ont fini avec succès, ont échoué ou une certaine règle d'inférence a été appliquée. Dans la suite, on va détailler cet algorithme et on va introduire des restrictions sur la stratégie de SPIKE, suffisantes pour que le schéma de parallélisation soit conforme à la stratégie.

L'analyse du système d'inférence de SPIKE, avec une stratégie plus raffinée qui interdit les simplifications d'une conjecture avec d'autres conjectures, nous a permis d'isoler des fragments de preuve qui peuvent être exécutés indépendamment. Une première étape de parallélisation est

basée sur l'observation du fait que chaque conjecture de l'ensemble de conjectures intermédiaires, issues d'une étape où GENERATE a été appliquée, est candidate à une étape de normalisation (notée ensuite par NORM), c'est-à-dire une suite maximale d'applications des J'-règles d'inférences instanciant A-SIMPLIFY. En outre, un ensemble de conjectures est initialement valide si chacune de ses conjectures l'est aussi. Par conséquent, l'opération de normalisation peut s'exécuter indépendamment pour chaque conjecture à l'aide d'un processus SPIKE lancé avec un même ensemble de prémisses. Dès que les opérations de normalisation sont terminées pour chaque conjecture, on joint les conjectures résultées dans un nouvel état global de la preuve.

Algorithme 3 L'algorithme mis en œuvre par SPIKEPAR

Entrée : l'état initial de la dérivation (E^0, H^0)

1: **soit** $(E, H) \leftarrow (E^0, H^0)$ et fin_preuve $\leftarrow$ *False*
2: **tant que** $\neg$ fin_preuve **et** $E \neq \emptyset$ **exécute**
3: (Normalisation parallèle) **soit** E de la forme $\cup_{j=1}^{n}\{C_j\}$
4: processus k : $(\{C_k\}, H) \vdash^{\mathsf{J}'_{\text{NORM}}} (E'_k, H)$
5: (jointure) $(E, H) \leftarrow (\cup_{j=1}^{n} E'_j, H)$
6: (GENERATE parallèle) **soit** E de la forme $\cup_{j=1}^{m}\{D_j\}$
7: processus k : $(\{D_k\}, H) \vdash^{\mathsf{J}'_{\text{GENERATE}}} (E'_k, H \cup \{D_k\})$
8: (jointure) $(E, H) \leftarrow (\cup_{j=1}^{m} E'_j, H \cup E)$
9: **fin tant que**

De la même manière, on raffine encore plus la stratégie en interdisant, pendant une application de GENERATE, les simplifications des conjectures avec des prémisses d'un même niveau de preuve† afin de permettre une opération de GENERATE parallèle. Ainsi, une règle GENERATE parallèle peut être considérée comme la composition des règles GENERATE sur chaque conjecture. Les conjectures et les prémisses, calculées par des processus SPIKE, sont réunies pour former le nouvel état de la preuve globale. Le schéma de parallélisation est défini par l'algorithme 3. Notons qu'un échec de la preuve d'une conjecture issue d'un processus SPIKE est interprété comme un échec global.

Pour une application donnée, il y a des points de synchronisation de toutes les traces de preuves engendrées avec un nombre arbitraire de processeurs, en supposant qu'un processeur exécute un seul processus SPIKE à la fois. Par exemple, l'ensemble (E, H) calculé au pas 5 (ou 8) est identique pour toutes les preuves issues de différentes exécutions, comptant pour la même itération. Par conséquent, le travail fait avec plusieurs processeurs est exactement le même que celui effectué en utilisant une stratégie séquentielle, sur un seul processeur. La preuve de correction et de correction réfutationnelle s'avère triviale car toute preuve construite avec une stratégie séquentielle peut se rejouer avec SPIKE en utilisant la stratégie raffinée. De plus, l'algorithme présenté définit une stratégie équitable parce que toutes les conjectures de l'ensemble E sont traitées aux étapes 4 et 7, assurant ainsi la complétude réfutationnelle. Finalement, SPIKEPAR met en œuvre une parallélisation conforme à la stratégie qui garantit l'utilisation de la même stratégie de preuve vis-à-vis de la manière de programmation des tâches ou du nombre de processeurs employés.

† On dit que deux conjectures ont le même niveau de preuve si la taille de leur historique (cf. la définition 5.2.4) est la même

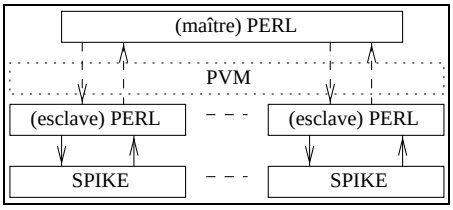

FIG. 5.4 – *Le schéma maître-esclave*

5.3.2 Mise en œuvre

La mise en œuvre de SPIKEPAR suit le schéma maître-esclave de la figure 5.4, qui est constitué d'un processus maître et d'un ou plusieurs processus esclaves. Le processus maître exécute l'algorithme 3 ; il identifie les fragments de preuve qui peuvent être traités indépendamment, crée des processus esclaves selon le type d'opération en cours d'exécution, collecte et interprète les résultats des processus esclaves. Chaque processus esclave contrôle un processus SPIKE afin de traiter la conjecture attribuée par le processus maître et veille à l'exécution correcte de l'opération qu'il a prise en charge.

Le transfert des paramètres d'un processus esclave à un processus SPIKE se fait à l'aide des fichiers. Lorsque la preuve de la conjecture est lancée, le processus esclave contrôle son état courant par l'analyse des mots-clés de la trace de preuve engendrée par le processus SPIKE. Le résultat de l'opération est envoyé au processus maître dès qu'il est détecté dans la trace de la preuve. Si un processus esclave signale un échec, le processus maître arrête tous les autres processus esclaves et termine par un échec global. La création/destruction des processus esclaves et le transfert des paramètres entre un processus esclave et maître se fait à l'aide de PVM [Geist et others, 1994], un environnement de programmation distribuée. L'utilisation de PVM permet également le chargement automatique des processus sur les processeurs disponibles dans sa configuration. La communication entre processus via PVM est faible et a une influence négligeable sur les temps d'exécution ; elle n'apparaît qu'au moment de la création et vers la fin du processus esclave.

L'algorithme de parallélisation et les codes des processus esclaves ont été écrits en PERL [Wall et others, 1997]. PERL a été choisi pour sa capacité de détecter efficacement des mots-clés, écrits sous la forme des expressions régulières, contenus par les traces de preuves. Le code PERL de SPIKEPAR s'étend sur trois fichiers : un est associé au processus maître et un pour chaque type de processus esclave, selon la nature de l'opération effectuée (GENERATE ou NORM). Le nombre de lignes de code est d'environ 1000.

5.3.3 Résultats expérimentaux

Nous avons mis en œuvre et testé SPIKEPAR sur un réseau de 4 stations de travail DEC/ALPHA, de type 500/333, avec une mémoire vive de 128 Moctets.

Un certain nombre de problèmes ont été traités avec SPIKEPAR, dont le problème du tour de cartes de Gilbreath (`gct`) et la correction d'un algorithme de tri par insertion (`sorted`) [Bouhoula, 1994]. Les résultats sont présentés dans le tableau 5.1. A noter que les résultats confirment que la parallélisation est *échelonnable*, dans le sens où, si le problème contient assez de parallélisme, les temps d'exécution s'améliorent avec l'ajout de processeurs.

étude	temps exécution sur n processeurs (s)				nbe processus esclaves	
de cas	$n=1$	$n=2$	$n=3$	$n=4$	GENERATE	NORM
`sorted`	23	17	14	13	7	20
`gct`	258	205	183	153	59	106

TAB. 5.1 – *Les résultats*

Pour le problème `gct` [Bouhoula, 1994], l'efficacité obtenue en utilisant deux processeurs est plus élevée que dans les cas où $n > 2$. Cela s'explique en analysant la structure de la preuve ; l'application de GENERATE aux deux conjectures initiales demande un temps important rapporté au temps total de la preuve. Ainsi, l'ajout d'un nouveau processeur ne contribuera guère à la construction de ce fragment de preuve.

SPIKEPAR a été aussi utilisé dans la détection des interactions de services téléphoniques, ce qu'on présentera dans le chapitre 7.

5.3.4 Discussions sur SPIKEpar

Les temps d'exécution d'une preuve avec SPIKEPAR dépendent aussi de la performance de SPIKE : une version de SPIKE plus performante va induire une version de SPIKEPAR plus efficace. Nous pensons que SPIKEPAR s'adapterait facilement à une nouvelle version de SPIKE, vu qu'en principe il suffit d'ajuster les nouveaux mots-clés les expressions régulières qui reconnaissent les mots-clés des traces de preuve.

Il y a également des inconvénients, par exemple, pour le passage à une parallélisation de granularité plus fine ; la parallélisation des opérations sur des termes ne peut plus être contrôlée avec SPIKEPAR. La solution, qui constitue un projet futur, est la conception d'une version parallèle de SPIKE. Elle permettra, entre autres, l'accès direct aux structures de données et un contrôle plus précis et efficace de la preuve.

5.4 Conclusions

Nous avons appliqué une méthodologie simple pour intégrer de nouvelles techniques de raisonnement dans le système d'inférence $\mathsf{A}(\mathcal{RM})$. Pour l'appliquer à une nouvelle technique particulière, il faut premièrement montrer qu'on peut construire un module de raisonnement en utilisant cette technique. Puis, selon le type d'ensemble couvrant contextuel engendré par le module de raisonnement, soit on étend des règles d'inférences déjà existantes, grâce aux propriétés de composition des ensembles couvrants contextuels, soit on définit de nouvelles règles d'inférence, en instanciant une des règles ADDPREMISE ou SIMPLIFY du système abstrait A.

Comme étude de cas, nous avons proposé une nouvelle technique de raisonnement : la subsomption sémantique inductive, et on l'a intégrée dans SPIKE. De manière similaire, nous avons procédé à l'intégration d'une procédure de décision pour l'arithmétique linéaire. Le nouveau système d'inférence a été utilisé pour montrer la correction de l'algorithme MJRTY, par l'emploi d'un raisonnement inductif et arithmétique.

Une autre optimisation du système d'inférence de SPIKE consiste dans la diminution du temps d'exécution des preuves par le traitement parallèle des conjectures. Nous avons ainsi proposé une interface parallèle, SPIKEPAR, qui isole les conjectures qui peuvent être traitées

indépendamment par des processus SPIKE. Le schéma de parallélisation est à gros grain, échelonnable et conforme à la stratégie. De plus, il garde les propriétés de correction et complétude du système d'inférence de SPIKE.

6

Techniques de raisonnement basées sur la coopération de procédures de décision en SPIKE

Sommaire

6.1 Introduction

Le raisonnement sur des combinaisons de plusieurs théories est essentiel pour des applications comme l'optimisation des compilateurs, les langages fonctionnels ou bien pour la vérification des circuits électroniques et des codes de spécifications. Un certain nombre d'algorithmes et de schémas de combinaisons ont été introduits par Nelson et Oppen dans [Nelson et Oppen, 1979] et Shostak dans [Shostak, 1984], ou plus récemment dans [Bjørner, 1998; Tiwari, 2000]. Dans ces approches, l'idée est de concevoir une procédure de décision pour un mélange de théories en utilisant des procédures de décision pour chaque théorie. Celles-ci communiquent entre elles par des égalités gérées par des procédures de décision pour la théorie de l'égalité, spécialisées par exemple dans le calcul de leur *clôture par congruence*. Ainsi, on définit de nouvelles procédures de décision pour la combinaison des théories considérées. Différents schémas de combinaison ont été mis en œuvre dans des démonstrateurs comme RRL [Kapur et Zhang, 1989], PVS [Owre *et al.*, 1992] et STeP [Bjørner et others, 1995], ou des compilateurs comme Touchstone [Necula, 1998].

Dans ce chapitre, on fait dans un premier temps une brève description des approches de Nelson-Oppen et Shostak. Ensuite, on présente un algorithme de coopération entre deux procédures de décision, une pour la théorie de l'égalité, et l'autre pour l'arithmétique linéaire, qui s'adapte bien à l'intégration dans le système d'inférence de SPIKE du chapitre 4. Puis on montre sur un exemple l'application de notre algorithme pour vérifier des conjectures. Finalement, on prouve que l'algorithme définit une procédure de (semi-)décision correcte.

6.2 Etat de l'art

Généralement, le *problème de combinaison* de la satisfaisabilité pour un mélange de plusieurs théories s'énonce comme suit : Etant données deux théories définies respectivement sur un ensemble d'axiomes Ax_1 et Ax_2 en utilisant la signature Σ_1 et Σ_2, on s'intéresse à construire de manière modulaire une procédure de décision pour décider la satisfaisabilité de toute formule ϕ sur la signature $\Sigma_1 \cup \Sigma_2$ dans la théorie de $Ax_1 \cup Ax_2$ en utilisant des procédures de décision pour les théories individuelles. Les algorithmes de combinaison usuels se distinguent par le choix des axiomes, des signatures et des formules.

6.2.1 Approche de Nelson-Oppen

Nelson et Oppen ont été les premiers à proposer dans [Nelson et Oppen, 1979] une solution générale au problème des combinaisons des théories. Les procédures de décision se construisent à partir des *procédures de satisfaisabilité* qui vérifient si un ensemble de formules atomiques est satisfaisable ou pas. Rappelons que pour montrer qu'une formule est une conséquence des axiomes, il suffit de montrer que sa négation est insatisfaisable, comme on l'a déjà vu dans l'exemple 1.6.1.

Dans [Nelson, 1981], on précise les conditions à satisfaire par les théories combinées : Afin de détecter l'insatisfaisabilité d'un ensemble Φ de formules atomiques construites sur les ensembles de symboles de fonctions libres de deux théories T_1 et T_2 (considérés disjoints), avec des procédures de satisfaisabilité S et T respectivement, il faut dans un premier temps transformer Φ dans un paire équivalente d'ensembles Φ_{T_1} et Φ_{T_2} telle que chaque ensemble contient seulement des formules atomiques de la théorie correspondante. Le prédicat d'égalité est le seul prédicat partagé. La partition et l'homogénéisation de l'information entre les théories individuelles se font par l'intermédiaire des *variables d'abstraction*. Alors, chaque procédure de satisfaisabilité doit détecter l'insatisfaisabilité de son propre ensemble de formules atomiques et doit aussi propager aux autres procédures toutes les égalités entre les variables d'abstraction qu'elle engendre. Si les deux théories sont *convexes* et les procédures de satisfaisabilité sont complètes, alors la procédure de coopération pour la théorie $T_1 \cup T_2$ est aussi complète.

On dit qu'une théorie n'est pas convexe s'il existe une formule ϕ et $2 * n$ variables $x_1, \ldots, x_n$, $y_1, \ldots, y_n$, avec $n \geq 2$, telles que ϕ implique la disjonction $\bigvee_{i=1}^{n} x_i = y_i$. Des exemples de théories convexes sont les théories sur les rationnels et sur les égalités, tandis que des théories non-convexes sont les théories sur les tableaux et sur l'arithmétique linéaire entière. Le schéma de Nelson et Oppen peut s'adapter pour manipuler les théories non-convexes. L'idée principale est de faire une analyse par cas lorsqu'une formule implique une disjonction afin de trouver laquelle des égalités de la disjonction est satisfaite.

Les exemples illustrés dans [Nelson, 1981] montre que, en général, les procédures de décision doivent interagir d'une façon non-triviale pour détecter l'insatisfaisabilité.

6.2.2 Approche de Shostak

Le schéma proposé par Shostak permet d'optimiser la coopération des procédures de décision pour des théories qui admettent des *canoniseurs* et des *solveurs*. Généralement, les canoniseurs d'une théorie sont des fonctions qui retournent le représentant de la classe de congruence de tout terme de la théorie. Une théorie est *canonisable* s'il existe un canoniseur σ telle que toute égalité $s = t$ est démontrable ssi $\sigma(s) = \sigma(t)$. L'égalité est prouvée dans la théorie *pure* de l'égalité, définie par un ensemble vide d'axiomes. Une théorie est *algébriquement solvable* si toute égalité $s = t$ peut s'écrire sous une forme résolue équivalente de la forme $\wedge_{i=1}^{n} x_i = t_i$, où chaque x_i apparaît en $s = t$ mais pas en aucun des t_j. Des exemples de théories canonisables et algébriquement solvables sont l'arithmétique linéaire réelle, la théorie convexe des listes et la théorie des ensembles monadiques.

Voici quelques arguments dans la faveur de l'approche de Shostak, tirés de [Cyrluk *et al.*, 1996; Kapur, 1997]. Son efficacité est justifiée par le fait que l'algorithme de clôture par congruence est le point clé dans le cadre de la combinaison [Shostak, 1979; Shostak, 1984]. Shostak ainsi centralise le raisonnement équationnel dans une seule procédure permettant aux procédures de décision de coopérer plus étroitement. D'autre part, dans l'approche de Nelson-Oppen on renomme tout sous-terme non-interprété (par rapport à une théorie individuelle) dans des variables d'abstraction et on propage toute nouvelle équation déduite entre les variables de théories différentes. Le travail est redondant car chaque théorie individuelle retrouve un peu près la même notion d'égalité. Une autre distinction importante est le caractère itératif de l'approche de Shostak. L'univers des termes à considérer peut changer puisque les formes canoniques des termes se calculent en fonction de l'ensemble courant d'équations.

D'autre part, l'algorithme original de Shostak est peu abordable. En particulier, la compréhension et la correction de la partie qui calcule implicitement la forme canonique d'un terme ont été largement débattues pendant la dernière décade, par exemple dans [Cyrluk *et al.*, 1996; Kapur, 1997]. Très récemment, H. Ruess et N. Shankar de SRI International ont montré dans [Ruess et Shankar, 2001] que l'algorithme de Shostak est incomplet et non-terminant, comme tous les autres algorithmes dérivés de lui [Cyrluk *et al.*, 1996; Bjørner, 1998]. De plus, ils proposent dans le même papier un nouvel algorithme plus simple, complet, correct et terminant.

6.3 Algorithme de coopération

Dans cette section, on va détailler un algorithme de coopération entre deux procédures de décision pour i) la théorie de l'égalité dont les axiomes ont été présentées dans l'exemple 1.3.1, et ii) l'arithmétique linéaire, présentée dans la section 1.6. Une fois intégré dans SPIKE, il servira de procédure de (semi-)décision pour éliminer des conjectures.

Une conjecture est éliminée si on obtient une contradiction à partir de sa négation. Pour cela, les formules atomiques de la négation d'une conjecture C doivent initialiser une structure de données adaptée à notre algorithme, appelée $\overline{C}$-structure, sur laquelle l'algorithme opère.

Important : tout au long de ce chapitre, on va entendre par terme (formule) *clos(e)* tout(e) terme (formule) contenant seulement des variables dont on interdit l'instanciation. Puisque les variables des formules atomiques contenues par la conjecture niée sont quantifiées existentiellement, elles peuvent être considérées comme des constantes. Ceci permet aux termes (formules)

d'être vu(e)s comme des termes (formules) clos(es) dans une signature élargie.

Définition 6.3.1 ($\overline{C}$-structure) *Soit C une conjecture. Une $\overline{C}$-structure est une structure de données décrite par un quadruple $\langle CR \mid A \mid G \mid L \rangle$, où*

- *CR contient des règles de réécriture inconditionnelles closes ;*
- *A contient des formules atomiques provenant de la négation de C ;*
- *G contient des équations et des diséquations inconditionnelles closes ;*
- *L représente la paire $(P \bullet IE)$, où P contient des inégalités linéaires et IE des équations implicites dérivées de P.*

Voici un exemple d'initialisation de la $\overline{C}$-structure d'une conjecture C.

Exemple 6.3.1 *Soit C la conjecture*†

$$(p(x) = True \ \wedge \ z \leq f(max(x,\, y)) = True \ \wedge \ 0 < min(x,\, y) = True \wedge \\ x \leq max(x,\, y) = True \wedge \ max(x,\, y) \leq x = True) \ \Rightarrow \ z < g(x) + y = True.$$

La $\overline{C}$-structure associée à C est

$$\begin{array}{lll} \langle CR & : & \emptyset \mid \\ A & : & \{p(x) = True,\, z \leq f(max(x,\, y)) = True,\, 0 < min(x,\, y) = True, \\ & & \quad x \leq max(x,\, y) = True,\, max(x,\, y) \leq x = True,\, z < g(x) + y = False\} \mid \\ G & : & \emptyset \mid \\ L & : & (\emptyset \bullet \emptyset)\rangle \end{array}$$

Afin de simplifier la notation, on va représenter seulement les champs non-vides de la $\overline{C}$-structure. Pour l'exemple précédent, la $\overline{C}$-structure devient ainsi

$$\begin{array}{lll} \langle\ A & : & \{p(x) = True,\, z \leq f(max(x,\, y)) = True,\, 0 < min(x,\, y) = True, \\ & & \quad x \leq max(x,\, y) = True,\, max(x,\, y) \leq x = True,\, z < g(x) + y = False\}\ \rangle \end{array}$$

Les règles de réécriture de CR permettront de normaliser les monômes de P et les équations closes de IE. Les formules atomiques de A seront partagées entre L et G selon le critère suivant : Toute formule atomique qui peut se transformer dans une conjonction (vue comme un ensemble) d'inégalités linéaires selon les transformations affichées dans le tableau 1.1, va être envoyée à P, sinon à G. G contient l'ensemble d'égalités auquel on va appliquer l'algorithme de clôture par congruence pour déduire de nouvelles égalités. La procédure de décision pour l'arithmétique linéaire va s'appliquer sur les inégalités linéaires de P.

On va noter par *linearize* le processus de transformation d'une formule atomique dans une conjonction d'inéquations linéaires. Une formule est *linéarisable* si et seulement si on peut lui appliquer *linearize*.

Exemple 6.3.2 *Toute formule atomique de la forme $p = True$ (resp. $p = False$) est équivalente à $linearize(p)$ (resp. $linearize(\neg p)$) si p est linéarisable. Ainsi*

- *$linearize(z \leq f(max(x,\, y)) = True) = \{z - f(max(x,\, y)) \leq 0\}$, et*
- *$linearize(z < g(x) + y = False) = \{g(x) + y - z \leq 0\}$.*

† [Kapur *et al.*, 1994]

La procédure de décision pour l'arithmétique linéaire L'algorithme *arith* met en œuvre la procédure de décision présentée dans l'exemple 1.6.1. Il prend en entrée un ensemble d'inégalités linéaires et retourne soit i) **inconsistant**, si une inégalité linéaire impossible a été dérivée, soit ii) un nouvel ensemble d'inégalités linéaire avec l'ensemble d'égalités implicites préalablement déduites.

Les monômes d'une inégalité linéaire sont ordonnés selon un pré-ordre total sur les termes, similaire à celui présenté dans [Kaufmann et Moore, 1999].

Définition 6.3.2 (pré-ordre total sur les termes clos) *Soient t_1 et t_2 deux termes et $<_{gt}$ la partie stricte du pré-ordre, définie comme suit : $t_1 <_{gt} t_2$ ssi*

- *le nombre de positions de variables de t_1 est inférieur à celui de t_2, ou*
- *les nombres de positions de variables de t_1 et t_2 sont égaux, mais le nombre de positions strictes de t_1 est inférieur à celui de t_2, ou*
- *les nombres de positions de variables de t_1 et t_2 sont égaux, les nombres de positions strictes de t_1 et t_2 sont égaux et t_1 est plus petit que t_2 selon un certain ordre lexicographique*†*.*

Ce pré-ordre peut s'étendre aux termes clos si on considère un pré-ordre lexicographique total sur les variables.

Le calcul de la clôture par congruence L'entrée de l'algorithme de calcul de la clôture par congruence, *congr*, est un ensemble fini d'équations et de diséquations. Le résultat est soit i) **inconsistant**, si une diséquation triviale de la forme $s \neq s$ a été dérivée, soit ii) un nouvel ensemble d'équations et de diséquations. Les nouvelles équations sont engendrées par un algorithme de *complétion* [Knuth et Bendix, 1970] sur les termes clos en utilisant la réécriture, comme dans [Huet et Lankford, 1978]. Les équations rencontrées pendant le processus de complétion sont transformées en règles de réécriture inconditionnelles, en les orientant à l'aide de l'ordre $<_{gt}$. Puisque l'algorithme de complétion sur un ensemble d'équations closes termine toujours, le nouveau système de réécriture obtenu à la fin peut être utilisé pour normaliser les membres des diséquations.

Dans la suite, on va schématiser l'échange de données entre les composantes d'une $\overline{C}$-structure et entre la $\overline{C}$-structure et l'environnement de preuve. Puis, on va présenter un exemple d'application de ce schéma de coopération.

6.3.1 Diagramme du flot de données

Les interactions entre les composantes d'une $\overline{C}$-structure et entre la $\overline{C}$-structure et l'environnement de preuve sont décrites dans la figure 6.1 par des opérations indiquant le flot de données. Les composantes de la $\overline{C}$-structure sont encadrés dans un rectangle en pointillé.

Chaque opération peut être considérée comme une transition entre deux états de la $\overline{C}$-structure. Par $\xrightarrow{\mathsf{name-op}}$, on dénote la transition associée à l'opération $\mathsf{name-op} \in \{\mathsf{G2CR}, \mathsf{A2G}, \mathsf{L2G}, \mathsf{A2L}, \mathsf{normalize}, \mathsf{augment\_L}\}$, définie dans la figure 6.2. De plus, $\xrightarrow{\mathsf{name-op}}$ retourne une liste d'opérations à exécuter ultérieurement.

† imposé par des ordres sur la structure des objets du langage OCaml [Leroy *et al.*, 2000], utilisé pour mettre en œuvre SPIKE.

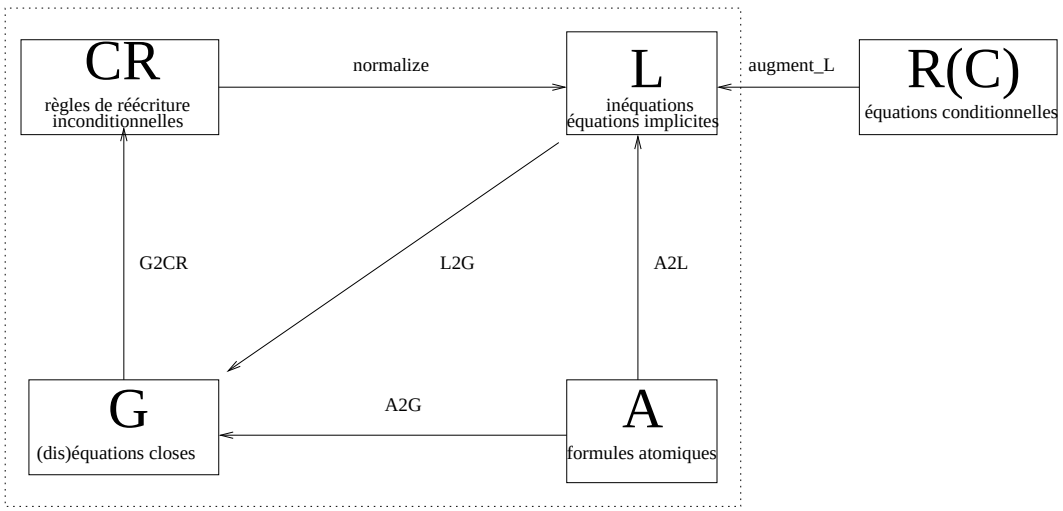

FIG. 6.1 – *Le flot de données*

Les premières deux opérations de la figure 6.2, A2L et A2G, initialisent respectivement les composantes P (de L) et G avec des inéquations linéaires et des équations closes obtenues à partir des formules atomiques de A. La règle A2L applique en outre l'algorithme *arith* sur le nouvel ensemble d'inéquations linéaires. Toutes les nouvelles règles de réécriture, obtenues après l'application de *congr* sur G, sont transférées à CR par G2CR. La règle L2G vide simplement les équations implicites de IE pour les stocker dans G. Les monômes des inégalités linéaires de P et les équations closes de IE sont normalisés par des règles de réécriture de CR, suite à l'application de normalize. La dernière des opérations, augment_L, augmente l'ensemble d'inéquations linéaires en utilisant des règles de réécriture provenant de l'environnement de preuve. Ensuite, elle applique *arith* sur le nouvel ensemble d'inégalités.

L'ensemble $R(C)$ contient des (instances d') équations conditionnelles provenant des axiomes, des lemmes ou bien des éléments du contexte de C, selon le type de la règle d'inférence appliquée à la clause C (voir la figure 3.1).

L'opération d'augmentation, dénotée par *Oracle_A*, est décrite dans la figure 6.3. Dans un premier temps, il faut trouver une instance d'une règle de $R(C)$ dont la conclusion est linéarisable et l'ensemble d'inégalités obtenu après linéarisation i) contient un monôme maximal d'une inégalité de P, mais avec un coefficient de signe opposé, et ii) tous les autres monômes s'identifient avec des monômes des inégalités de P. Si la conclusion de la règle est prouvable, on retourne l'ensemble d'inégalités. Dans la preuve des conclusions, on a le droit d'utiliser des éléments du contexte de C, selon le schéma d'intégration de la définition 3.5.2.

La stratégie La façon d'exécuter les opérations peut être fixée par une liste de noms d'opérations, initialisée par [A2G, A2L]. Le premier élément de la liste détermine l'opération courante à exécuter. A chaque application d'une opération, on remplace son nom avec les noms d'opérations de la liste retournée après son exécution. On dit que le schéma de coopération a échoué si on arrive à la liste vide []. Dans ce cas, le schéma de coopération n'a pas été suffisamment puissant pour détecter des inconsistances.

$\langle CR \mid A \cup \mathcal{E} \mid G \mid (P \bullet IE)\rangle \xrightarrow{\text{A2L}} \langle CR \mid A \mid G \mid (P' \bullet IE \cup IE')\rangle$

$$où \begin{cases} (1)\ \mathcal{E} \text{ est l'ensemble maximal tel que } \forall e \in \mathcal{E},\ e \text{ est linéarisable} \\ (2)\ (IE',\ P') = arith(\{\cup_{e' \in \mathcal{E}} linearize(e')\} \cup P) \end{cases}$$

Retourne : si $\mathcal{E} = \emptyset$ alors [] sinon (si $IE' = \emptyset$ alors [augment_L] sinon [augment_L; L2G])

$\langle CR \mid A \cup \mathcal{E} \mid G \mid L\rangle \xrightarrow{\text{A2G}} \langle CR \mid A \mid G \cup \mathcal{E} \mid L\rangle$

où $\mathcal{E}$ est l'ensemble maximal tel que $\forall a = b \in \mathcal{E}$, $a = b$ n'est pas linéarisable

Retourne : [G2CR]

$\langle CR \mid A \mid G \mid L\rangle \xrightarrow{\text{G2CR}} \langle CR \cup \mathcal{E} \mid A \mid congr(G) \mid L\rangle$

où $\forall a \to b \in \mathcal{E}$, on a $a = b \in congr(G)$ et $a >_t b$

Retourne : [normalize]

$\langle CR \mid A \mid G \mid (P \bullet IE)\rangle \xrightarrow{\text{L2G}} \langle CR \mid A \mid G \cup IE \mid (P \bullet \emptyset)\rangle$

Retourne : [G2CR]

$\langle CR \mid A \mid G \mid (P \bullet IE)\rangle \xrightarrow{\text{normalize}} \langle CR \mid A \mid G' \mid (P' \bullet IE')\rangle$

$$où \begin{cases} (1)\ P' \text{ est } P \text{ dont les monômes sont normalisés par des règles de } CR \\ (2)\ IE' \text{ est } IE \text{ dont les membres de ses équations} \\ \text{sont normalisés par des règles de } CR \end{cases}$$

Retourne : [A2L]

$\langle CR \mid A \mid G \mid (P \bullet IE)\rangle \xrightarrow{\text{augment\_L}} \langle CR \mid A \mid G \mid (P' \bullet IE \cup IE')\rangle$

$$où \begin{cases} (1)\ \mathcal{E} = Oracle\_A(\langle CR \mid A \mid G \mid (P \bullet IE)\rangle) \\ (2)\ (P',\ IE') = arith(\{\cup_{e \in \mathcal{E}} linearize(e)\} \cup P) \end{cases}$$

Retourne : si $\mathcal{E} = \emptyset$ alors [] sinon (si $IE' = \emptyset$ alors [A2L] sinon [L2G; A2L])

FIG. 6.2 – *Opérations décrivant le flot de données*

Oracle_A$(\langle CR \mid A \mid G \mid (P \bullet IE)\rangle)$

$$\text{où} \begin{cases} (1)\ p_1 \wedge \ldots \wedge p_n \Rightarrow l = r \text{ est une instance d'une équations conditionnelle de } R(C) \\ (2)\ u \text{ est un monôme de } linearize(l = r) \text{ dont le coefficient est } c \\ (3) \text{ les autres monômes de } linearize(l = r) \text{ se trouvent dans la } \overline{C}\text{-structure} \\ (4)\ u \text{ est aussi un monôme maximal d'une inégalité de } P \text{ dont le coefficient est } c' \\ \quad \text{tel que } c * c' < 0 \\ (5) \text{ il existe une } J'\text{-preuve de } (\cup_{i=1}^{n}\{C \vee p_i\},\ H) \text{ sachant que } H \\ \text{sont des éléments du contexte de } C \end{cases}$$

Retourne : $l = r$

FIG. 6.3 – *L'opération d'augmentation* Oracle_A

Un exemple

Soit C la conjecture de l'exemple 6.3.1. On suppose que $R(C)$ contient $\{max(x', y') = x' \Rightarrow min(x', y') = y',\ p(x') = True \Rightarrow f(x') \leq g(x') = True\}$.

On va montrer comment on peut déduire une inconsistance par l'application des règles de la figure 6.2 sur la $\overline{C}$-structure initiale de C :

$$\begin{array}{lll} \langle\ A & : & \{p(x) = True,\ z \leq f(max(x,\ y)) = True,\ 0 < min(x,\ y) = True, \\ & & \quad x \leq max(x,\ y) = True,\ max(x,\ y) \leq x = True,\ z < g(x) + y = False\}\ \rangle \end{array}$$

Premièrement, on distribue les éléments de A à G et L. La première formule atomique de A est la seule qui n'est pas linéarisable. Par conséquent, l'état de la $\overline{C} - structure$ après l'application de la règle A2G et l'initialisation de la composante P de L par A2L est :

$$\begin{array}{lll} \langle\ G & : & \{p(x) = True\} \\ \quad L & : & (\{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \\ & & \quad \leq 0,\ g(x) + y - z \leq 0\} \bullet \emptyset)\ \rangle \end{array}$$

Par l'addition des inégalités $x - max(x,\ y) \leq 0$ et $max(x,\ y) - x \leq 0$, on obtient l'inégalité $0 \leq 0$. Ainsi, la procédure de décision pour l'arithmétique linéaire dérive l'équation implicite $max(x,\ y) = x$. D'autre part, l'inégalité $g(x) + y - f(max(x,\ y)) \leq 0$ est le résultat de l'addition de $g(x) + y - z \leq 0$ et $z - f(max(x,\ y) \leq 0$. Le nouvel état de la $\overline{C}$-structure après l'application de A2L devient :

$$\begin{array}{lll} \langle\ G & : & \{p(x) = True\} \\ \quad L & : & (\emptyset \bullet \{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \\ & & \quad \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(max(x,\ y)) \leq 0\} \bullet \{max(x,\ y) = x\})\rangle \end{array}$$

On transfère $max(x,y) = x$ de IE vers G, par l'intermédiaire de L2G :

$$\begin{array}{lll} \langle\ G & : & \{p(x) = True,\ max(x,\ y) = x\} \\ \quad L & : & (\{z - f(max(x,\ y)) \leq 0,\ 1 - min(x,\ y) \leq 0,\ x - max(x,\ y) \leq 0,\ max(x,\ y) - x \\ & & \quad \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(max(x,\ y)) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

A cette étape, on applique la règle G2CR. L'exécution de *congr* sur CR ne produit aucun changement. Pourtant, puisque $max(x, y) >_t x$, l'égalité $max(x, y) = x$ est transformée dans la règle de réécriture $max(x, y) \to x$ et est finalement envoyée à CR :

$$\begin{array}{rcl} \langle\ CR & : & \{max(x, y) \to x\} \\ G & : & \{p(x) = True\} \\ L & : & (\{z - f(max(x, y)) \leq 0,\ 1 - min(x, y) \leq 0,\ x - max(x, y) \leq 0,\ max(x, y) - x \\ & & \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(max(x, y)) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

On applique normalize. Ainsi, $max(x,y)$ est transformé en x dans toutes les inégalités de P :

$$\begin{array}{rcl} \langle\ CR & : & \{max(x, y) \to x\} \\ G & : & \{p(x) = True\} \\ L & : & (\{z - f(x) \leq 0,\ 1 - min(x, y) \leq 0, \\ & & g(x) + y - z \leq 0,\ 0 \leq 0,\ g(x) + y - f(x) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

Par A2L, on utilise la procédure de décision pour l'arithmétique linéaire sur le nouvel ensemble d'inégalités :

$$\begin{array}{rcl} \langle\ CR & : & \{max(x, y) \to x\} \\ G & : & \{p(x) = True\} \\ L & : & (\{z - f(x) \leq 0,\ 1 - min(x, y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0, \\ & & g(x) + y - f(x) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

Dans la suite, il ne résulte plus d'équations implicites dérivées ou de nouvelles inégalités. A cette étape, on utilise augment_L pour ajouter de nouvelles inégalités à P. Ceci est possible grâce à l'équation conditionnelle $\{max(x', y') = x'x \Rightarrow min(x', y') = y'\}$ qui permet d'éliminer le monôme maximal $min(x, y)$ de $1 - min(x, y) \leq 0$ en utilisant la substitution $\{x' \mapsto x, y' \mapsto y\}$. Ainsi, par l'ajout de l'ensemble d'inégalités $\{min(x, y) - y \leq 0, y - min(x, y) \leq 0\}$, obtenu par la linéarisation de $min(x, y) = y$, l'application de A2L induit l'équation implicite $min(x, y) = y$. Elle sera transférée à G par $A2G$:

$$\begin{array}{rcl} \langle\ CR & : & \{max(x, y) \to x\} \\ G & : & \{p(x) = True,\ min(x, y) = y\} \\ L & : & (\{z - f(x) \leq 0,\ 1 - min(x, y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0, \\ & & g(x) + y - f(x) \leq 0,\ 1 - y \leq 0,\ min(x, y) - y \leq 0,\ y - min(x, y)\} \bullet \emptyset)\rangle \end{array}$$

Comme on a procédé précédemment pour l'équation $max(x, y) = x$, l'équation $min(x, y) = y$ peut être orientée de gauche vers la droite. Par conséquent, $min(x, y) \to y$ est éliminée de G et ajoutée à CR, par l'intermédiaire de G2CR.

$$\begin{array}{rcl} \langle\ CR & : & \{max(x, y) \to x,\ min(x, y) \to y\} \\ G & : & \{p(x) = True\} \\ L & : & (\{z - f(x) \leq 0,\ 1 - min(x, y) \leq 0,\ g(x) + y - z \leq 0,\ 0 \leq 0, \\ & & g(x) + y - f(x) \leq 0,\ 1 - y \leq 0,\ min(x, y) - y \leq 0,\ y - min(x, y)\} \bullet \emptyset)\rangle \end{array}$$

On normalise les inégalités de P avec la règle $min(x, y) \to y$ pour obtenir :

$$\begin{array}{rcl} \langle\ CR & : & \{max(x,\, y) \to x,\, min(x,\, y) \to y\} \\ G & : & \{p(x) = True\} \\ L & : & (\{z - f(x) \leq 0,\, 1 - y \leq 0,\, g(x) + y - z \leq 0, \\ & & 0 \leq 0,\, g(x) + y - f(x) \leq 0\} \bullet \emptyset)\rangle \end{array}$$

Puisque l'application de la procédure de décision n'apporte pas de nouvelles informations, on applique une seconde fois augment_L. Cette fois-ci, on utilise $p(x') = True \Rightarrow f(x') \leq g(x') = True$ avec la substitution $\{x' \mapsto x\}$. La condition $p(x) = True$ est satisfaite en utilisant l'information contenue par la composante G de la $\overline{C}$-structure. On ajoute ainsi l'inégalité $f(x) - g(x) \leq 0$ à P :

$$\begin{array}{rcl} \langle\ CR & : & \{max(x,\, y) \to x,\, min(x,\, y) \to y\} \\ G & : & \{p(x) = True\} \\ L & : & (\{f(x) - g(x) \leq 0,\, z - f(x) \leq 0,\, 1 - y \leq 0,\, g(x) + y - z \leq 0, \\ & & 0 \leq 0,\, g(x) + y - f(x) \leq 0,\, 1 \leq 0\} \bullet \emptyset)\rangle \end{array}$$

Finalement, la procédure de décision engendre l'inégalité impossible $1 \leq 0$ après l'addition de $1 - y \leq 0$, $f(x) - g(x) \leq 0$ et $g(x) + y - f(x) \leq 0$. Par conséquent, elle retourne **inconsistant**.

6.4 Propriétés de l'algorithme de coopération

Dans cette section, on va analyser les propriétés de correction et de terminaison du schéma de coopération décrit dans la section 6.3.

D'abord, on va montrer que l'algorithme appliqué à une conjecture C retourne un ensemble couvrant contextuel vide de C si on retourne **inconsistant**.

Théorème 6.4.1 (correction) *Soit C une clause et $Cxt = (\mathsf{C}^1, \mathsf{C}^2)$ le contexte associé à C. Si, pendant l'application des règles de la figure 6.2 sur sa $\overline{C}$-structure, soit* arith *soit* congr *retourne* ***inconsistant*** *alors $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \models_{ind} C\tau$ pour chaque substitution close τ.*

Preuve On associe à une $\overline{C}$-structure $S = \langle CR \mid A \mid G \mid (P \bullet IE)\rangle$ l'ensemble de formules atomiques

$$\Phi_S \equiv CR \cup A \cup G \cup IE \cup \cup_{p \in P}\{p = True\}$$

Il suffit de montrer que :

1. si S_1 est la $\overline{C}$-structure initiale et $\bigwedge \Phi^C \equiv \neg C$ alors $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \cup \Phi^C\tau \models_{ind} \Phi_{S_1}\tau$, pour toute substitution close τ .
2. pour toutes $\overline{C}$-structures S_i et S_j telles que $S_i \xrightarrow{\mathsf{name-op}} S_j$ avec $\mathsf{name-op} \in \{\mathsf{G2CR}, \mathsf{A2G}, \mathsf{L2G}, \mathsf{A2L}, \mathsf{normalize}, \mathsf{augment\_L}\}$, on a $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \cup \Phi_{S_i}\tau \models_{ind} \Phi_{S_j}\tau$, pour toute substitution close τ.
3. si soit *arith* soit *congr* retourne **inconsistant** lorsqu'on applique une des règles A2L, G2CR, augment_L sur une $\overline{C}$-structure S, alors $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \not\models_{ind} \Phi_S\tau$, pour toute substitution close τ.

Supposons par absurde que ces conditions sont satisfaites et qu'on a retourné **inconsistant** au bout d'une suite de transformations de $\overline{C}$-structures $S_1 \xrightarrow{\mathsf{name-op}} \ldots \xrightarrow{\mathsf{name-op}} S_n$, avec $n \geq 1$. D'autre part, on suppose qu'il existe une substitution close θ telle que $Ax \cup \mathsf{C}^1_{\preceq_c C\theta} \cup \mathsf{C}^2_{\prec_c C\theta} \not\models_{ind} C\theta$. A partir de la première condition, on a $Ax \cup \mathsf{C}^1_{\preceq_c C\theta} \cup \mathsf{C}^2_{\prec_c C\theta} \models_{ind} \Phi_{S_1}\theta$.

Afin de trouver la contradiction, dans la troisième condition on instancie τ avec θ pour obtenir $Ax \cup \mathsf{C}^1_{\preceq_c C\theta} \cup \mathsf{C}^2_{\prec_c C\theta} \not\models_{ind} \Phi_{S_n}\theta$. Par des applications successives de la deuxième condition sur chaque S_i, on déduit que $Ax \cup \mathsf{C}^1_{\preceq_c C\theta} \cup \mathsf{C}^2_{\prec_c C\theta} \not\models_{ind} \Phi_{S_i}\theta$, pour tout $i \in [1..n-1]$ et en particulier pour $i = 1$. Contradiction.

Dans la suite, on va vérifier chacune des conditions. La première est satisfaite car $\Phi_S \equiv \cup_{e\in A}\{e\}$, où A contient toutes les formules atomiques de $\neg C$, résultant ainsi $\Phi_S = \Phi^C$.

Pour vérifier la deuxième condition, on utilise les propriétés suivantes par rapport à :

(Prop. 1). *linearize*. Pour toute formule atomique linéarisable e, si $linearize(e)$ est l'ensemble d'inégalités linéaires P_1 alors $\{e\}\tau \models_{ind} \cup_{p\in P_1}\{p_1 = True\}\tau$, pour toute substitution close τ.

(Prop. 2). *arith*. Pour tout ensemble d'inégalités linéaires P_1, si $arith(P_1) = (IE_2,\ P_2)$, alors $P_1 \Leftrightarrow^*_A P_2$. De plus, les équations implicites de IE_2 sont des conséquences de P_1, selon les théorèmes 1.6.1, 1.6.2, et 1.6.3. Par conséquent, $\cup_{p_1\in P_1}\{p_1 = True\}\tau \models_{ind} (\cup_{p_2\in P_2}\{p_2 = True\} \cup \cup_{e\in IE_2}\{e\})\tau$, pour toute substitution close τ.

(Prop. 3). *congr*. Pour tout ensemble d'équations et diséquations closes $\mathcal{E}$, si $congr(\mathcal{E}) = \mathcal{E}'$, alors $\mathcal{E}\tau \models_{ind} \mathcal{E}'\tau$, pour toute substitution close τ.

On va faire une analyse par cas selon le type d'opération effectuée.

- On considère que A2L a été appliquée sur $S_1 = \langle CR \mid A \cup \mathcal{E} \mid G \mid (P \bullet IE)\rangle$ tel que $\mathcal{E}$ est l'ensemble maximal contenant seulement des formules atomiques linéarisables. La $\overline{C}$-structure résultée est $S_2 = \langle CR \mid A \mid G \mid (P' \bullet IE \cup IE')\rangle$, où P', IE' sont définies dans la suite. Dans un premier temps, si $\mathcal{E} \neq \emptyset$ alors toutes les formules atomiques de $\mathcal{E}$ sont linéarisées dans l'ensemble d'inégalités linéaires dénoté par P_1. Par les propriétés (P2) du chapitre 2 et (Prop. 1), on déduit que $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \cup \cup_{e\in\mathcal{E}}\{e\}\tau \models_{ind} \cup_{p\in P_1}\{p_1 = True\}\tau$, pour toute substitution close τ.
  Puis, sur l'ensemble d'inégalités $P \cup P_1$, on applique $arith$ tel que $arith(P \cup P_1) = (IE', P')$. De nouveau par les propriétés (Prop. 2) et (P2), on a $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \cup \cup_{p\in P\cup P_1}\{p = True\}\tau \models_{ind} (\cup_{p'\in P'}\{p' = True\} \cup \cup_{e\in IE'}\{e\})\tau$, pour toute substitution close τ. Donc $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \cup \Phi_{S_1}\tau \models_{ind} \Phi_{S_2}\tau$, pour toute substitution close τ.
- On suppose que A2G a été appliquée sur $S_1 = \langle CR \mid A \cup \mathcal{E} \mid G \mid L\rangle$ tel qu'on obtient $S_2 = \langle CR \mid A \mid G \cup \mathcal{E} \mid L\rangle$. La deuxième condition est trivialement satisfaite puisque $\Phi_{S_1} = \Phi_{S_2}$.
- Supposons que G2CR a été appliquée sur $S_1 = \langle CR \mid A \mid G \mid L\rangle$ en résultant $S_2 = \langle CR \cup \mathcal{E} \mid A \mid congr(G) \mid L\rangle$, où chaque règle de réécriture de $\mathcal{E}$ est obtenue par l'orientation d'une équation de $congr(G)$. La condition est satisfaite par la propriété (Prop. 3).
- Si normalize a été appliquée à une $\overline{C}$-structure $S_1 = \langle CR \mid A \mid G \mid (P \bullet IE)\rangle$, la condition est aussi satisfaite : $\overline{C}$-structure résultée a été obtenue par des remplacements d´égaux par égaux des éléments de P et IE par l'intermédiaire des règles de réécriture de CR.
- Finalement, on va montrer que la condition est aussi satisfaite si on applique augment_L sur $S_1 = \langle CR \mid A \mid G \mid (P \bullet IE)\rangle$ pour obtenir $S_2 = \langle CR \mid A \mid G \mid (P' \bullet IE \cup IE')\rangle$. Supposons que $\mathcal{E} = Oracle\_A(\langle CR \mid A \mid G \mid (P \bullet IE)\rangle)$. D'une part, on a $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \models_{ind} \mathcal{E}\tau$, pour toute substitution close τ. D'autre part, si $(P',\ IE') =$

$arith(\{\cup_{e\in\mathcal{E}} linearize(e)\} \cup P)$, on obtient comme dans le cas de A2L que $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \cup \cup_{p\in\{\cup_{e\in\mathcal{E}} linearize(e)\cup P\}}\{p = True\}\tau \models_{ind} (\cup_{p'\in P'}\{p' = True\} \cup \cup_{e\in IE'}\{e\})\tau$, pour toute substitution close τ.

On va analyser si la dernière condition est satisfaite. Soit $S = \langle CR \parallel A \parallel G \parallel (P \bullet IE)\rangle$. On suppose que **inconsistant** a été retourné par

- *congr(G).* Puisqu'il existe une diséquation de G de la forme $s \neq s$, on a $\models_{ind} \neg(\bigwedge \Phi_S)$. Par conséquent, $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \models_{ind} \neg(\bigwedge \Phi_S)\tau$, pour toute substitution close τ. Sous une forme équivalente, on peut dire que $Ax \cup \mathsf{C}^1_{\preceq_c C\tau} \cup \mathsf{C}^2_{\prec_c C\tau} \not\models_{ind} \Phi_S\tau$, pour toute substitution close τ.
- *arith(P).* Alors, il existe une inégalité impossible dans P. De la même façon comme on a procédé pour le cas précèdent, on peut déduire que la dernière condition est aussi satisfaite par S.

Fin de preuve

Théorème 6.4.2 (terminaison des dérivations en horizontale) *Toute suite d'applications des règles de la figure 6.2 termine.*

Preuve Supposons l'existence d'une suite de transformations de $\overline{C}$-structures $S_1 \xrightarrow{\mathsf{name-op}} \ldots \xrightarrow{\mathsf{name-op}} S_i \xrightarrow{\mathsf{name-op}} \ldots$ selon une stratégie arbitraire.

L'ensemble de monômes de P se modifie seulement par la réécriture avec des règles de CR. Puisque l'ordre $>_t$ est bien fondé, il existe une étape j à partir de laquelle normalize ne s'appliquera plus pour modifier cet ensemble. Soit $S_j = \langle CR \parallel A \parallel G \parallel (P \bullet IE)\rangle$ la $\overline{C}$-structure à l'étape j. Le nombre de monômes maximaux est fini et les instances de $R(C)$ qu'on peut utiliser pendant l'augmentation est aussi fini. De plus, tous les monômes des inéquations linéaires ajoutés par l'augmentation existait déjà dans des inéquations de P. Ainsi l'application d'*arith* termine sur l'ensemble augmenté d'inéquations. Par conséquent, le nombre d'équations implicites qu'on peut engendrer pendant les étapes qui suivent j est fini puisque elles établissent seulement des relations d'égalité entre de monômes de P. Soit IE' cet ensemble d'équations implicites†. D'autre part, l'application de *congr* sur $G \cup IE'$ termine aussi.

Fin de preuve

6.5 Travaux voisins

Boyer et Moore [Boyer et Moore, 1985] ont été les premiers à définir et mettre en œuvre l'heuristique d'augmentation afin d'accroître l'efficacité de l'intégration d'une procédure de (semi-)décision pour l'arithmétique linéaire dans leur démonstrateur NQTHM [Boyer et Moore, 1979]. Leur procédure de (semi-)décision peut également manipuler des disjonctions d'inéquations linéaires ainsi que des inéquations linéaires conditionnelles. Suite à de nombreuses expérimentations, ils ont conclu que la procédure de décision est très peu utilisée sans cette heuristique, tandis qu'avec elle le dégrée d'automatisation du démonstrateur s'améliore considérablement. Malheureusement, leur schéma d'intégration est informel et il n'y a pas de preuve de terminaison.

† dans le pire de cas, on considère que IE' contient toutes les relations d'égalités entre les membres des monômes de P.

Plusieurs travaux ont été inspirés du schéma d'intégration proposé par Boyer et Moore. Kapur, Musser et Nie présentent dans [Kapur et Nie, 1994] un schéma enrichi avec une procédure de décision pour la théorie de l'égalité qui est mis en œuvre dans le démonstrateur [Kapur *et al.*, 1994]. De même, aucune preuve de terminaison n'est faite. Des preuves de la terminaison de schémas similaires n'ont été données que récemment. Ainsi, dans [Janičić *et al.*, 1999] on illustre un cadre flexible d'intégration des procédures de décision. Pourtant, la condition de terminaison est une contrainte forte qui affaiblit ses performances. Encore plus général est le schéma paramétré utilisé dans une version récente d'un cadre de simplification appelé *Constraint Contextual Rewriting* [Armando et Ranise, 1998], qui permet de séparer la réécriture de la procédure de décision donnée comme paramètre. Dans [Armando et Ranise, 2000] on montre sa terminaison.

Un résultat dans le but de formaliser le schéma de Boyer et Moore a été présenté dans [Giunchiglia *et al.*, 1994] sous la forme d'un cadre de spécification qui permet la description du processus de composition des systèmes de raisonnement à l'aide des règles similaires au calcul des séquents. Son niveau logique a été décrit dans [Coglio *et al.*, 1997].

6.6 Conclusions

Nous avons présenté une procédure de (semi-)décision construite à partir d'un schéma de coopération entre une procédure de décision pour la théorie de l'égalité et une procédure de (semi-)décision pour l'arithmétique linéaire. Elle peut être facilement intégrée dans des démonstrateurs basés sur des ensembles couvrants contextuels pour lesquels elle permettrait d'engendrer des ensembles couvrants contextuels vides.

Pour des raisons d'efficacité, nous avons choisi des procédures de (semi-)décision pour les théories individuelles les plus simples. Par rapport à [Boyer et Moore, 1985], notre schéma ne permet pas le traitement des disjonctions d'inéquations linéaires et des inéquations linéaires conditionnelles. De plus, par opposition avec [Nelson, 1981], nous n'utilisons pas de variables d'abstraction. Ceci rend impossible l'application de la procédure de (semi-)décision pour l'arithmétique linéaire à des sous-termes de monômes ou de membres d'équations et d'inéquations.

Deuxième partie

Applications à la vérification de logiciels de télécommunications

7

Analyse des interactions de services téléphoniques

Sommaire

7.1 Introduction

Au cours de ces dernières années, on a assisté à une croissance continue du nombre de services de télécommunications offerts par les réseaux téléphoniques, comme les renvois d'appels, les numéros abrégés ou le filtrage d'appels. Lorsqu'on ajoute un nouveau service à un système téléphonique, on modifie le comportement global du système par son comportement, ce qui peut entraîner des interactions avec les services déjà existants. Un problème délicat est la détection des interactions incompatibles avec le comportement attendu des services ou inattendues du point de vue de l'utilisateur. Vérifier si un service produit le comportement attendu est une tâche complexe, difficile à résoudre sans l'aide d'outils automatiques.

Dans ce chapitre, on présente dans un premier temps une méthodologie de spécification de systèmes téléphoniques à un niveau abstrait et puis une étude de cas. La méthodologie permet de détecter et de résoudre *off-line* des interactions du point de vue de l'utilisateur avec des techniques basées sur la réécriture conditionnelle et la récurrence implicite. Elle repose sur une vue fonctionnelle et globale du réseau qui se modifie constamment par des séquences de commandes. La spécification des services se fait à l'aide des équations conditionnelles. L'outil

automatique qu'on va utiliser est SPIKE, présenté dans le chapitre 4. On peut distinguer deux modes de travail de SPIKE : dans le mode *réfutationnel*, il se comporte comme un model-checker pour détecter des situations conflictuelles entre les services. Normalement, on va utiliser le mode réfutationnel lorsqu'on s'attend à rencontrer des problèmes dans des configurations de systèmes téléphoniques de petite taille. L'exploration de l'espace de recherche se fait facilement par un mécanisme d'instanciation graduelle. Dans le mode *positif*, le prouveur peut vérifier des invariants des systèmes potentiellement infinis par des procédures de preuve grâce au mécanisme de récurrence implicite de SPIKE.

La structure du chapitre est la suivante. On présente en premier lieu le problème d'interactions par deux exemples typiques. Dans la section 7.2, on définit formellement deux classes d'interactions de services. La section 7.3 contient un bref survol de travaux similaires et d'approches basées sur d'autres outils (semi)automatiques, langages de spécification et techniques de détection et résolution. Notre méthodologie pour la détection et la résolution des interactions de services est développée dans la section 7.5. Elle se repose sur les spécifications formelles du réseau et des services introduites avant dans la section 7.4. Ensuite, la section 7.6 décrit dans une étude de cas l'application de la méthodologie lorsqu'on compose les spécifications de deux services donnés. Le chapitre se termine par des remarques techniques sur la vérification de cet exemple. Dans l'annexe A, le lecteur peut consulter la spécification composée obtenue avec notre méthodologie.

Exemples de services et d'interactions de services

Les services utilisés dans les exemples tout au long de ce chapitre sont :

- CFU (Call Forward Unconditional) - le *renvoi inconditionnel des appels*. Tous les appels à un abonné donné sont renvoyés à un certain numéro téléphonique.
- TCS (Terminating Call Screening) - le *filtrage des appels à leur arrivée à destination*. L'abonné doit refuser toute connexion en provenance des postes dont le numéro téléphonique se trouve dans une liste donnée, appelée *liste noire*.
- ABD (ABbreviated Dialling) - la *numérotation abrégée*. Des petits nombres, à quelques chiffres, sont traduits en numéros téléphoniques. Ces numéros s'appellent des *numéros abrégés*.
- OCS (Originating Call Screening) - le *filtrage des appels au départ*. L'abonné ne peut pas appeler les postes dont les numéros se trouvent dans une liste noire gérée par l'abonné.

Exemple 7.1.1 *Un premier exemple présente une interaction due à l'inter-fonctionnement de CFU et TCS. Elle peut être détectée par une approche à haut niveau, basée sur l'analyse des propriétés de ces services. On suppose que CFU et TCS sont disponibles dans un réseau à trois utilisateurs. Le deuxième utilisateur est abonné à CFU et renvoie tous les appels reçus vers le troisième utilisateur qui, à son tour, est abonné à TCS et a le premier utilisateur dans sa liste noire. L'interaction se passe au moment où le premier utilisateur appelle le deuxième : grâce au CFU, l'appel est ainsi renvoyé vers le troisième et finalement la connexion s'établit entre le premier et le troisième. Ceci est en contradiction avec la spécification du service TCS.*

Exemple 7.1.2 *L'exemple suivant montre une interaction de service qui peut être détectée par une approche à bas niveau, plus précisément, par l'analyse du comportement des services. On suppose qu'un utilisateur est abonné à la fois à ABD et OCS. Si celui-ci appelle un numéro abrégé se trouvant dans la liste noire de OCS, on peut avoir deux comportements différents selon*

les priorités d'exécution des deux services lorsque le petit nombre associé à ce numéro abrégé est composé ; si ABD est prioritaire, l'appel est annulé. Sinon, comme le petit nombre ne se trouve pas dans la liste noire, la connexion va s'établir.

7.2 Définitions formelles des interactions de services

Les spécifications de services téléphoniques doivent décrire leur *comportement*, ainsi que leurs *propriétés* attendues.

Par son comportement, tout nouveau service ajouté va modifier le comportement global du système téléphonique. On suppose que les services sont ajoutés de manière *incrémentale* à un service de base, nommé $POTS$ (Plain Old Telephone System), qui intègre les fonctionnalités primaires du système téléphonique. Formellement, soient deux descriptions de services, F_1 et F_2, et leur ensemble de propriétés, Φ_1 et Φ_2, tels que $F_1/POTS \models_{ind^*} \Phi_1$ et respectivement $F_2/POTS \models_{ind^*} \Phi_2$, où $F_i/POTS$ est l'ensemble d'axiomes de la spécification logique du service F_i ($i \in [1..2]$) qui est ajoutée au $POTS$. Alors, il n'y a pas d'interaction si on peut déduire $(F_1 \oplus F_2)/POTS \models_{ind^*} \Phi_1 \cup \Phi_2$, où $(F_1 \oplus F_2)$ est la *spécification composée* des services F_1 and F_2.

Généralement, on distingue deux classes d'interactions : certaines sont dues au non-déterminisme du comportement global du système engendré par la composition des comportements individuels des services adjacents. Elles sont similaires à l'interaction entre OCS et ABD de l'exemple 7.1.2. Les autres interactions se distinguent lorsque la spécification composée ne satisfait pas les propriétés de (ou n'est pas *conforme* à) chaque spécification de service à part, comme le scénario présenté dans l'exemple 7.1.1 l'a montré pour la combinaison des services CFU et TCS.

Le fonctionnement du système téléphonique est modélisé, en général, soit par un système distribué formé par un ensemble de processus distribués et communicants, soit comme un automate qui décrit le comportement du système global, composé par les automates décrivant les comportements de chacun de ses services. Dans le deuxième cas, le comportement d'un service est indiqué sous la forme de *séquences calculables* d'états de l'automate associé, qui sont construites par l'exécution d'un ensemble de transitions simples à partir d'un état initial.

Dans la suite, on va définir formellement les deux classes d'interactions. Pour la première, Bredereke [Bredereke, 1995; Bredereke, 1996] donne la définition suivante :

Définition 7.2.1 *(interaction due au non-déterminisme) Soit A l'automate qui décrit le comportement global d'un système téléphonique. On dit que le service f interagit avec le service g si i) $f \neq g$, et ii) il existe une séquence calculable c^f qui contient au moins une f-transition simple, et iii) une séquence calculable c^g qui contient au moins une g-transition simple, et iv) un index $i \in \mathbb{N}$ tel que*

1. $< c_0^f, \ldots, c_i^f > = < c_0^g, \ldots, c_i^g >$, *et*
2. *la transition simple (c_i^f, c_{i+1}^f) appartient seulement à l'automate décrivant le comportement de f, et*
3. *la transition simple (c_i^g, c_{i+1}^g) appartient seulement à l'automate décrivant le comportement de g.*

Alors, on a un non-déterminisme du comportement global qui se produit au point de choix i. De tels points de choix indiquent des interactions potentielles à bas niveau entre les fonctionnements des deux services.

Introduisons la deuxième classe d'interactions. Soit L un ensemble de *commandes* représentant des actions observables du point de vue de l'utilisateur. On va dénoter i) par $\mathbb{P}(L)$ l'ensemble de *traces*, représentant des séquences calculables de commandes, et ii) par $Trace(G)$, l'ensemble de traces de l'automate G décrivant le comportement d'un service téléphonique.

Définition 7.2.2 *(interaction due à la non-conformité) Soient I et C deux automates décrivant respectivement les fonctionnements des services f et g. On dit que I est* conforme *à C si*

$$\forall \sigma \in Trace(C),\ \forall A \subseteq L,\ si\ Accepte(C,\ \sigma,\ A)\ alors\ Accepte(I,\ \sigma,\ A),$$

où $Accepte(G,\ T,\ A)$ est vrai si l'automate G accepte la trace T pour l'ensemble d'actions observables A. Alors il y a une interaction au niveau des propriétés des services f et g si I n'est pas conforme à C.

Le concept d'interaction de services est plus général et complexe. Les deux classes d'interactions qu'on a présentées ne sont pas adéquates, par exemple, au traitement des aspects non-fonctionnels des services (en particulier ceux qui concernent la performance, le temps réel ou les propriétés de sécurité) dont l'analyse formelle est plus difficile à réaliser.

7.3 Etat de l'art

Spécification et détection

Les outils automatisés sont principalement utilisés dans l'analyse *off-line* des services. Ceci ne serait pas possible sans des descriptions formelles modélisant le système téléphonique, ainsi que ses services. La méthodologie standard pour la détection des interactions requiert deux telles descriptions : l'une est faite dans un langage qui permet d'exprimer le comportement des services, comme SDL [Turner, 1993] dans [Faci et Logrippo, 1994], Estelle [ISO/TC 97/SC 21, 1989] dans [Bredereke et Gotzhein, 1995], LOTOS [ISO/TC 97/SC 21, 1988] dans [Boumezbeur et Logrippo, 1993; Stepien et Logrippo, 1995] ou Promela [Holzmann et Peled, 1995] dans [Lin et Lin, 1994]. L'autre description utilise un langage de plus haut niveau, assez expressif pour spécifier leurs propriétés. Celui-ci est, en général, une variante de la logique de premier ordre, comme Prolog [Clocksin et Mellish, 1981] dans [Frappier *et al.*, 1996], ou de la logique temporelle, comme la logique d'actions de Lamport (TLA) [Lamport, 1991] dans [Blom *et al.*, 1994]. Il y a des langages de spécification suffisamment puissantes pour exprimer les deux descriptions à la fois, comme B [Abrial, 1996] dans [Mermet *et al.*, 1998; Cansell et Méry, 2000] ou les spécifications conditionnelles†. D'autre part, la technique de vérification largement utilisée dans la détection des interactions est le model-checking.

Dans l'article de Lin et Lin [Lin et Lin, 1994], les comportements de services sont écrits en Promela tandis que les propriétés dans un langage basé sur la logique temporelle. L'outil SPIN vérifie les propriétés par model-checking.

A l'aide de l'outil GEODE [Encontre, 1989], Combes et Pickin [Combes et Pickin, 1994] vérifient par model-checking des propriétés exprimées en logique temporelle. Pour cela, les formules logiques sont traduites en diagrammes de séquences de messages (MSC) [ITU-T, 1993b]. Les descriptions de bas niveau sont codées en SDL.

† A voir notre méthodologie présentée dans la section 7.5.

Braithwaite et Atlee [Braithwaite et Atlee, 1994] utilisent une technique de spécification semi-formelle, basée sur une notation tabulaire, pour vérifier des propriétés liées aux automates, comme le non-déterminisme. Cette méthode permet aussi la découverte des violations des propriétés spécifiées explicitement par l'utilisateur.

LOTOS est un langage de spécification largement utilisé pour décrire les comportements des services. Bouma et Zuidweg [Bouma et Zuidweg, 1993] expriment les propriétés des services dans une logique temporelle ramifiée et utilisent manuellement l'outil LITE pour faire du model-checking. Il existe aussi des variantes plus automatiques de cette méthodologie grâce à l'outil Caesar/Aldébaran [Fernandez *et al.*, 1992] dans [Korver, 1993]. Une approche similaire plus récente est celle proposée par Thomas [Thomas, 1998] qui emploie le μ-calcul modal comme langage de spécification des propriétés.

Les méthodes de détection *on-line*, appliquées lorsque les services sont activés, sont moins nombreuses et plutôt informelles. [Tsang et Magill, 1998] entraînent des réseaux neuronaux pour apprendre le comportement de chaque service séparément. La modification du comportement d'un service, lors de son inter-fonctionnement avec d'autres services, sera détectée par le réseau neuronal qui a « appris à reconnaître » le fonctionnement correct du service.

Les étapes de modélisation et de spécification sont souvent allégées par des méthodes informelles, basées sur la reconnaissance et le traitement du langage naturel. Le système GATOR [Dankel *et al.*, 1994] traduit semi-automatiquement des textes en langage naturel en ensembles de prédicats. L'utilisateur doit lui fournir des explications sur les termes inconnus, par rapport à une base de connaissance formée par des prédicats obtenus de manière similaire, et donner des précisions pour éliminer les ambiguïtés du texte. Ces ambiguïtés représentent des interactions potentielles. Dans [Charnois, 1997], la base de connaissance est représentée sous la forme d'une composition de graphes conceptuels, chaque service étant décrit par un tel graphe.

Résolution

Généralement, il existe deux classes de méthodes pour résoudre les interactions de services. La plus classique est la résolution par *restriction*, lorsqu'un service ou certaines de ces fonctionnalités sont suspendus. Le choix du service à suspendre se fait à l'aide d'une table de priorités entre les services [Bredereke et Gotzhein, 1994], ou bien par un langage de spécification comme Delphi [Boström et Engstedt, 1995]. Fritsche [Fritsche, 1995] propose un gestionnaire d'interactions qui contrôle des messages passés entre les services et le réseau par l'intermédiaire d'une matrice contenant les conditions selon lesquelles un service est suspendu, altéré ou remis en marche. Une autre technique de résolution *on-line* plus élaborée est MCP (Modular Control with Priorities) [Chen *et al.*, 1997]. Elle permet l'interdiction des fonctionnalités jugées indésirables par des « surveillants » qui supervisent le comportement de chaque service. Encore plus puissante est celle proposé par [ITU-T, 1993a] : les gestionnaires d'interactions FIM (Feature Interactions Manager) sont des arbitres entre les services et le réseau. Aggoun et Combes [Aggoun et Combes, 1997] utilisent respectivement des observateurs passifs et actifs pour la résolution *off-line* et *on-line*. Les observateurs actifs doivent être capable de résoudre les situations conflictuelles en temps réel.

L'autre méthode de résolution est basée sur la *coopération* entre les services pour trouver un compromis dans les cas critiques. Iraqi et Erradi [Iraqi et Erradi, 1997] ont proposé un protocole qui permet d'éviter les interactions par échange d'informations entre les services. Griffeth et Velthuijsen [Griffeth et Velthuijsen, 1994] utilisent des agents qui négocient entre eux pour

trouver, en suivant une hiérarchie de buts, un système d'opérations pour faire communiquer un ensemble de téléphones.

7.4 Modélisation de services téléphoniques

Nous avons choisi de modéliser le fonctionnement du système téléphonique par un automate global et de formaliser le comportement et les propriétés des services téléphoniques par des systèmes de réécriture conditionnelle. L'ensemble de commandes à considérer pendant la spécification d'un certain service dépend de la *granularité* de la spécification qui définit le niveau d'abstraction choisi par le concepteur du service. Par exemple, une commande de connexion entre deux utilisateurs peut être vue comme une suite de commandes conforme au scénario suivant : n_1 décroche le récepteur, il compose le numéro de n_2 qui répond si la ligne n'est pas occupée. Au niveau d'abstraction le plus élevé de la modélisation, on pourrait s'intéresser seulement à savoir si n_1 est en connexion ou pas avec n_2, sans se préoccuper de la manière dont n_1 se connecte avec n_2.

Une spécification de service contient trois composantes :

- la composante de *contrôle*, constituée principalement par des fonctions décrivant le comportement du service,
- la composante *logique*, formée par des prédicats capturant les propriétés attendues par rapport au comportement du service, et
- la composante d'*interfaçage* qui spécifie les structures de données et les fonctions partagées avec les spécifications des autres services lors de leur composition.

La spécification d'un service est définie par la structure

$$Spec =< S,\, Srew,\, r >$$

où S est un ensemble de symboles de sortes, $Srew$ son système de réécriture conditionnelle et r l'ensemble de fonctions qui permettra de corréler l'information partagée avec d'autres spécifications.

Maintenant, on va détailler le contenu de la structure *Spec*. L'ensemble S doit inclure au moins les symboles de sortes suivants : Com pour les commandes, N pour les utilisateurs, Nat pour les numéros de téléphone, $Bool$ pour les booléens, $\mathbb{P}(N)$ pour les listes d'utilisateurs et $\mathbb{P}(Com)$ pour des listes de commandes.

Le système de réécriture $Srew$ est divisé en deux. La **composante de contrôle** est définie par une seule fonction spéciale, dénotée par $Apply\_network$ et déclarée comme

$$\boxed{Apply\_network : Com \times \mathbb{P}(N) \rightarrow \mathbb{P}(N)}$$

Cette fonction est chargée de décrire le nouvel état global du réseau téléphonique lorsqu'une commande s'applique dans le réseau. Dans notre approche, l'état global d'un réseau téléphonique est décrit par l'état de chaque utilisateur individuel. Formellement, si Tel_i définit l'état de l'utilisateur i, alors la configuration du réseau est la liste $[Tel_1, \ldots, Tel_n]$, où n est le nombre d'utilisateurs dans le réseau.

La **composante logique** est formée par un ensemble de prédicats concernant la satisfaction des propriétés du service du point de vue de l'utilisateur après l'exécution d'une liste de commandes dans le réseau. Chaque prédicat $Prop$ est déclaré comme

$$\boxed{Prop : \mathbb{P}(Com) \times \mathbb{P}(N) \rightarrow Bool}$$

Par exemple, un tel prédicat peut capturer l'intention d'un abonné au service TCS de ne pas être appelé par des utilisateurs mentionnés dans sa liste noire.

La **composante d'interfaçage** comporte des fonctions d'*identification* et de *translation* qui interprètent les symboles de fonctions (constructeurs ou fonctions définies) de la spécification composée dans la spécification du service.

Pour des raisons esthétiques, on va représenter une règle de réécriture conditionnelle $(\bigwedge_{i=1}^{n} a_i = b_i) \Rightarrow lhs \rightarrow rhs$ sous la forme du séquent

$$\frac{lhs}{rhs} < nom\_règle, (\bigwedge_{i=1}^{n} a_i = b_i) > ,$$

où *nom_règle* est l'identificateur de la règle.

7.5 Méthodologie pour détecter et résoudre des interactions de services

Notre méthode générale pour détecter et résoudre des interactions de services téléphoniques est basée sur des propriétés des systèmes de réécriture construits à partir de spécifications conditionnelles. Les règles de réécriture conditionnelles sont suffisamment expressives pour décrire à la fois le comportement et les propriétés des services dans une même spécification. La technique de vérification s'appuie sur la réécriture et sur des mécanismes de preuve par récurrence implicite, décrits dans le chapitre 4.

La méthodologie est la suivante :

M-1. proposer des systèmes de réécriture conditionnelle suffisamment complets et convergents sur les termes clos qui spécifient chaque service à part †.

M-2. construire les interfaces de spécifications, par la définition des structures de données et des fonctions qui seront partagées.

M-3. combiner, via ces interfaces, les spécifications de services afin d'obtenir une première version de la spécification composée.

M-4. tester la propriété de complétude suffisante de la spécification composée et compléter les fonctions qui ne sont pas complètement définies.

M-5. dans la spécification composée, détecter les points de choix non-déterministe et résoudre les interactions potentielles non-déterministes en introduisant des précédences sur les actions. L'existence des points de choix est attestée si la spécification composée n'est pas convergente sur les termes clos. Retourner à M-4 si nécessaire.

M-6. détecter les interactions dues à la non-conformité en utilisant SPIKE en mode réfutationnel, ou bien certifier leur absence par la preuve des invariants en mode positif.

M-7. résoudre les interactions potentielles et retourner à M-4 si nécessaire.

Si l'application de la méthodologie finit avec succès, le résultat ainsi obtenu est une spécification des services à composer qui est suffisamment complète, convergente sur les termes clos. De plus, elle certifie que l'inter-fonctionnement de ces services est conforme avec le comportement de chacun des services à part.

Les six premières étapes de la méthodologie, instanciée pour analyser l'inter-fonctionnement de *deux* services, sont schématisées dans la figure 7.1.

† Cette hypothèse est motivée par le fait que les services sont normalement conçus indépendamment les uns des autres.

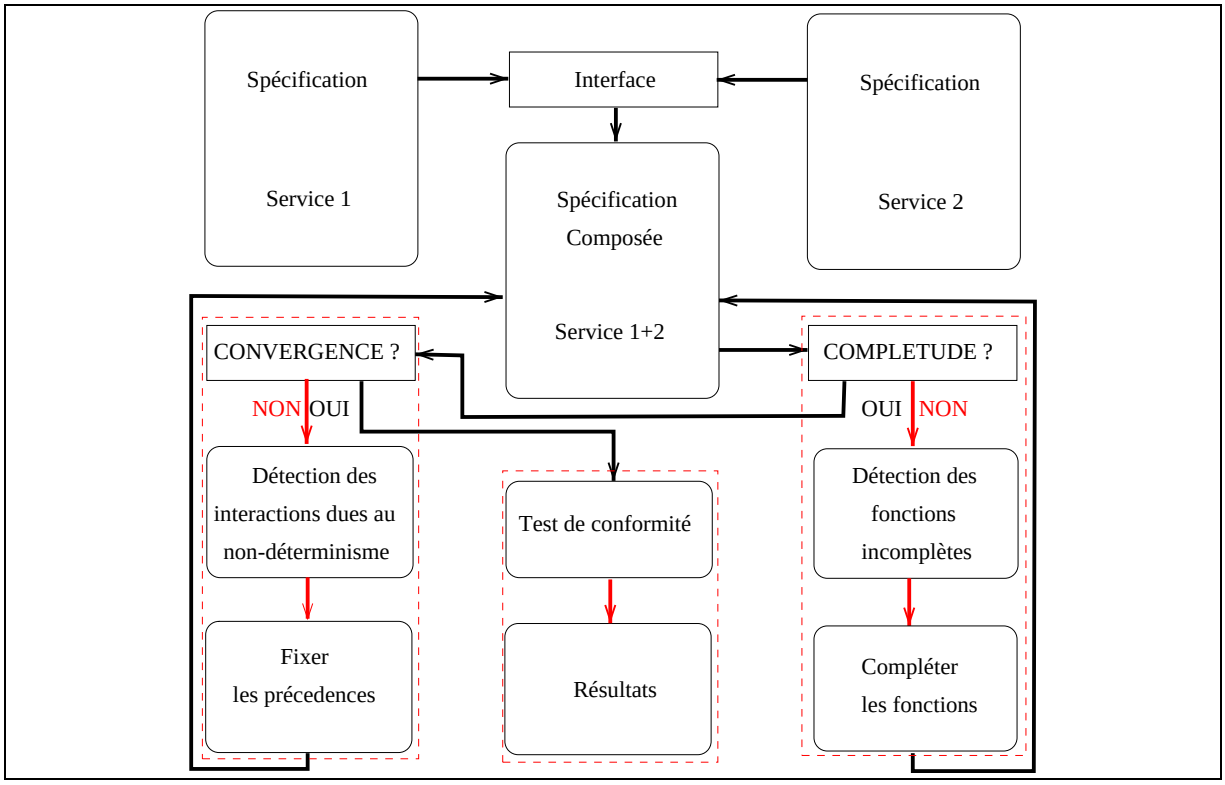

FIG. 7.1 – *L'analyse et la résolution des interactions de deux services avec notre méthodologie*

7.6 Application à l'analyse de l'inter-fonctionnement des services CFU et TCS

Dans cette section, on va appliquer la méthodologie présentée dans la section 7.5 pour détecter et résoudre les interactions potentielles issues de l'inter-fonctionnement des services CFU et TCS.

7.6.1 Spécification du service CFU

Les sortes :

Les sortes principales de la spécification du service CFU sont utilisées pour décrire des

- *utilisateurs*, N^C. Le symbole de constructeur de cette sorte est Tel^C.
- *commandes*, Com^C. Les symboles de constructeurs de cette sorte sont $COMM^C$ et CFU^C.
- *identificateurs d'utilisateur* (ou de numéros de téléphone), Nat^C. Les symboles de constructeur de sorte Nat^C sont 0^C et s^C.

Autres sortes sont : $Bool^C$ pour les booléens, avec les constructeurs $True^C$ et $False^C$, $\mathbb{P}(Nat^C)$ pour des listes of numéros, $\mathbb{P}(N^C)$ pour l'état global du système téléphonique et $\mathbb{P}(Com^C)$ pour des listes de commandes.

La signification des constructeurs Tel^C, $COMM^C$ et CFU^C est respectivement :

- $Tel^C(n_1,\ n_2,\ n_3) : Nat^C \times Nat^C \times Nat^C \rightarrow N^C$ décrit l'état de l'utilisateur n_1. Par exemple, il nous permet de savoir si l'utilisateur est connecté à l'utilisateur n_2 ou bien il renvoie à n_3 les appels reçus.
- $COMM^C(n_1,\ n_2) : Nat^C \times Nat^C \rightarrow Com^C$ spécifie que n_1 essaie de se connecter à n_2.
- $CFU^C(n_1,\ n_2) : Nat^C \times Nat^C \rightarrow Com^C$ indique que le service CFU^C est activé pour l'utilisateur n_1 et que les appels sont renvoyés vers n_2.

Généralement, un utilisateur est inactif dans une commande si son numéro de téléphone a la valeur 0. Par exemple, $COMM^C(n_1,\ 0)$ signifie que n_1 a fini sa communication avec n_2 et $Tel^C(n_1,\ n_2,\ 0)$ que le service CFU est désactivé pour l'utilisateur n_1.

Les règles de réécriture :

La fonction $Apply\_network^C(c,\ r) : Com^C \times \mathbb{P}(N^C) \rightarrow \mathbb{P}(N^C)$ de la composante de contrôle décrit le comportement du CFU lorsqu'un appel est reçu par l'abonné. Parmi les trois règles qui la définissent : $Apply_1^C$, $Apply_2^C$ et $Apply_3^C$, le cas le plus intéressant est formalisé par la règle $Apply_3^C$; la commande $COMM^C(n_1,\ n_2)$, représentant l'appel de n_1 vers n_2, est appliquée si l'utilisateur n_2 a le service CFU activé et il existe un numéro auquel tout appel vers n_2 est renvoyé. Elle instancie n_2 avec ce numéro. Ensuite, une nouvelle commande $COMM^C$ est traitée par une fonction secondaire $Apply\_aux^C : Com^C \times \mathbb{P}(N^C) \rightarrow \mathbb{P}(N^C)$ détaillant le comportement du service. Pour les autres cas, les arguments de la commande restent inchangés et la commande sera traitée par $Apply\_aux^C$, selon $Apply_1^C$ et $Apply_2^C$. La fonction $transit^C(n_1,\ r) : Nat^C \times \mathbb{P}(N^C) \rightarrow Nat^C$ retourne 0 si le service CFU n'est pas activé pour l'utilisateur avec le numéro n_1, sinon elle représente le numéro où l'appel sera renvoyé.

$$\frac{Apply\_network^C(CFU^C(n_1,\ n_2),\ r)}{Apply\_aux^C(CFU^C(n_1,\ n_2),\ r)} < Apply_1^C,\ (True) >$$

La règle de réécriture

$$\frac{Apply\_network^C(COMM^C(n_1,\ n_2),\ r)}{Apply\_aux^C(COMM^C(n_1,\ n_2),\ r)} < Apply_2^C,\ c >$$

est appliquée si la condition $c \equiv (transit^C(n_2,\ r) = 0 \wedge n_1 \neq n_2)$ est satisfaite.

$$\frac{Apply\_network^C(COMM^C(n_1,\ n_2),\ r)}{Apply\_aux^C(COMM^C(n_1,\ transit^C(n_2,\ r)),\ r)} < Apply_3^C,\ cond >$$

où la condition $cond \equiv (transit^C(n_2,\ r) \neq 0 \wedge n_1 \neq n_2)$.

Toutes les commandes $COMM(n_1,\ n_1)$ sont filtrées par $Apply_4^C$:

$$\frac{Apply\_network^C(COMM^C(n_1,\ n_2),\ r)}{r} < Apply_4^C,\ (n_1 = n_2) >$$

La fonction $Apply\_aux : Com^C \times \mathbb{P}(N^C) \rightarrow \mathbb{P}(N^C)$ est définie par les règles étiquetées par App_i, avec $i \in [1..6]$:

$$\frac{Apply\_aux^C(c,\ [])}{[]} < App_1^C,\ (True) >$$

La règle App_1^C traite la situation triviale lorsque la commande est exécutée sur une configuration vide du système global. App_2^C applique la commande CFU^C à l'état de chaque utilisateur comme

c'est décrit par la fonction App_tel^C. Déclarée par $App\_tel^C(c,\, t) : Com^C \times N^C \rightarrow N^C$, elle applique la commande c sur l'état courant de l'utilisateur t. Cette fonction est définie dans l'annexe A.

$$\frac{Apply\_aux^C(CFU^C(n_1,\, n_2),\, [t@r])}{[App\_tel^C(CFU^C(n_1,\, n_2),\, t)@Apply\_aux^C(CFU^C(n_1,\, n_2),\, r)]} < App_2^C,\, c_1 > ,$$

avec $c_1 \equiv (n_1 \neq 0 \wedge \neg busy^C(n_1,\, [t@r]))$.

Le prédicat $busy^C : Nat^C \times \mathbb{P}(N^C) \rightarrow Bool^C$ vérifie si un utilisateur est connecté :

$$\frac{busy^C(n,\, [])}{False^C} < busy_1^C,\, (True) >$$

$$\frac{busy^C(n,\, [Tel^C(n_1,\, n_2,\, n_3)@r])}{True^C} < busy_2^C,\, c_2 >$$

où $c_2 \equiv (n_1 = n \wedge n_2 \neq 0)$.

$$\frac{busy^C(n,\, [Tel^C(n_1,\, n_2,\, n_3)@r])}{busy^C(n,\, r)} < busy_3^C,\, \neg c_2 >$$

La règle App_3^C filtre les commandes CFU^C lorsqu'elles ne satisfont pas c_1.

$$\frac{Apply\_aux^C(CFU^C(n_1,\, n_2),\, [t@r])}{[t@r]} < App_3^C,\, \neg c_1 >$$

Le cas traitant la connexion de deux utilisateurs est introduit par App_4^C. App_5^C couvre le cas de la déconnexion d'un utilisateur.

$$\frac{Apply\_aux^C(COMM^C(n_1,\, n_2),\, [t@r])}{connect^C(n_1,\, n_2,\, [t@r])} < App_4^C,\, c_3 >$$

La condition c_3 est $(n_1 \neq 0 \wedge n_2 \neq 0)$.

$$\frac{Apply\_aux^C(COMM^C(n_1,\, n_2),\, [t@r])}{disconnect^C(n_1,\, [t@r])} < App_5^C,\, c_4 > ,$$

où la condition c_4 est $(n_1 \neq 0 \wedge n_2 = 0)$.

La fonction $connect^C(n_1,\, n_2,\, r) : Nat^C \times Nat^C \times \mathbb{P}(N^C) \rightarrow \mathbb{P}(N^C)$ connecte l'utilisateur n_1 avec n_2 (si possible) tandis que la fonction $disconnect^C(n,\, r) : Nat^C \times \mathbb{P}(N^C) \rightarrow \mathbb{P}(N^C)$ déconnecte n du réseau r. Ces fonctions sont aussi détaillées dans l'annexe A.

La règle App_6^C filtre les commandes COM^C lorsque le numéro n_1 est désactivé :

$$\frac{Apply\_aux^C(COMM^C(n_1,\, n_2),\, [t@r])}{[t@r]} < App_6^C,\, (n_1 = 0) >$$

La composante logique est formée par un seul prédicat, $no\_selfcall : N^C \rightarrow Bool^C$, qui est vrai si l'utilisateur ne s'appelle pas par lui-même.

$$\frac{no\_selfcall(Tel^C(n_1,\, n_2,\, n_3))}{True^C} < no\_selfcall_1,\, (n_1 \neq n_2) >$$

$$\frac{no\_selfcall(Tel^C(n_1,\, n_2,\, n_3))}{False^C} < no\_selfcall_2,\, (n_1 = n_2) >$$

Les fonctions d'identification et de translation constituant la composante d'interfaçage sont détaillées dans la section 7.6.3.

Le système de réécriture ainsi construit est suffisamment complet et convergent sur les termes clos.

7.6.2 Spécification du service TCS

Les sortes :

Les sortes de la spécification du service TCS sont similaires aux celles de la spécification du service CFU de la section 7.6.1 :

- *utilisateurs*, N^T. Le symbole de constructeur de cette sorte est Tel^T.
- *commandes*, Com^T. Les symboles de constructeurs de cette sorte sont $COMM^T$ et TCS^T.
- *identificateurs d'utilisateur* (ou de numéros de téléphone), Nat^T. Les symboles de constructeur de sorte Nat^T sont 0^T et s^T.

$Bool^T$, $\mathbb{P}(Nat^T)$, $\mathbb{P}(N^T)$ et $\mathbb{P}(Com^T)$ sont respectivement les sortes pour les booléens, les listes de numéros, l'état global du réseau téléphonique et les listes de commandes.

La signification des constructeurs Tel^T, $COMM^T$ et TCS^T est :

- $Tel^T(n_1, n_2, ln, f) : Nat^T \times Nat^T \times \mathbb{P}(Nat^T) \times Bool^T \rightarrow N^T$ décrit l'état de l'utilisateur n_1. ln est la liste noire associée au service TCS et n_2 est l'utilisateur à qui n_1 est connecté. Lorsque l'argument f est $True^T$, n_1 est à l'origine de l'appel.
- $COMM^T(n_1, n_2) : Nat^T \times Nat^T \rightarrow Com^T$ indique que n_1 essaie de se connecter à n_2.
- $TCS^T(n_1, n_2) : Nat^T \times Nat^T \rightarrow Com^T$ spécifie que l'utilisateur qui possède le numéro téléphonique n_2 ajoute n_1 dans sa liste noire.

Le symbole 0 a la même signification comme pour le service CFU. De plus, on convient que si la liste noire est [], alors le service TCS est inactive pour le propriétaire de la liste.

Les règles de réécriture :

La fonction de la composante de contrôle, $Apply\_network^T : Com^T \times \mathbb{P}(N^T) \rightarrow \mathbb{P}(N^T)$, est définie comme suit. App_1^T est similaire à App_1^C.

$$\frac{Apply\_network^T(c, [])}{[]} < App_1^T, (True) >$$

App_2^T, définie ci-dessous, traite le cas où la commande TCS^T est exécutée.

$$\frac{Apply\_network^T(TCS^T(n_1, n_2), [t@r])}{[App\_tel^T(TCS^T(n_1, n_2), t)@Apply\_network^T(TCS^T(n_1, n_2), r)]} < App_2^T, d_1 >$$

est valide lorsque la condition $d_1 \equiv (n_1 \neq 0 \wedge n_2 \neq 0 \wedge \neg busy(n_2, [t@r]))$ est satisfaite.

Le prédicat $busy^T : Nat^T \times \mathbb{P}(N^T) \rightarrow Bool^T$ est similaire à $busy^C$. Le règles App_3^T et App_6^T traitent les cas où les commandes TCS^T et $COMM^T$ sont filtrées.

$$\frac{Apply\_network^T(TCS^T(n_1, n_2), [t@r])}{[t@r]} < App_3^T, \neg d_1 >$$

Les règles App_4^T et App_5^T sont similaires à App_4^C et App_5^C, respectivement.

$$\frac{Apply\_network^T(COMM^T(n_1, n_2), [t@r])}{connect^T(n_1, n_2, [t@r])} < App_4^T, d_2 > ,$$

où d_2 est $(n_1 \neq 0 \wedge n_1 \neq n_2 \wedge n_2 \neq 0)$.

$$\frac{Apply\_network^T(COMM^T(n_1,\, n_2),\, [t@r])}{disconnect^T(n_1,\, [t@r])} < App_5^T,\, d_3 > ,$$

où la condition est $d_3 \equiv (n_1 \neq 0 \wedge n_2 = 0)$.

$$\frac{Apply\_network^T(COMM^T(n_1,\, n_2),\, [t@r])}{[t@r]} < App_6^T,\, \neg(d_3 \vee d_2) >$$

Les fonctions $App\_tel^T : Com^T \times N^T \rightarrow N^T$, $connect^T : Nat^T \times Nat^T \times \mathbb{P}(N^T) \rightarrow \mathbb{P}(N^T)$ et $disconnect^T : Nat^T \times \mathbb{P}(N^T) \rightarrow \mathbb{P}(N^T)$ ont des significations similaires à App_tel^C, $connect^C$ et $disconnect^C$, respectivement.

La composante logique est définie par le prédicat

$$no\_call\_black\_list(Tel^T(n_1,\, n_2,\, ln,\, f)) : N^T \rightarrow Bool^T,$$

qui est valide lorsque l'utilisateur n_2 n'est pas dans la liste noire ln de l'utilisateur n_1 :

$$\frac{no\_call\_black\_list(Tel^T(n_1,\, n_2,\, ln,\, f))}{True^T} < no\_call\_bl_1,\, (\neg Member^T(n_2,\, ln)) >$$

$$\frac{no\_call\_black\_list(Tel^T(n_1,\, n_2,\, ln,\, f))}{f} < no\_call\_bl_2,\, (Member^T(n_2,\, ln)) >$$

Le prédicat $Member^T(n,\, l) : Nat^T \times \mathbb{P}(Nat^T) \rightarrow Bool^T$ vérifie si un numéro téléphonique se trouve dans la liste l.

Le système de réécriture ainsi construit est suffisamment complet et convergent sur les termes clos.

La composante d'interfaçage est détaillée dans la section 7.6.3.

7.6.3 Spécification composée de CFU et TCS

Afin de composer les deux spécifications de services, CFU et TCS, présentées respectivement dans les sections 7.6.1 et 7.6.2, il faut définir en premier lieu les nouvelles sortes de la spécification composée, corrélées aux sortes des spécifications de CFU et TCS par les fonctions d'identification et de translation. Une *fonction d'identification* met en correspondance un symbole de fonction de la spécification composée avec un symbole de fonction de la spécification d'un service. D'autre part, une *fonction de translation* de la spécification d'un service interprète une fonction g de la spécification composée avec une fonction de la spécification du service dont les arguments sont strictement inclus dans l'ensemble d'arguments de g. Pour distinguer les nouveaux symboles de sortes et de fonctions dans la spécification composée, on leur adjoint un M en exposant.

Les sortes des numéros téléphoniques et des booléens sont similaires au cas des spécifications de services présentée précédemment. Les symboles de constructeurs de la sorte Nat^M sont 0^M et s^M auquels on associe les 0- et s-constructeurs des spécifications de TCS et CFU par des fonctions d'identification idf^C et idf^T : $idf^C(0^M) = 0^C$, $idf^T(0^M) = 0^T$, $idf^C(s^M) = s^C$ et $idf^T(s^M) = s^T$. On procède de la même manière pour la sorte $Bool^M$.

L'ensemble de symboles de constructeurs de sorte Com^M est $\{CFU^M, COMM^M, TCS^M\}$. Les relations avec les autres constructeurs sont : $idf^C(CFU^M) = CFU^C$, $idf^T(TCS^M) = TCS^T$, $idf^C(COMM^M) = COMM^C$ et $idf^T(COMM^M) = COMM^T$.

Enfin, l'ensemble de symboles de constructeurs de sorte N^M est $\{Tel^M\}$, avec $Tel^M : Nat^M \times Nat^M \times Nat^M \times \mathbb{P}(Nat^M) \times Bool^M \rightarrow N^M$, tel que

$$\begin{cases} idt^C(Tel^M(n_1,\, n_2,\, n_3,\, ln,\, f)) = & Tel^C(n_1,\, n_2,\, n_3) \\ idt^T(Tel^M(n_1,\, n_2,\, n_3,\, ln,\, f)) = & Tel^T(n_1,\, n_2,\, ln,\, f). \end{cases}$$

$\mathbb{P}(Nat^M)$, $\mathbb{P}(N^M)$ et $(\mathbb{P}(Com^M)$ représentent respectivement les sortes qui désignent des listes de numéros, l'état global du système téléphonique et des listes de commandes.

La vérification de la propriété de complétude suffisante

La première étape est d'étendre de manière *conservative* les spécifications de chacun des services par rapport aux sortes de la spécification composée.

Le vérification de la propriété de complétude suffisante d'un système de réécriture conditionnelle basé sur des constructeurs libres peut se faire à l'aide des *arbres de motifs* [Bouhoula, 1994] associés à chaque fonction définie. Dans la figure 7.2, on montre la complétude de la fonction $Apply\_network^T$ de la spécification du service TCS.

L'arbre de motifs a la propriété que les termes de ses feuilles (dénotées par OK dans la figure) sont des (instances de) parties gauches des règles de réécriture et que la disjonction des conditions associées à chaque feuille est une conséquence initiale des axiomes de la spécification. Pour chaque (instance d'une) telle règle $(\bigwedge_{i=1}^{n} C_i) \Rightarrow lhs \rightarrow rhs$, il y a une flèche partant de la feuille et pointant vers la notation $\bigwedge_{i=1}^{n} C_i \mapsto rhs$. Chaque nœud interne de l'arbre contient des *variables d'expansion* similaire à une variable de récurrence de la définition 4.2.5. Par exemple, la variable d'expansion du terme $Apply\_network^T(x1, x2)$ est $x1$. Les successeurs directs d'un nœud interne sont le résultat du remplacement des variables d'expansion par des termes de la même sorte appartenant à l'ensemble test construit pour le système de réécriture de la spécification du TCS. Le même raisonnement s'applique tant qu'il n'y a pas de noeuds expandables.

Dans la figure 7.3, on présente un deuxième test de complétude suffisante de $Apply\_network^T$. Cette fois-ci, il n'est plus satisfait : la commande CFU^M est un nouveau constructeur qui sera pris en compte lors de l'extension des variables de sorte Com^M. Lors de la construction de l'arbre de motifs de $Apply\_network^T$, on observe que le cas $Apply\_network^T(CFU^M(n_1,\, n_2),\, [r@t])$ n'est pas défini.

$Apply\_network^T$ doit être étendu de manière conservative, c'est-à-dire que le comportement de TCS vis-à-vis des commandes de la spécification de TCS ne doit pas changer. Ceci devient possible si les commandes CFU^M sont filtrées :

$$\frac{Apply\_network^T(CFU^M(n_1,\, n_2),\, [r@t])}{[r@t]} < App_7^T,\, (True) >$$

On effectue des extensions conservatives de la même manière pour les autres fonctions incomplètes.

A la deuxième étape, on construit les composantes de contrôle de la spécification composée. La fonction de la composante de contrôle, $Apply\_network^M$, est définie sachant que :

$$\begin{cases} idf^T(Apply\_network^M(lc,\, r)) = & Apply\_network^T(lc,\, r) \\ idf^C(Apply\_network^M(lc,\, r)) = & Apply\_network^C(lc,\, r) \end{cases}$$

c'est-à-dire comme l'union des règles de réécriture qui définissent les fonctions $Apply\_network^T$ et $Apply\_network^C$. On répète cette opération aussi pour les fonctions auxiliaires *connect*, *disconnect*, ..., avec l'aide des fonctions d'interfaçage.

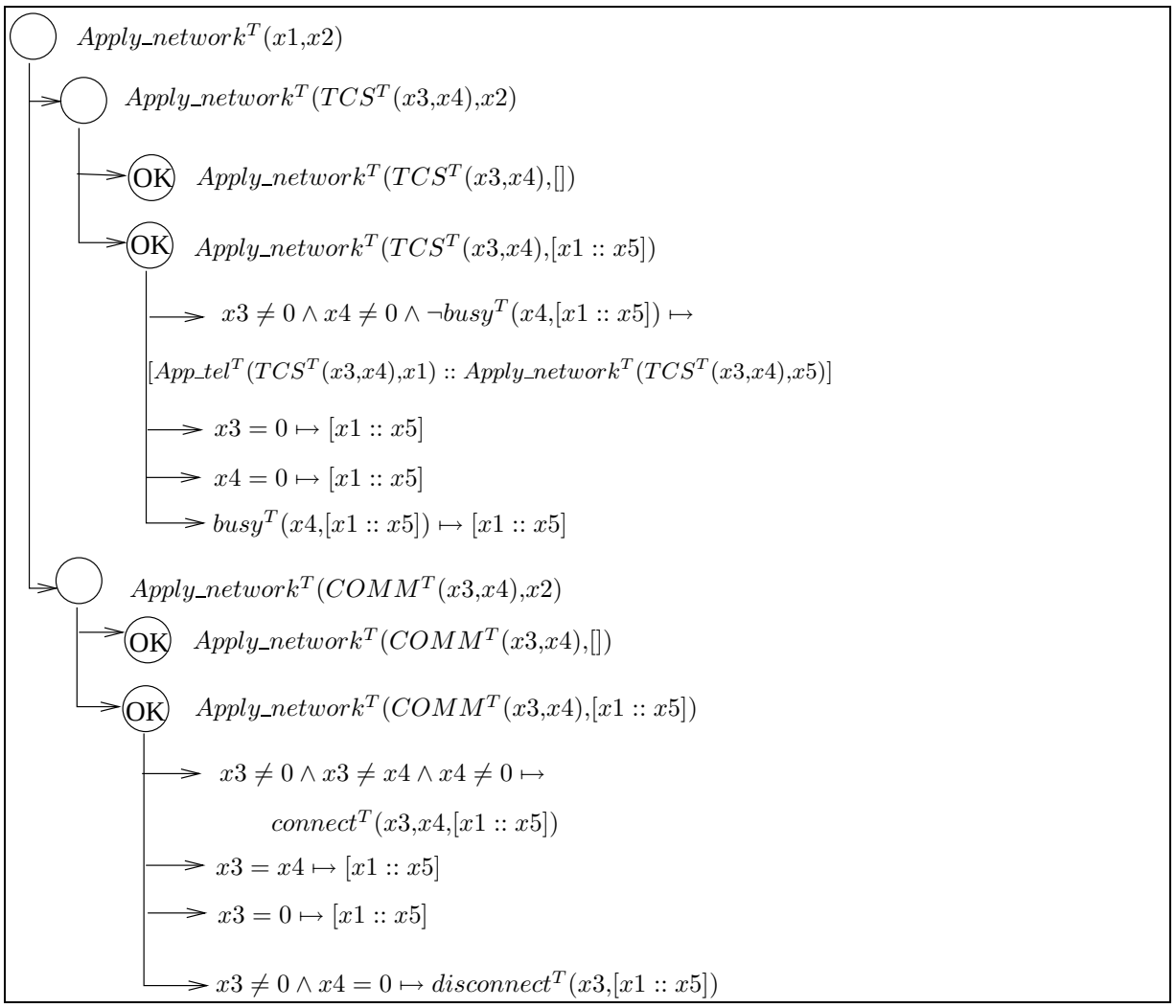

FIG. 7.2 – *$Apply\_network^T$ est complètement définie dans la spécification de TCS*

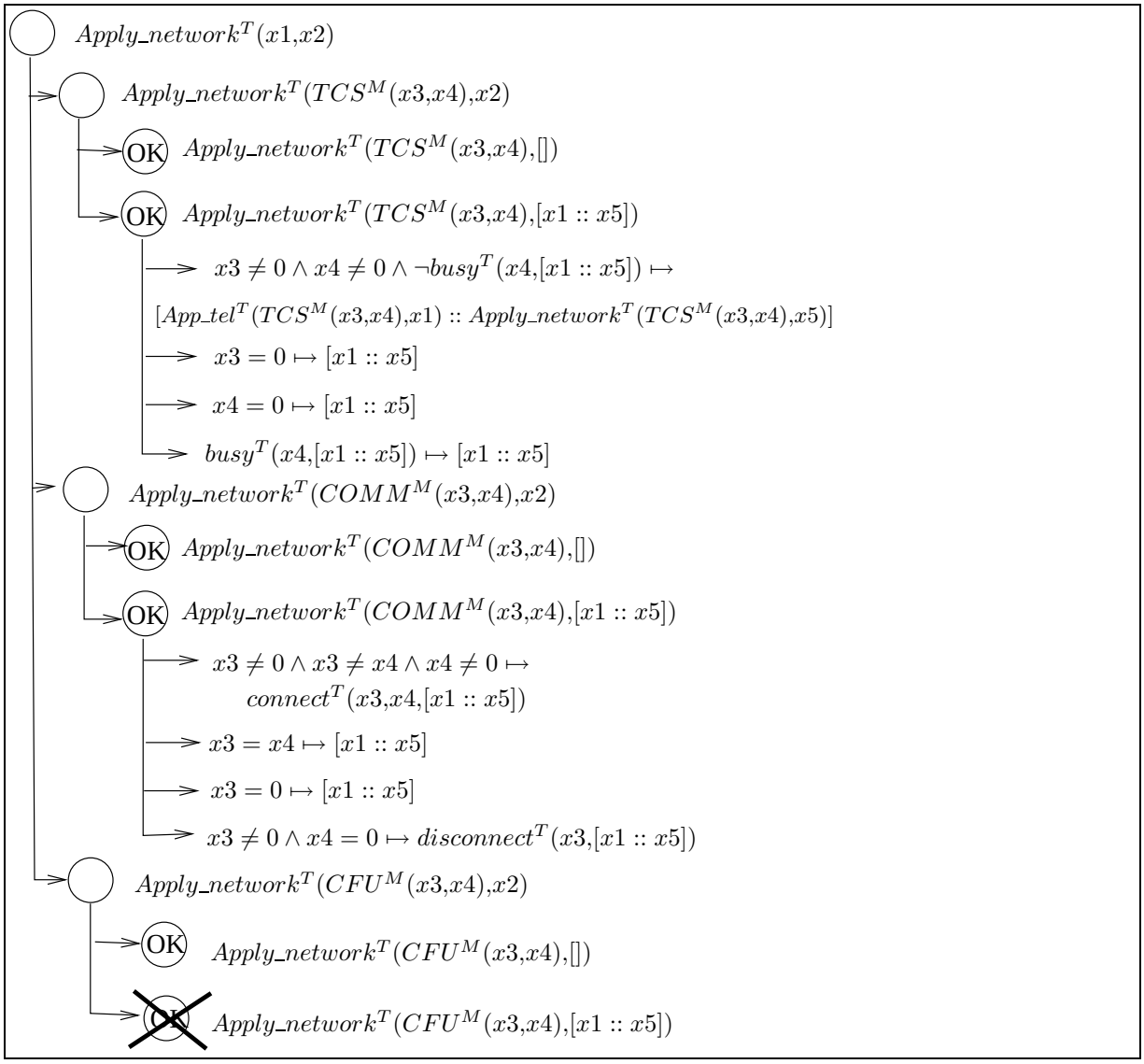

FIG. 7.3 – *$Apply\_network^T$ n'est pas complètement définie dans la spécification composée*

La vérification de la propriété de convergence sur les termes clos

Une fois que le test de la propriété de complétude suffisante du système de réécriture de la spécification composée est effectué, on va vérifier à l'aide du test de convergence sur les termes clos si le système de réécriture dérive des comportements non-déterministes. Ce test échoue s'il existe deux (instances de) règles de réécriture r_1 et r_2 telles que les gardes de transitions de l'automate sont satisfaites mais les transitions sont différentes :

$$\frac{\begin{array}{cc}(r_1) & C_1 \Rightarrow g \rightarrow d_1\\ (r_2) & C_2 \Rightarrow g \rightarrow d_2\end{array}}{Ax \not\models_{ini} C_1 \wedge C_2 \Rightarrow d_1 = d_2}$$

Un cas de non-déterminisme a été trouvé lors de l'exécution de la commande $COMM^M(1, 2)$ sur la configuration de réseau $Init =$

$$\{Tel^M(1,\, 0,\, 0,\, [],\, False^M),\, Tel^M(2,\, 0,\, 3,\, [],\, False^M),\, Tel^M(3,\, 0,\, 0,\, [],\, False^M)\}$$

On obtient deux résultats ; on peut exécuter la règle App_4^T car sa condition d'applicabilité est satisfaite, ce qui mène à la configuration irréductible

$$\{Tel^M(1,\, 2,\, 0,\, [],\, True^M),\, Tel^M(2,\, 1,\, 3,\, [],\, False^M),\, Tel^M(3,\, 0,\, 0,\, [],\, False^M)\}$$

D'autre part, on peut aussi appliquer la règle $Apply_3^C$, car $transit^C(2,\, Init) = 3(\neq 0)$ et $1 \neq 2$. La configuration irréductible est cette fois-ci

$$\{Tel^M(1,\, 3,\, 0,\, [],\, True^M),\, Tel^M(2,\, 0,\, 3,\, [],\, False^M),\, Tel^M(3,\, 1,\, 0,\, [],\, False^M)\}$$

Les deux formes normales sont différentes, donc il y a un non-déterminisme dans le comportement du service composé.

La raison de ce résultat est l'existence de différentes interprétations vis-à-vis des commandes de connexion $COMM^C$ et $COMM^T$. Afin d'éliminer le point de choix, il suffit de décider si $COMM^M$ sera interprété comme $COMM^C$ ou bien comme $COMM^T$. La seconde alternative n'est pas compatible avec la spécification du service CFU, car ceci rend impossible l'opération de renvoi d'appel. Par conséquence, on va interpréter la commande $COMM^M$ comme $COMM^C$.

D'autres points de choix ont été éliminés de la même manière. La plupart d'entre eux sont engendrés par la superposition des règles de réécriture des spécifications de services avec celles qui représentent des extensions conservatives. Dans ces cas, on a accordé la priorité aux règles originaires des spécifications de services.

La vérification des propriétés de conformités

A la dernière étape, on définit les prédicats de la composante logique comme des conjonctions des prédicats constituant les composantes logiques des spécifications de services TCS et CFU. Dans la spécification composée, nous avons décidé d'utiliser un nouveau prédicat, $no\_call\_bl\_himself(Tel^M(n_1,\, n_2,\, n_3,\, ln,\, f)) : N^M \rightarrow Bool^M$, équivalent à la conjonction des (extensions conservatives des) prédicats $no\_selfcall$ et $no\_call\_black\_list$. Le prédicat est faux si soit n_1 est appelé par lui-même soit l'utilisateur n_2 est dans la liste noire ln lorsque f est $False^M$, c'est-à-dire qu'il y a une connexion entre n_1 et n_2 malgré le fait que n_2 est dans la liste noire de n_1 :

$$\frac{no\_call\_bl\_himself(Tel^M(n_1,\, n_2,\, n_3,\, ln,\, f))}{True^M} < no\_call\_bl\_h_1,\, e_1 > ,$$

où la condition e_1 est $(\neg Member^M(n_2,\, ln) \wedge n_1 \neq n_2)$.

$$\frac{no\_call\_bl\_himself(Tel^M(n_1,\, n_2,\, n_3,\, ln,\, f))}{f} < no\_call\_bl\_h_2,\, e_2 >$$

La condition e_2 est $(Member^M(n_2,\, ln) \wedge n_1 \neq n_2)$.

$$\frac{no\_call\_bl\_himself(Tel^M(n_1,\, n_2,\, n_3,\, ln,\, f))}{False^M} < no\_call\_bl\_h_3,\, (n_1 = n_2) >$$

Le prédicat $Member^M$ est similaire au $Member^T$.

La conjecture qui décrit le test de conformité, selon la définition 7.2.2, est l'invariant :

$$\boxed{\forall \overline{cl}\ \forall Init,\, Accepte^C(\overline{cl},\, Init)\ \wedge\ Accepte^T(\overline{cl},\, Init) \Rightarrow Accepte^M(\overline{cl},\, Init),}$$

où $\overline{cl}$ représente la liste de commande de $\mathbb{P}(Com^M)$ et $Init = [Tel_1, \ldots, Tel_k]$ décrit la configuration de l'état initial du réseau téléphonique. Notons que $\overline{cl}$ est une liste non-bornée†.

Les prédicats d'admissibilité $Accepte^T$, $Accepte^C$ et $Accepte^M$, tous de profil $\mathbb{P}(Com^M) \times \mathbb{P}(N^M) \rightarrow \mathbb{P}(N^M)$, vérifient si la configuration $Init$ et celle obtenue après l'exécution des commandes de $\overline{c}$ sur $Init$ sont admissibles (voir le prédicat $Accepte$ de la définition 7.2.2). Soit $\overline{c} = [c_1, c_2, \ldots, c_n]$ une liste de commandes et $Map(g, [x_1, \ldots, x_n], I) = g(x_n, g(\ldots(x_2, g(x_1, I))\ldots))$ l'application successive des commandes de la liste de commandes $[x_1, \ldots, x_n]$ sur un état initial du réseau, en utilisant la fonction g d'une composante de contrôle arbitraire. Les prédicats d'admissibilité sont définis comme suit :

- Soit $[Tel'_1, \ldots, Tel'_k] = Map(Apply\_network^T, \overline{c}, Init)$.
  Le prédicat $Accepte^T(\overline{c}, Init)$ est vrai si
  $$\forall i \in [1..k], (no\_call\_black\_list(Tel'_i) \wedge no\_call\_black\_list(Tel_i)).$$

- Soit $[Tel'_1, \ldots, Tel'_k] = Map(Apply\_network^C, \overline{c}, Init)$.
  $Accepte^C(\overline{c}, Init)$ est vrai si
  $$\forall i \in [1..k], (no\_selfcall(Tel'_i) \wedge no\_selfcall(Tel_i)).$$

- Soit $[Tel'_1, \ldots, Tel'_k] = Map(Apply\_network^M, \overline{c}, Init)$.
  Le prédicat $Accepte^M(\overline{c}, Init)$ est vrai si
  $$\forall i \in [1..k], (no\_call\_bl\_himself(Tel'_i) \wedge no\_call\_bl\_himself(Tel_i)).$$

La variable $Init$ est

$$[Tel^M(1,\, 0,\, [],\, 0,\, False^M),\, Tel^M(2,\, 0,\, [],\, 0,\, False^M)]$$

et correspond à la situation où les utilisateurs ne sont pas connectés et aucun service n'est activé.

Nous avons ainsi trouvé que la spécification composée est conforme aux spécifications des services CFU et TCS pour des configurations de réseaux constituées par deux utilisateurs. *Spike* prouve la conjecture de conformité en appliquant GENERATE dans une première étape : la variable de récurrence $\overline{cl}$ est remplacée par chaque élément de l'ensemble test

$$\{[TCS^M(0,\, 0)@\overline{cl'}], \ldots, [TCS^M(2,\, 2)@\overline{cl'}], [CFU^M(0,\, 0)@\overline{cl'}], \ldots,$$
$$[CFU^M(2,\, 2)@\overline{cl'}], [COMM^M(0,\, 0)@\overline{cl'}], \ldots, [COMM^M(2,\, 2)@\overline{cl'}]\}$$

† ce qui rend difficile ce test avec des méthodes basées sur le model-checking

Puis, les conjectures ainsi obtenues sont simplifiées avec des règles de réécriture et la conjecture est mise dans l'ensemble de prémisses.

Certaines de ces nouvelles conjectures sont éliminées en appliquant les hypothèses de récurrence lorsque la liste de commande est filtrée. Par exemple, la conjecture correspondant à l'instanciation de $\overline{cl}$ avec $[CFU^M(0,\,0)@\overline{cl}]$ a été simplifiée et réduite à la conjecture de conformité (modulo renommage de variables). La commande $CFU^M(0,\,0)$ a été filtrée par les règles App_7^T, App_3^C et la règle correspondante à App_3^C de $Apply\_network^M$. Par conséquent, l'application de cette commande ne modifie pas la configuration initiale du réseau. Les autres listes de commandes, obtenues après l'application en plusieurs reprises de GENERATE, ont été filtrées par les prédicats de la composante logique de CFU et TCS. Finalement, on est arrivé à un ensemble vide de conjectures et la dérivation se termine avec succès.

On considère cette preuve assez difficile à cause du nombre relativement élevé de conjectures produites pendant les étapes de GENERATE.

Dans le cas où les configurations du réseau contiennent trois utilisateurs, on trouve une interaction due à la conformité si on suit le scénario suivant :

$$[TCS^M(1,\,3),\,CFU^M(2,\,3),\,COMM^M(1,\,2)]$$

instancie la liste de commandes $\overline{cl}$, et

$$[Tel^M(1,\,0,\,[],\,0,\,False^M),\,Tel^M(2,\,0,\,[],\,0,\,False^M),\,Tel^M(3,\,0,\,[],\,0,\,False^M)]$$

instancie $Init$.

Ce scénario correspond en fait à l'interaction décrite dans l'exemple 7.1.1 ; $\overline{cl}$ satisfait les propriétés des spécifications de TCS et CFU, mais non celles de la spécification composée. *Spike* a rejeté la conjecture de conformité en travaillant en mode réfutationnel. La stratégie de preuve employée choisit en priorité la normalisation des conjectures closes car elles représentent des contre-exemples potentiels.

7.7 Conclusions

Dans ce chapitre, nous avons montré le potentiel offert par les outils de démonstration automatique pour i) la conception des spécifications de services téléphoniques, ii) la détection des interactions engendrées au moment de l'inter-fonctionnement de plusieurs services sur un même réseau, et iii) la résolution des interactions non-souhaitables. Nous avons proposé une méthodologie qui permet la détection et la résolution *off-line* des interactions observables par l'utilisateur avec des techniques basées sur la réécriture conditionnelle et la récurrence implicite. L'étude de cas choisie est l'analyse de l'inter-fonctionnement des deux services, TCS et CFU, en utilisant les facilités offertes par le prouveur SPIKE.

Le point de départ est représenté par les spécifications informelles des deux services. Puis, le comportement et les propriétés d'un service sont modélisés formellement dans une même spécification conditionnelle qui indique la manière dont une certaine configuration du réseau, modélisé globalement, se modifie après l'application des commandes spécifiques au service. La détection et la résolution des interactions de services sont effectuées pendant le processus de spécification.

Après la preuve de complétude suffisante et de convergence sur les termes clos des services individuels, nous avons proposé les sortes et les fonctions lors de la conception de la spécification

composée à l'aide des composantes d'interfaçage de chaque spécification de service individuellement. Dans la spécification composée, les spécifications des services ont perdu leur propriété de complétude suffisante par rapport aux nouveaux constructeurs et, par conséquent, elles ont du être étendues de manière conservative. Les interactions dues au non-déterminisme ont été détectées automatiquement. Leur résolution a été faite par l'introduction de précédences dans les points de choix. Enfin, après ces modifications, la spécification composée obtenue a été démontré suffisamment complète et convergente sur les termes clos.

Pour détecter les interactions de conformité, nous avons vérifié la conjecture de conformité sur des configurations de réseau à deux et trois utilisateurs. Dans le premier cas, on a utilisé l'interface parallèle de SPIKE, SPIKEPAR, décrite dans la section 5.3. L'outil a travaillé dans le mode positif et a appliqué la récurrence pour prouver la validité de la conjecture. Dans le deuxième cas, il a détecté une interaction en simulant le model-checking.

Nous avons appliqué avec succès notre méthodologie pour analyser l'interopérabilité des services de numérotation abrégée et filtrage des appels au départ. Ainsi, nous avons détecté l'interaction due au non-déterminisme, présentée dans l'exemple 7.1.2.

Pendant ces études de cas, nous avons utilisé

- une procédure de test de la **propriété de complétude suffisante du système de réécriture associé à la spécification conditionnelle** Le concepteur de la spécification est informé si ses fonctions définies sont complètes. Dans le cas contraire, SPIKE propose des suggestions pour compléter les cas manquants si les fonctions définies sont à conditions booléennes ;
- une procédure de test de la **propriété de convergence sur les termes clos du système de réécriture** Ce test permet de détecter des interactions due au non-déterminisme dans le comportement de la spécification composée. La procédure de convergence sur les termes clos qu'on utilise permet aussi de détecter des cas non-triviaux, en particulier lorsque ceux-ci correspondent à des situations où les conditions des règles conditionnelles s'interposent ;
- les **techniques de preuves d'invariants** de SPIKE afin de tester la conformité de la spécification composée ;
- SPIKE pour **simuler des comportements de services sur des configurations particulières** Etant données une configuration du réseau et une liste finie de commandes closes, on peut « visualiser » le comportement du service en analysant la trace de la preuve pendant les étapes de simplification. Ceci faciliterait le travail du concepteur de la spécification composée.

8

Preuve de la conformité du protocole *ABR*

Sommaire

8.1 Introduction

ABR (*A*vailable *B*it *R*ate) est un protocole utilisé pour régler les débits optimaux de transfert d'information dans les réseaux de haut débit, comme les réseaux *ATM* (*A*synchronous *T*ransfer *M*ode). La raison pour laquelle il est bien adapté au transport de données est sa garantie d'accorder un débit minimal aux utilisateurs, appelés aussi *sources ABR*†. Il peut augmenter de manière dynamique selon les ressources disponibles dans le réseau. Afin d'acquérir une grande flexibilité, ABR emploie des mécanismes laborieux de gestion du trafic [Jain, 1996]. L'opérateur doit maîtriser le débit courant en temps-réel pour éviter la congestion du réseau et pour assurer qu'elle est consistante avec le débit minimal admis, ce qu'on va appeler la *vérification (ou le contrôle) de la conformité*. Plusieurs algorithmes mettant en œuvre le contrôle de la conformité ont été proposés et débattus dans des comités de standardisation.

Parmi ceux-ci, on distingue l'algorithme Acr, défini par le forum ATM, qui est considéré comme une référence parce qu'il calcule le plus grand débit par rapport aux autres. Acr utilise une liste de cellules où les débits permis à certains instants sont enregistrés. Un autre algorithme,

† Ce débit minimal est explicitement spécifié dans le contrat entre l'opérateur qui gère le réseau et l'utilisateur.

nommé B' et inventé par C. Rabadan de France-Télécom [Rabadan, 1997], a été conçu initialement pour calculer une approximation du contrôle optimal pour des listes à deux cellules. Il s'avère plus efficace car le débit à contrôler à un moment donné peut être obtenu de manière instantanée. Cet algorithme a été généralisé dans [Rabadan et Klay, 1997] pour des listes de cellules de longueurs arbitraires.

Il est essentiel pour un opérateur de justifier que le contrôle de la conformité qu'il offre aux consommateurs n'affecte pas la qualité du service (QoS) fournie par les réseaux ATM. Une tâche aussi complexe a besoin de validation formelle, ce qui demande des arguments mathématiques et de techniques de raisonnement évoluées comme l'analyse par cas ou la récurrence. C'est pour cette raison que certains opérateurs utilisent des outils de vérification automatisés (comme des assistants de preuves ou des model-checkers) pendant le traitement des obligations de preuves.

Dans ce chapitre, on va dériver une preuve automatisée qui montre que la généralisation de l'algorithme B', notée Acr1, est équivalente à l'algorithme de référence Acr, c'est-à-dire que chaque étape de la preuve d'équivalence a été prouvée formellement. Le démonstrateur de théorèmes utilisé est PVS [Owre *et al.*, 1992]. Même si PVS est un prouveur interactif qui travaille sous le contrôle direct de l'utilisateur, il est capable de faire de grandes étapes de déduction de manière autonome par l'appel aux procédures de décision pour l'arithmétique, à la réécriture et à la récurrence.

La structure du chapitre est la suivante. D'abord, on décrit dans la section 8.3 le principe du protocole ABR et du contrôle de la conformité. Puis, on présente respectivement dans les sections 8.4 et 8.5 les algorithmes Acr et Acr1 pour le calcul du débit admis à tout instant. La suite du chapitre est dédiée à la preuve d'équivalence entre ces deux algorithmes. On introduit d'abord les propriétés clé dans la section 8.6.1, puis on fait un survol de la preuve dans la section 8.6.2. Comme la preuve avec PVS est trop complexe pour être présentée ici `in extenso`, on va ébaucher son squelette en expliquant ses lemmes principaux. Ensuite, on commente la preuve automatisée dans la section 8.6.3. Enfin, on montre que les techniques de preuve par ensembles couvrants contextuels de SPIKE permettent l'automatisation de la preuve formelle. Pour cela, la spécification PVS a été traduite dans une spécification conditionnelle acceptée par SPIKE. Nous montrons que SPIKE, intégrant une procédure de décision pour l'arithmétique linéaire comme décrit dans le chapitre 6, prouve complètement automatiquement plus de la moitié des lemmes donnés par l'utilisateur. Ainsi, son interaction avec le prouveur est réduite considérablement.

8.2 Etat de l'art

Quelques algorithmes ABR ont été formellement certifiés antérieurement. Pourtant, tous ces algorithmes sont des approximations d'Acr, car ils opèrent sur des listes de cellules de taille finie, comme l'algorithme B'. Celui-ci a été montré correct récemment dans [Monin et Klay, 1999]. La preuve est basée sur le calcul des préconditions les plus faibles (weakest preconditions) [Dijkstra, 1976], en construisant des invariants inductifs, et a été complètement formalisée avec l'assistant de preuves COQ [Barras *et al.*, 1997]. Selon [Monin et Klay, 1999], la preuve de correction de B' « a été un argument clé dans le processus de standardisation d'ABR ». Certaines techniques de preuve on été expérimentées dans le cadre du projet FORMA (`http://www-verimag.imag.fr`) pour traiter la validation des protocoles ABR. Les approches basées sur le model-checking ont été pénalisées par les paramètres numériques de l'algorithme. L. Fribourg a obtenu aussi de bons

résultats avec les automates temporels étendus [Fribourg, 1998]. Une preuve de la conformité du protocole B' par des méthodes basées sur les automates temporisés paramettrés de Hytech [Henzinger *et al.*, 1997] a été présentée dans [Bérard et Fribourg., 1999]. L'annexe B contient la spécification PVS et SPIKE des deux algorithmes et les conjectures à prouver.

8.3 Le protocole ABR

La technologie ATM permet aux réseaux de transmettre, sur le même support média et simultanément, des applications diverses qui nécessitent différents débits de flots de données ou des QoS. ATM est une technologie orientée à la connexion car les utilisateurs doivent déclarer les spécifications du service et les paramètres de trafic à tous les commutateurs intermédiaires au moment de l'initialisation des connexions. L'utilisateur a aussi le droit de modifier ces paramètres à la demande. Afin de garantir la QoS, un contrat visant le type du trafic utilisé est négocié lorsque la connexion s'établit. La gestion du trafic doit assurer que les utilisateurs ont la QoS souhaitée même dans les conditions où le débit varie constamment. Pour conclure, la gestion du trafic doit assurer que tous les contrats sont accomplis.

Afin de résoudre le problème critique de la maîtrise de la congestion du réseau, les débits effectifs de données émises par les applications des utilisateurs sont dirigés par un algorithme de conformité appelé $GCRA$ (*G*eneric *C*ell *R*ate *A*lgorithm). Plusieurs modes de trafic ont été conçus ; les protocoles à débit de bit constant CBR (*C*onstant *B*it *R*ate) et à débit de bit variable VBR (*V*ariable *B*it *R*ate) ont été ciblés au trafic du son ou de la vidéo. La classe de services ABR est spécialement adaptée au trafic standard de données, où les contraintes temporelles ne sont pas fortes. Des applications typiques utilisant ABR sont le courrier électronique, le portail WWW et le transfert de fichiers.

Le principe d'ABR est de partager la bande de fréquence disponible de manière équitable entre les sources actives, appelées *terminaux ABR*, tel que le réseau offre à l'utilisateur le meilleur débit compatible avec le trafic courant. Le débit admissible ACR (*A*llowed *C*ell *R*ate) est calculé à partir de l'information sur la charge du réseau et peut varier avec le temps pour une même connexion. Le réseau informe périodiquement l'utilisateur de son nouveau débit qu'il peut utiliser par le biais de cellules de données particulières, appelées *cellules pour la gestion des ressources RM* (*R*esource *M*anagement). Par conséquent, le débit à utiliser par la source est dynamiquement ajusté dans une boucle fermée (à voir la figure 8.1). Comme le débit alloué varie pendant une connexion utilisant le mode ABR, la vérification de la conformité est effectuée par un GCRA dynamique (DGCRA).

Différents algorithmes de conformité ABR ont été proposés dans des comités de standardisation. L'algorithme Acr [Berger *et al.*, 1995] peut être considéré comme une référence pour définir le contrôle du flot de données de l'utilisateur. Chaque cellule de données envoyée par le terminal ABR arrive à une interface de contrôle et Acr calcule le débit maximal admissible de cette cellule. Le coût de calcul induit par Acr a été jugé trop élevé et, par conséquent, plusieurs propositions d'améliorations ont été envisagées.

Par exemple, on a exploité le fait que le changement de débit est seulement déterminé par des cellules RM qui se dirigent de l'interface vers le terminal ABR, au moment de leur départ. Ces cellules RM, de type *backward*†, sont moins fréquentes que les cellules de données. Par conséquent, il est plus efficace de programmer le débit à l'avance, au moment où on reçoit les

† par opposition aux cellules *forward*, qui se dirigent au sens inverse, de l'interface vers le réseau

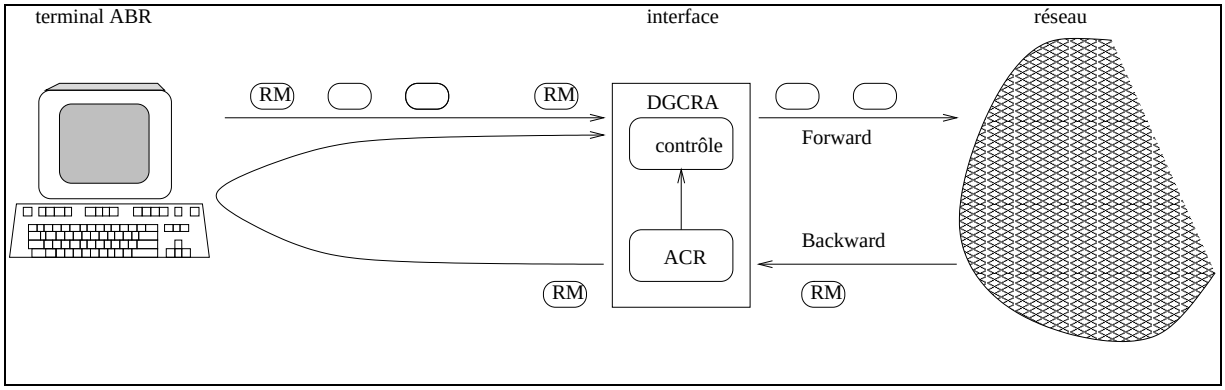

FIG. 8.1 – *La boucle de contrôle des débits entre le réseau et un terminal ABR*

cellules RM de type backward à l'interface. Ceci est la raison de la création de l'algorithme incrémental Acr1 qui gère une liste de débits programmés dans le futur. Cette liste est ajoutée à chaque moment où une nouvelle cellule de type backward arrive à l'interface. Le coût du calcul est aussi moins élevé par rapport à Acr, puisque Acr doit calculer le maximum d'une liste de débits, comme on va montrer dans la section 8.4.

8.4 L'algorithme Acr

Dans cette section, on va détailler l'algorithme Acr. Il a été présenté pour la première fois au Forum ATM par Berger, Bonomi et Fendick [Berger *et al.*, 1995] et est considéré comme une référence pour les autres algorithmes.

Le principe de l'algorithme Acr exécuté par l'interface est de gérer une liste de cellules RM de type backward reçues du réseau (à voir la figure 8.1) afin de déterminer le débit à utiliser par la source à tout instant. A chaque cellule RM, on associe un couple (t,er), où t est l'instant où la cellule RM de type backward quitte l'interface vers le terminal ABR et er est le nouveau débit à imposer au terminal ABR par le réseau. Dans la suite, on va identifier ces cellules par leurs couples associés. Par convention, on appelle *instant* (resp. *débit*) la première (resp. deuxième) composante d'un couple. Le dispositif de contrôle des débits, situé aussi dans l'interface, va recevoir la nouvelle valeur du débit après le terminal ABR. Ceci est dû non seulement aux délais de transport de l'information dans le réseau mais aussi à l'inertie avec laquelle la source ajuste son application au nouveau débit. Par conséquent, le dispositif de contrôle doit vérifier une valeur à un instant t qui a été reçue à l'instant $t - \tau$, où τ représente le délai de propagation égal à la durée nécessaire à une cellule RM pour aller de l'interface au terminal ABR et retourner à l'interface. Cette valeur peut varier dans le temps selon la charge du réseau. Afin de prendre en compte ces variations, l'organisation ITU-T a proposé que le débit à vérifier au moment t soit calculé comme le maximum de débits des cellules reçues dans une fenêtre temporelle délimitée par $t-\tau_2$ et $t-\tau_3$ et le débit de la cellule reçue juste avant ou au moment $t-\tau_2$. Les *paramètres de la fenêtre* τ_2 et τ_3 satisfont la condition $\tau_2 > \tau_3$. Ils se négocient au moment de la création du contrat entre l'utilisateur et l'opérateur du réseau.

Plus formellement, soit $l = [(t_1,\ er_1),\ (t_2,\ er_2), \ldots,\ (t_n,\ er_n)]$ une liste de cellules RM. La *première cellule* de l est $(t_1,\ er_1)$. La liste l est *temporellement décroissante* si $i < j \Rightarrow t_i \geq t_j$, pour tout $i \in [1..n-1]$ et $j \in [2..n]$. Afin de manipuler les cas limite dans les définitions qui

vont suivre, on va définir $t_0 = +\infty$. On dénote par $\cdot$ (resp. @) l'opérateur *cons* (resp. *append*) sur les listes.

Le débit à vérifier au moment t par rapport à une liste temporellement décroissante $l = [(t_1, er_1), (t_2, er_2), \ldots, (t_n, er_n)]$ de cellules RM de type backward quittant l'interface est :

$$Acr(l, t) = \begin{cases} MaxEr(Wind(l, t)) & \text{si } Wind(l, t) \neq \emptyset, \\ 0 & \text{sinon} \end{cases}$$

où

$$\begin{array}{lcl} Wind(l, t) & = & \{(t_i, er_i) \in l |\ (t - \tau_2 < t_i \leq t - \tau_3) \text{ ou } (t_i \leq t - \tau_2 < t_{i-1})\} \\ MaxEr(s) & = & max\{er | (t, er) \in s\} \end{array}$$

Par exemple, avec cette politique de contrôle du débit, l'utilisateur peut bénéficier à l'instant $t+\tau_3$ d'une croissance du débit annoncée par une cellule RM de type backward arrivée à l'instant t à l'interface, c'est-à-dire le plus tôt possible. D'autre part, une décroissance du débit, ne va être prise en compte qu'à l'instant $t+\tau_2$, c'est-à-dire le plus tard possible. Par conséquent, cette politique est en faveur de l'utilisateur. Notons que pour une liste fixée l, $Acr(l, t)$ est décroissant après $t+\tau_3$ car la fenêtre est (non strictement) décroissante par rapport à la relation d'inclusion.

8.5 L'algorithme incrémental Acr1

Dans cette section, on va introduire l'algorithme incrémental idéal Acr1 pour la vérification de la conformité. Contrairement à Acr, l'algorithme Acr1 calcule une liste de débits à vérifier dans le futur. Par conséquent, il gère une liste $Prog(l)$ de cellules (l est comme avant) (t_j, er_j) contenant le débit futur er_j à vérifier, ainsi que l'instant t_j quand il sera pris en compte. Pareil comme l, $Prog(l)$ sera une liste temporellement décroissante. Cette liste est mise à jour lors de la réception d'une cellule RM de type backward. Un gain important par rapport à Acr est dû au fait que les cellules RM sont moins fréquentes que les cellules de données. Le rapport recommandé par le Forum ATM† entre les cellules RM et celles de données est 1/32. Supposons que la liste $Prog(l)$ a été déjà construite. Le débit à vérifier à l'instant t est :

$$Acr1(l, t) = Prog(l)_t$$

où p_t est une fonction qui calcule par extrapolation le débit à l'instant t d'une liste de débits programmés p. Cette fonction extrait simplement de p le débit programmé à l'instant t ou immédiatement avant ce dernier. Ce débit sera celui à vérifier à l'instant t. Formellement,

$$p_t \quad = \quad \begin{cases} er_i & \text{s'il existe une cellule } (t_i, er_i) \in p \text{ telle que } t \geq t_i \text{ et} \\ & \quad \text{il n'y a pas de cellule } (t_j, er_j) \in p \text{ avec } t \geq t_j,\ i > j \text{ et } i, j \in [1..n] \\ 0 & \text{sinon} \end{cases}$$

On va décrire maintenant la manière dont la liste $Prog(l)$ est mise à jour. Une liste l^1 est un *préfixe* d'une liste l s'il existe une liste l^2 telle que $l = l^1@l^2$. Etant donnés une liste l^1 et un instant t, on va dénoter par $l^1_{>_\tau t}$ le préfixe maximal de l^1 contenant des cellules avec des instances supérieures à t. De manière similaire, on définit $l^1_{\leq_\rho er}$ comme le préfixe maximal de l^1 contenant des cellules avec des débits inférieurs ou égaux à er. Ceci aboutit à la définition récursive suivante de $Prog(l)$:

† ATM Forum Traffic Management Specification Version 4.0.
ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.ps

$$Prog((t,\, er)\cdot l) = \begin{cases} \textbf{si} & er \geq Prog(l)_{t+\tau_3} \textbf{ alors } (t+\tau_3,\, er)\cdot l^1 \\ & \text{où } Prog(l) = Prog(l)_{>_\tau t+\tau_3}@l^1. \\ \textbf{sinon} & \textbf{si } Prog(l)_{\leq_\rho er} \text{ est vide } \textbf{alors } (t+\tau_2,\, er)\cdot Prog(l) \\ & \textbf{sinon } (t',\, er)\cdot l^2 \\ & \text{où } Prog(l) = Prog(l)_{\leq_\rho er}@l^2 \\ & \text{ et } Prog(l)_{\leq_\rho er} = L@[(t',\, er')] \end{cases}$$

$Prog(l)$ est, par définition, une liste vide si l l'est aussi.

8.6 Vérification de la conformité de l'algorithme Acr1

Dans la suite, on va montrer que l'algorithme Acr1 calcule les mêmes débits que l'algorithme référence Acr, c'est-à-dire

$$\forall t\ \forall l \quad Acr1(l,\, t) = Acr(l,\, t)$$

8.6.1 Deux propriétés clé

Il y a deux propriétés qui ont été abondamment utilisées pendant la preuve. La première, `time_dec`, dit que $Prog(l)$ est triée par ordre décroissant sur la composante temporelle des cellules. La deuxième, `rate_inc`, spécifie qu'un certain préfixe de $Prog(l)$ est trié par ordre croissant sur la deuxième composante des cellules. Les deux supposent que l est temporellement décroissante. On va dénoter par $Timel(l)$ l'instant de la première cellule (0 si l est vide).

Propriété 8.6.1 (time_dec) *Etant donnée une liste l temporellement décroissante, la liste $Prog(l)$ l'est aussi.*

Preuve Par récurrence sur l. Lorsque l est vide, $Prog(l)$ l'est aussi, donc aussi temporellement décroissante. On suppose par l'hypothèse de récurrence que pour toute liste temporellement décroissante l_1 de longueur inférieure ou égale à la longueur de l, $Prog(l_1)$ est temporellement décroissante. On va prouver que $Prog((t,\, er)\cdot l)$ est temporellement décroissante lorsque $t \geq Timel(l)$. On va faire une analyse par cas guidée par la définition de $Prog$ de la section 8.5. Dans cette analyse on va remarquer que toute sous-liste d'une liste temporellement décroissante est aussi temporellement décroissante.

1. on suppose que $er \geq Prog(l)_{t+\tau_3}$. Soit l^1 satisfaisant $Prog(l) = Prog(l)_{>_\tau t+\tau_3}@l^1$. Alors, $Prog((t,\, er)\cdot l)$ est égale à $(t+\tau_3,\, er)\cdot l^1$ qui est temporellement décroissante car $t+\tau_3 \geq Timel(l^1)$ et l^1 le sont aussi.
2. sinon, soit $Prog(l) = Prog(l)_{\leq_\rho er}@l^2$. Si $Prog(l)_{\leq_\rho er}$ est vide, alors la liste $(t+\tau_2,\, er)\cdot Prog(l)$ est temporellement décroissante car $t+\tau_2 \geq Timel(Prog(l))$. Ceci est obtenu par (i) la propriété de liste temporelle décroissante de $(t,\, er)\cdot l$, et (ii) le fait que pour toute liste non-vide temporellement décroissante $L = (\overline{t},\, \overline{e})\cdot L^1$ on a $\overline{t}+\tau_2 \geq Timel(Prog(L))$ †.
Si $Prog(l)_{\leq_\rho er}$ est une liste non-vide de la forme $L^2@[(t',\, er')]$ alors $(t',\, er)\cdot l^2$ est une liste temporellement décroissante comme on peut déduire à partir de la propriété de liste temporellement décroissante de $Prog(l)$.

Fin de preuve

† Ceci peut être prouvé par une simple récurrence sur L.

On dénote par $l_{\frown t}$ le préfixe maximal l^1 de l tel que l'instant de toute cellule de l^1, éventuellement à l'exception de la dernière, est supérieure à t. Pour cela, on observe que pour toute liste non-vide temporellement décroissante l, le préfixe $l_{\frown t}$ et l ont la même première cellule. On dit qu'une liste de cellules $l = [(t_1,\ er_1),\ (t_2,er_2), \ldots,\ (t_n,\ er_n)]$ est *strictement croissante par le débit* si les débits de ses cellules sont strictement croissants : $i < j \Rightarrow er_i < er_j$, pour tout $i,\ j \in [1..n]$. On peut observer que, étant données une liste strictement croissante par le débit l et deux instances $t > t'$, on a $l_t \leq l_{t'}$.

Propriété 8.6.2 (rate_inc) *Etant donnée une liste de cellules l, le préfixe $Prog(l)_{\frown Timel(l)+\tau_3}$ de $Prog(l)$ est strictement croissant par le débit.*

Preuve Par récurrence sur l. Si l est vide alors $Prog(l)$ est vide, donc strictement croissante par le débit. Sinon, par hypothèse de récurrence, on suppose que $Prog(l)_{\frown Timel(l)+\tau_3}$ est strictement croissante par le débit. Si $t \geq Timel(l)$ on va prouver que $Prog((t,\ er) \cdot l)_{\frown t+\tau_3}$ est aussi strictement croissante par le débit, en effectuant une analyse par cas selon la définition de $Prog$.

1. si $er \geq Prog(l)_{t+\tau_3}$ alors $Prog((t,\ er) \cdot l) = (t + \tau_3,\ er) \cdot l^1$, où l^1 est telle que $Prog(l) = Prog(l)_{>_\tau t+\tau_3} @ l^1$. Il résulte que $Prog((t,\ er) \cdot l)_{\frown t+\tau_3}$ est égale à $[(t+\tau_3,\ er)]$ et elle est strictement croissante par le débit.

2. sinon, soit l^2 telle que $Prog(l) = Prog(l)_{\leq_\rho er} @ l^2$:

2.1. si $Prog(l)_{\leq_\rho er}$ est vide, alors $Prog((t,\ er) \cdot l) = (t + \tau_2,\ er) \cdot Prog(l)$.
On a $Prog((t,\ er) \cdot l)_{\frown t+\tau_3} = (t + \tau_2,\ er) \cdot Prog(l)_{\frown t+\tau_3}$. De plus, $Prog(l)_{\frown t+\tau_3}$ est un préfixe de $Prog(l)_{\frown Timel(l)+\tau_3}$ car $t \geq Timel(l)$. Par conséquent, $(t + \tau_2,\ er) \cdot Prog(l)_{\frown t+\tau_3}$ est strictement croissante par le débit.

2.2. sinon $Prog(l)_{\leq_\rho er}$ est de la forme $L@[(t',\ er')]$ et $Prog((t,\ er) \cdot l) = (t',\ er) \cdot l^2$. Si l^2 est vide, $Prog((t,\ er) \cdot l)_{\frown t+\tau_3}$ est formée par la cellule unique $(t',\ er')$, qui est clairement strictement croissante par le débit. Comme l^2 est une sous-liste de $Prog(l)$, la liste $l^2_{\frown t+\tau_3}$ est une sous-liste de $Prog(l)_{\frown t+\tau_3}$. Par conséquent, elle est strictement croissante par le débit. De plus, er est plus petit que le débit de la première cellule de l^2, et donc aussi de celle de $l^2_{\frown t+\tau_3}$, lorsque l^2 n'est pas vide. On conclut que $Prog((t,\ er) \cdot l)_{\frown t+\tau_3} = (t',\ er) \cdot l^2_{\frown t+\tau_3}$ est strictement croissante par le débit.

Fin de preuve

8.6.2 Squelette de la preuve

Dans la suite, on va utiliser les notations suivantes. Supposons que x est une cellule, $Time(x)$ son instant, $Er(x)$ son débit et τ_2, τ_3 les deux paramètres caractérisant la fenêtre temporelle qu'on a mentionné dans la section 8.4. Notons que les approches par model-checking avec MEC [Arnold, 1990] et UPPAAL [Bengtsson *et al.*, 1996] ont du attribuer des valeurs à ces paramètres pour qu'elles soient applicables. Dans notre cas, on suppose seulement la contrainte $\tau_2 > \tau_3$ sans aucune mention supplémentaire. On introduit $T_2(x)$ pour dénoter $Time(x) + \tau_2$ et $T_3(x)$ pour dénoter $Time(x) + \tau_3$. Le prédicat $\mathcal{S}^{\geq_\tau}(l)$ (resp. $\mathcal{S}^{<_\rho}(l)$) est vrai si l est temporellement décroissante (resp. strictement croissante par le débit). Comme toutes les variables sont quantifiées universellement, on va omettre de mentionner les quantificateurs des préfixes de formules.

La preuve de la conjecture $Acr1(l,\ t) = Acr(l,\ t)$ est immédiate si on arrive à prouver le *lemme principal* $P(l)$, où :

$$P(l) : \quad \mathcal{S}^{\geq_\tau}(l) \Rightarrow Prog(l)_t = MaxEr(Wind(l,\ t))$$

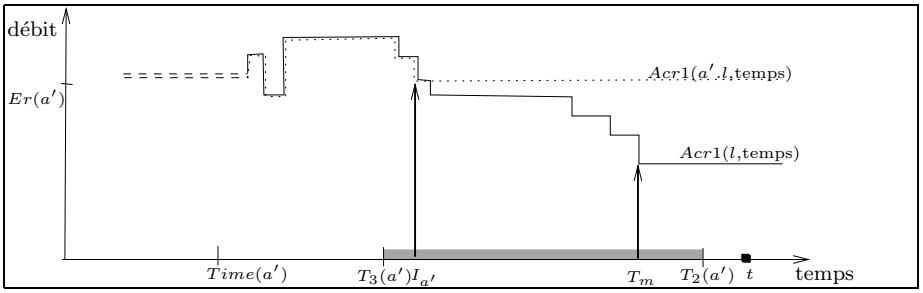

FIG. 8.2 – $T_2(a') \leq t$

On applique la récurrence sur la longueur de l. Lorsque l est vide, la preuve est immédiate par l'expansion des définitions de $Prog$, $MaxEr$ et $Wind$. Pour montrer le pas de récurrence, on suppose $P(l)$ et on essaie de prouver $P(a' \cdot l)$. Plus précisément, il faut prouver que $Prog(a' \cdot l)_t = MaxEr(Wind(a' \cdot l,\, t))$ est obtenu sachant que $\mathcal{S}^{\geq\tau}(a' \cdot l)$. A partir de l'hypothèse $\mathcal{S}^{\geq\tau}(a' \cdot l)$, on déduit $\mathcal{S}^{\geq\tau}(l)$ car l est une sous-liste de $(a' \cdot l)$. Par conséquent, on peut assumer que $Prog(l)_t = MaxEr(Wind(l,\, t))$. Les arguments pour prouver le pas de récurrence sont aussi basés sur la propriété suivante, qui dépend de l'hypothèse de récurrence :

Propriété 8.6.3 *La liste $Prog(l)_{\frown T_3(a')}$ est strictement croissante par le débit.*

Preuve Selon la propriété 8.6.2, on a $\mathcal{S}^{\geq\tau}(l) \Rightarrow \mathcal{S}^{<\rho}(Prog(l)_{\frown(Timel(l)+\tau_3)})$. A partir de l'hypothèse $\mathcal{S}^{\geq\tau}(l)$, il résulte $\mathcal{S}^{<\rho}(Prog(l)_{\frown(Timel(l)+\tau_3)})$. On a aussi $Time(a') \geq Timel(l)$ car $\mathcal{S}^{\geq\tau}(a' \cdot l)$ et, par conséquent, $Prog(l)_{\frown T_3(a')}$ est un préfixe de $Prog(l)_{\frown(Timel(l)+\tau_3)}$. Donc $Prog(l)_{\frown T_3(a')}$ est une liste strictement croissante par le débit.

Fin de preuve

Par T_m, on dénote $Prog(l)_t$. Alors on a $t \geq T_m$ et on déduit qu'il n'y a pas de cellule de la liste $Prog(l)$ ayant son instant dans l'intervalle $(T_m,\, t]$. Soit $I_{a'}$ l'instant de la première cellule de $Prog(a \cdot l)$. On a les faits suivants sur $I_{a'}$.

1. $I_{a'} \in \{T_2(t'),\, T_3(t') \mid t'$ est l'instant d'une cellule $(a' \cdot l)\}$, comme on peut facilement prouver par récurrence sur l.
2. $I_{a'} \in [T_3(a'),\, T_2(a')]$. Comme $a' \cdot l$ est temporellement décroissante, à partir du premier fait, il résulte que $I_{a'} \leq T_2(a')$. De plus, à partir de la définition $Prog$: (i) si $Er(a') < Acr1(l,\, T_3(a'))$ alors le préfixe $Prog(l)_{\leq_\rho Er(a')}$ contient seulement des cellules avec des débits $\leq Er(a')$ qui, par conséquent, ont des instances $> T_3(a')$. Si $Prog(l)_{\leq_\rho Er(a')}$ est vide, alors $I_{a'} = T_2(a')$, sinon $I_{a'}$ est l'instance de sa dernière cellule. Par conséquence, $I_{a'} > T_3(a')$. (ii) si $Er(a') \geq Acr1(l,\, T_3(a'))$ alors $I_{a'} = T_3(a')$.

Le domaine où $I_{a'}$ peut prendre des valeurs dans chaque cas est inclus dans l'intervalle temporel hachuré qu'on trouve dans la figure illustrant le cas respectif. A partir de la définition de $Prog$, on peut montrer que le débit de la première cellule de $Prog(a' \cdot l)$ est $Er(a')$. Donc, $Prog(a' \cdot l)$ est de la forme $(I_{a'},\, Er(a')) \cdot L^2$, pour quelque liste L^2.

On va effectuer une analyse par cas, selon la position de t par rapport aux valeurs $T_2(a')$ et $T_3(a')$:

1. si $T_2(a') \leq t$ (voir la figure 8.2) alors $Acr1(a' \cdot l,\, t) = Acr(a' \cdot l,\, t) (= Er(a'))$ car
   - $MaxEr(Wind(a' \cdot l,\, t)) = Er(a')$ parce que $Wind(a' \cdot l,\, t) = [a']$.

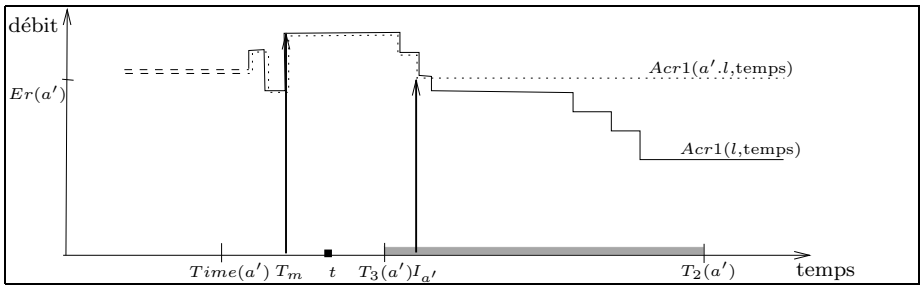

FIG. 8.3 – $T_3(a') > t$

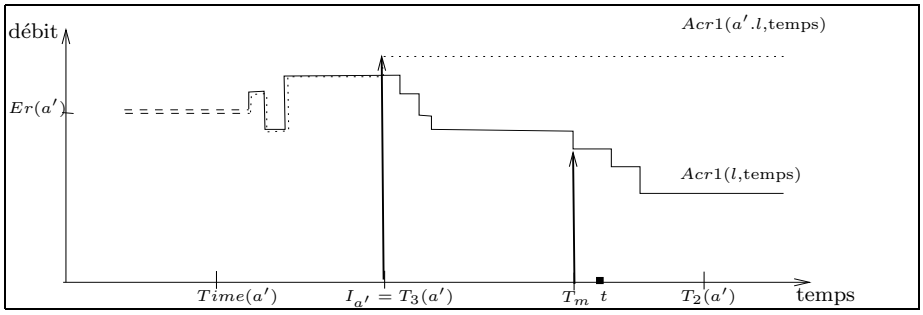

FIG. 8.4 – $T_2(a') > t \geq T_3(a')$ *et* $Acr1(l, T_3(a')) \leq Er(a')$

- $Prog(a' \cdot l)_t = Er(a')$ parce que $(I_{a'}, Er(a'))$ est la première cellule de $Prog(a' \cdot l)$ et $I_{a'} \leq T_2(a') \leq t$.

2. Si $T_3(a') > t$ (à voir la figure 8.3) alors

- a' n'est pas membre de $Wind(a' \cdot l, t)$. Par conséquent, $Acr(a' \cdot l, t) = Acr(l, t)$, et
- $Acr1(a' \cdot l, t) = Acr1(l, t)$ car $I_{a'} \in [T_3(a'), T_2(a')]$.

Par l'hypothèse de récurrence, $Acr1(l, t) = Acr(l, t)$. Il en résulte que $Acr1(a' \cdot l, t) = Acr(a' \cdot l, t)$.

3. Si $T_2(a') > t \geq T_3(a')$ on déduit que $a' \in Wind(a' \cdot l, t)$, par la définition de *Wind*.

3.1. Si $Wind(l, t)$ est vide, alors la liste l l'est aussi. Sinon, la première cellule de la liste l appartient à $Wind(l, t)$. Alors, $Prog([a']) = [(Er(a'), T_3(a'))]$ car $t \geq T_3(a')$ et $Wind([a'], t) = [a']$. Donc $Acr1([a'], t) = Acr([a'], t) = Er(a')$.

3.2. Si $Wind(l, t)$ n'est pas vide, soit ER_m la valeur du débit maximal des cellules de $Wind(l, t)$. Conformément à l'hypothèse de récurrence $Acr(l, t) = Acr1(l, t)$. Donc, le débit de la cellule de $Prog(l)$ programmée à T_m est ER_m.

Afin de programmer le débit $Er(a')$, on fait une analyse par cas dictée par la définition de $Prog$:

3.2.1. Si la condition

$$Acr1(l, T_3(a')) \leq Er(a') \qquad (Cond.1)$$

est vraie (see Fig. 8.4), alors $I_{a'} = T_3(a')$. De plus, $(I_{a'}, Er(a'))$ est la première cellule de $Prog(a' \cdot l)$. Alors $Acr1(a' \cdot l, t) = Er(a')$ parce que $T_3(a') \leq t$.

Il reste à prouver que $Acr(a' \cdot l, t)$ est aussi égal à $Er(a')$ ou la condition équivalente $ER_m \leq Er(a')$.

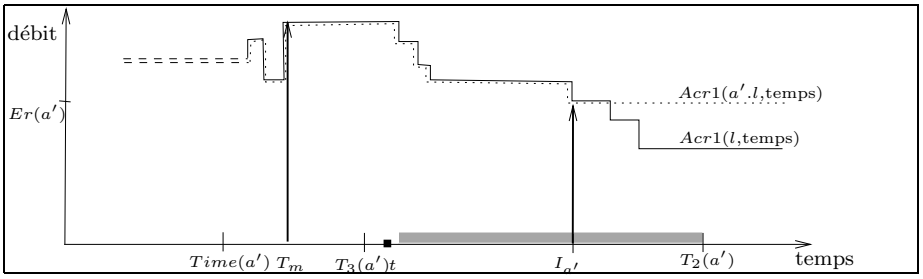

FIG. 8.5 – $T_2(a') > t \geq T_3(a')$, $Acr1(l, T_3(a')) > Er(a')$ *et* $T_m \leq T_3(a')$

De nouveau, à partir de $T_3(a') \leq t$ et le fait que la sous-liste $Prog(l)_{\frown T_3(a')}$ est strictement croissante par le débit, selon la propriété 8.6.3, on déduit que $Acr1(l, t) \leq Acr1(l, T_3(a'))$. Par conséquent, selon la condition $Cond.1$ et l'hypothèse de récurrence, on a $ER_m \leq Er(a')$.

3.2.2. Si la condition

$$Acr1(l, T_3(a')) > Er(a') \qquad (Cond.2)$$

est vraie, on obtient $I_{a'} > T_3(a')$. On distingue les cas suivants :

3.2.2.1 $T_m \leq T_3(a')$ (à voir la figure 8.5). Comme $T_3(a') \leq t$, il résulte que $t \geq T_3(a') \geq T_m$. Par la définition de T_m, il n'y a pas de cellule de $Prog(l)$ ayant son instant dans l'intervalle $(T_m, t]$. Donc, $Acr1(l, T_3(a')) = ER_m$. D'une part, par la condition $Cond.2$, on a $ER_m > Er(a')$. Par conséquent, $Acr(a' \cdot l, t) = ER_m$. D'autre part, l'instant $I_{a'}$ peut être soit $T_2(a')$ soit l'instant d'une cellule de $Prog(l)$. $T_2(a')$ ne peut pas être non plus comprise dans l'intervalle $(T_m, t]$ car $T_2(a') > t$. Si $I_{a'} = T_2(a')$ alors $I_{a'} > t$. Maintenant, on suppose que $I_{a'}$ est l'instant d'une cellule de $Prog(l)$. Comme $I_{a'} > T_3(a')$, $t \geq T_3(a') \geq T_m$ et il n'y a pas de cellule de $Prog(l)$ ayant son instant situé dans l'intervalle $(T_m, t]$, on déduit que $I_{a'} > t$. Par conséquent, $Acr1(a' \cdot l, t)$ est aussi égal à ER_m.

3.2.2.2. $T_m > T_3(a')$ (à voir la figure 8.6). Il résulte que $t \geq T_m > T_3(a')$. On a les cas suivants à considérer :

– si $Er(a') < ER_m$ alors $Acr(a' \cdot l) = ER_m$. Par la propriété 8.6.3, on déduit que $I_{a'} \geq T_m$. L'instant $I_{a'}$ peut être soit $T_2(a')$ soit l'instant d'une cellule de $Prog(l)$ qui est supérieure à t parce qu'il n'y a pas de cellules de $Prog(l)$ dont l'instant est dans l'intervalle $(T_m, t]$. Donc, $I_{a'} > t$ et on peut conclure que $Acr1(a' \cdot l, t) = Acr1(l, t) = ER_m$.

– si $Er(a') \geq ER_m$ alors $Acr(a' \cdot l) = Er(a')$. De nouveau par la propriété 8.6.3, on obtient $I_{a'} \leq T_m$. Soit $Prog(l) = Prog(l)_{\leq_\rho Er(a')} @ l^2$. Alors $(T_m, ER_m) \in Prog(l)_{\leq_\rho Er(a')}$. Il résulte que $Acr1(a' \cdot l, t) = Acr1(a' \cdot l, T_m) = (I_{a'}, Er(a')) \cdot l_t^2 = Er(a')$.

8.6.3 Commentaires sur la preuve automatisée avec PVS

Le théorème d'équivalence des algorithmes Acr et Acr1 a été développé avec succès en utilisant l'environnement de preuves PVS [Owre *et al.*, 1992]. PVS met à la disposition des utilisateurs un langage de spécification expressif qui se construit sur une logique typée d'ordre supérieur afin de définir des types de données abstraits. Par conséquent, la spécification des algorithmes, déjà fournis dans un style fonctionnel, a été relativement facile. Dans ce travail, on s'est restreint à une spécification de premier ordre parce que notre intention était de prouver le théorème d'équivalence avec SPIKE d'une manière plus automatique.

La première étape difficile a été rencontrée lorsqu'on a voulu prouver que si une liste de cellules l est temporellement décroissante, alors la liste programmée adjacente $Prog(l)$ l'est

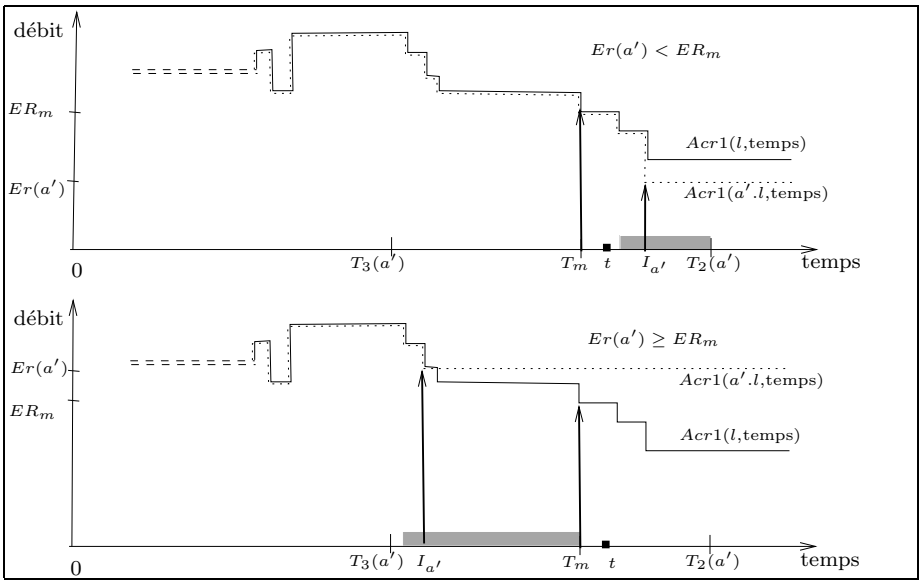

FIG. 8.6 – $T_2(a') > t \geq T_3(a')$, $Acr1(l, T_3(a')) > Er(a')$ *et* $T_m > T_3(a')$

aussi, conformément à la propriété 8.6.1. On a essayé d'appliquer la récurrence sur la longueur de la liste sans succès car, au moment de la preuve du pas de récurrence, la liste programmée de la conclusion de la récurrence n'est pas une sous-liste de celle de l'hypothèse de récurrence. Après une analyse plus approfondie du problème, on a découvert la nécessité d'avoir des lemmes auxiliaires concernant les propriétés sur les instants des cellules de la liste. Pourtant, les fonctions initiales n'ont pas été suffisantes pour les exprimer. C'est pour cette raison qu'on a pris la décision d'enrichir la spécification initiale par des fonctions auxiliaires comme `Timel`.

La vérification formelle de `time_dec` suit de près la preuve semi-formelle de la section 8.6.2. Par ses règles ASSERT et GRIND, PVS applique des procédures de décision pour faire du raisonnement arithmétique, utilise des procédures qui calculent la clôture par congruence pour raisonner sur les égalités, installe des théories et des règles de réécriture ainsi que des définitions liées à la conjecture afin de prouver les cas triviaux, de simplifier des expressions complexes et des définitions, ou pour faire du filtrage. Plus en détail, la preuve formelle de la propriété 8.6.1 demande 32 étapes dirigées par l'utilisateur PVS, dont 6 sont des applications de ASSERT, 5 de GRIND et 4 de LEMMA. La règle LEMMA permet aux instances de lemmes prouvés antérieurement d'être utilisés dans la preuve courante.

La preuve de la conjecture principale `main_conj` est de loin la plus complexe qu'on a expérimentée tout au long de notre preuve d'équivalence. Elle suit le squelette de la preuve semi-formelle décrite dans la section 8.6.2. On a d'abord appliqué la règle INDUCT pour effectuer de la récurrence sur la longueur de la liste de cellules. Le cas de base a été facilement accompli avec une opération GRIND. Pourtant, la preuve du pas de récurrence a demandé 27 invocations de lemmes et 6 fois l'application de la règle CASE afin d'appliquer le raisonnement par cas. L'analyse de chaque cas particulier a été une source de développement de nouveaux lemmes, qui à leur tour, peuvent avoir besoin de nouveaux lemmes pour accomplir leurs preuves. La profondeur du graphe de dépendance des lemmes est 7.

L'analyse de certains cas particuliers présentés dans la preuve du lemme `main_conj` a suggéré d'autres fonctions auxiliaires afin d'exprimer des propriétés supplémentaires. Par exemple, les

fonctions `ListUpTo(l,t)` et `SortedE(l)`, définissent respectivement les notations $l_{\frown t}$ et $\mathcal{S}^{<_\rho}(l)$ introduites dans la section 8.6.1. Certains cas de la preuve formelle suivent de près la preuve à la main, comme les cas 1, 2, 3.1 et 3.2.1. Les cas 3.2.2.1 et 3.2.2.2 sont plus complexes et ont demandé l'utilisation de 17 lemmes. La preuve de `main_conj` contient 120 étapes guidées par l'utilisateur, dont sept représentent des applications de la règle GRIND et 29 de la règle ASSERT. Ceci indique que le raisonnement arithmétique et équationnel ont été intensivement utilisés.

Dans les tableaux 8.1 et 8.2, on présente quelques statistiques sur les démarches des preuves de lemmes de cette spécification afin de donner une idée de leur taille et de leur difficulté. L'ordre des lemmes dans les tableaux satisfont la convention suivante : si le lemme a utilise le lemme b alors b se trouve au-dessus de a dans les tableaux. Les deux premières colonnes représentent l'index et le nom du lemme concerné. Les cinq colonnes suivantes illustrent respectivement le nombre d'invocations de lemmes et le nombre d'applications des règles INDUCT, ASSERT, GRIND et CASE pendant la preuve du lemme. Les colonnes huit et neuf montrent respectivement le nombre d'interactions avec l'utilisateur et le nombre d'appels des lemmes précédents. Sur la dernière ligne, on présente le total global de ces statistiques. Les lemmes des neuf premières lignes représentent des obligations de preuves qui doivent être déchargées afin que la théorie soit typée. Ils subsument dix autres obligations de preuve engendrées pendant le processus de typage et ont été prouvées complètement automatiquement par PVS.

La preuve globale dure 78.97s sur un PC disposant d'un processeur Intel Pentium II à 333 MHz et de 128 Mbytes de mémoire vive. Elle a été conçue en environ deux mois, en tenant compte du temps nécessaire pour se familiariser avec PVS.

8.6.4 Automatisation de la preuve avec SPIKE

Nous avons essayé SPIKE, intégrant la procédure de décision pour l'arithmétique linéaire comme décrit dans le chapitre 6, sur les lemmes fournis par l'utilisateur dans la preuve avec PVS. Pour cela, nous avons représenté les deux algorithmes sous la forme de la spécification conditionnelle de l'annexe B.

Les lemmes qui ont été prouvés complètement automatiquement sont marqués avec une étoile dans la dernière colonne des tableaux 8.1 et 8.2. Ainsi, 49 lemmes ont été prouvés sans l'aide de l'utilisateur. Si PVS avait intégré les techniques de preuve de SPIKE, alors la preuve aurait demandé 646 moins d'interactions.

Notons que tout au long des preuves des lemmes, nous avons utilisé la même stratégie, présentée dans le champ `Strategy`. D'abord, on essaie d'éliminer la conjecture courante avec la procédure de décision pour l'arithmétique linéaire. En cas d'échec, on applique des règles de SPIKE instanciant A-DELETE et A-SIMPLIFY : TAUTOLOGY, NEGATIVE CLASH, SUBSOMPTION, POSITIVE DECOMPOSITION, NEGATIVE DECOMPOSITION, POSITIVE CLASH. Puis, on applique des règles très simples instanciant A-SIMPLIFY, comme ELIMINATE REDUNDANT LITERAL, ELIMINATE TRIVIAL LITERAL et AUTO SIMPLIFICATION : si la conjecture courante contient deux littéraux identiques, alors ELIMINATE REDUNDANT LITERAL élimine une occurrence ; si la conjecture courante est de la forme $\bigwedge_i a_i = b_i \Rightarrow l_1 \vee \ldots \vee l_n$, alors AUTO SIMPLIFICATION permet de réécrire l_j ($j \in [1..n]$) avec des règles de réécriture obtenues par l'orientation des équations $a_i = b_i$. ELIMINATE TRIVIAL LITERAL est similaire à ELIMINATE TRIVIAL EQUATION. Si ces opérations échouent aussi, on applique des variantes de CASE SIMPLIFY et SIMPLIFY comme CONDITIONAL REWRITING, EQUATIONAL REWRITING, CONTEXTUAL REWRITING, PARTIAL

#	nom du lemme	#LEMMA	#INDUCT	#ASSERT	#GRIND	#CASE	#i.u.	#appels	SPIKE
1	MemberC_TCC1	0	0	0	0	0	0	0	
2	MemberT_TCC1	0	0	0	0	0	0	0	
3	MemberE_TCC1	0	0	0	0	0	0	0	
4	SortedT_TCC1	0	0	0	0	0	0	0	
5	SortedE_TCC1	0	0	0	0	0	0	0	
6	ListUpTo_TCC1	0	0	0	0	0	0	0	
7	Wind_TCC1	0	0	0	0	0	0	0	
8	Wind_TCC2	0	0	0	0	0	0	0	
9	MaxEr_TCC1	0	0	0	0	0	0	0	
10	firstat_timeat	0	1	0	1	0	9	1	*
11	firstat_progat	0	1	0	1	0	9	2	*
12	sorted_sorted	0	1	0	2	0	4	48	*
13	sorted_insat1	1	1	2	2	0	11	2	*
14	sorted_insin2	1	1	0	2	0	6	2	*
15	sorted_e_two	0	1	0	1	0	6	9	*
16	sorted_e_insin	1	1	2	2	0	16	1	*
17	sorted_e_member	2	1	2	3	0	17	1	
18	member_t_insin	0	1	2	4	0	19	1	*
19	member_t_insat	0	1	0	3	0	8	1	
20	member_less	1	1	3	2	0	14	2	
21	member_insin_time	2	1	4	3	0	27	1	
22	member_firstat	0	1	3	1	0	16	2	*
23	timel_insat_t	0	1	0	2	0	5	2	*
24	timel_insin	2	1	0	3	0	15	7	
25	erl_insin	0	1	0	2	0	5	1	*
26	erl_insat	0	1	0	2	0	5	1	*
27	erl_prog	2	1	0	3	0	12	1	*
28	time_progat_er	0	1	0	1	0	10	2	*
29	timeat_tcrt	1	1	1	1	0	11	2	*
30	timeat_timel	1	1	2	2	1	14	6	
31	timel_timeat_max	1	1	3	2	1	15	1	*
32	sorted_timeat	2	1	5	2	1	23	1	
33	sorted_timel_timeat	2	1	5	2	1	20	1	
34	timel_prog_conj1	3	1	4	2	1	23	1	
35	time_dec	4	1	6	5	0	32	9	
36	timel_prog	4	1	6	2	1	27	6	
37	null_insat	0	1	1	1	0	4	1	*
38	null_insin	0	1	1	1	0	4	1	*
39	null_prog	2	1	5	2	0	20	3	

TAB. 8.1 – *Quelques statistiques concernant la preuve globale*

#	nom du lemme	#LEMMA	#INDUCT	#ASSERT	#GRIND	#CASE	#i.u.	#appels	SPIKE
40	null_listat	0	1	2	1	0	12	1	*
41	null_listat1	0	1	0	1	0	3	1	*
42	cons_insat	0	1	0	2	0	11	1	*
43	cons_listat	0	1	0	1	0	3	1	*
44	progat_two_timeat	5	1	9	2	2	41	2	*
45	progat_timel_erl	1	1	1	2	0	14	1	*
46	progat_insat	1	1	1	3	0	16	1	*
47	progat_insat1	1	1	1	1	0	18	3	*
48	progat_insin_timeat	1	1	5	2	1	26	5	
49	progat_insin	2	1	6	2	1	32	3	*
50	listat_insin_tcrt	2	1	6	2	1	32	1	*
51	progat_insin_t	1	1	6	1	0	22	1	*
52	listupto1_erl	0	1	2	1	0	11	1	*
53	null_listupto	0	1	1	2	0	11	1	*
54	listupto_t_insat	1	1	3	1	0	19	1	*
55	listupto_insin_tcrt	3	1	9	2	0	50	1	
56	sorted_e_listupto	4	1	9	4	3	39	3	
57	timel_listupto	0	1	0	1	0	6	1	*
58	sorted_listupto	2	1	5	3	0	37	1	*
59	progat_listupto	1	1	4	1	0	28	2	*
60	leftmax	3	0	2	0	0	9	1	*
61	leftmax_max	1	1	2	3	1	16	1	*
62	right_prog	14	0	14	2	1	54	1	
63	right_wind	0	0	0	0	0	5	1	*
64	time_listat	1	1	1	1	0	12	2	*
65	listat_listupto	2	1	6	2	0	39	1	*
66	sorted_cons_listat	1	1	4	1	0	17	1	*
67	progat_member_time	7	1	16	3	2	49	1	
68	sorted_cons_two	11	1	12	2	1	48	1	
69	sorted_cons_two_nil	1	1	1	1	0	7	1	
70	sorted_e_progat_prog_two	10	1	9	4	2	47	2	
71	rate_inc	1	1	1	1	0	7	1	
72	sorted_e_member_two	6	1	11	3	2	42	1	
73	new_listat_prog	4	1	10	3	1	47	1	
74	progat_prog_two	17	1	23	5	5	83	1	
75	null_wind1	0	1	3	1	0	10	1	*
76	null_wind2	1	0	1	1	1	7	1	*
77	member_t_timel	0	1	4	1	0	17	1	
78	timeat_greater	2	1	7	2	1	28	1	
79	timel_insin1	0	0	0	1	0	1	1	*
80	null_listupto1	0	1	0	2	0	6	1	*
81	sorted_e_cons	0	1	1	1	0	6	1	*
82	erl_cons	0	1	0	1	0	5	1	*
83	no_time	0	1	4	1	0	16	1	*
84	prev_time	3	1	10	3	1	36	1	
85	monoton_e	7	1	11	5	1	55	1	
86	monoton_e1	4	1	10	4	1	49	1	
87	main_conj	27	1	29	7	6	120	1	
88	final	1	0	1	0	0	10	0	*
	Total	**181**	**73**	**320**	**158**	**40**	**1686**	**181**	**49**

TAB. 8.2 – *Quelques statistiques concernant la preuve globale (suite)*

Case Rewriting et Total Case Rewriting. Les appels récursifs au prouveur, demandés par Conditional Rewriting, Contextual Rewriting et Total Case Rewriting, créent des preuves suivant respectivement les stratégies `normalize`, `rec_strategy_rewriting_level0` et `delete_set`. La stratégie `normalize` est prédéfinie et permet de normaliser une conjecture par des applications successives de Conditional Rewriting. Finalement, si aucune de ces règles ne peut s'appliquer, on utilise Generate. L'état courant de la preuve s'affiche par la stratégie `print_goals`.

Un exemple

On va détailler la preuve SPIKE du lemme `sorted_listupto` de l'annexe B.2, nécessaire pour résoudre le cas 3.2.1. de la preuve globale de la section 8.6.2.

`sorted_listupto` affirme que, étant données une instance arbitraire t et une liste temporellement décroissante l, le préfixe maximal l' de l contenant des cellules avec des instances supérieures à t est aussi temporellement décroissant :

$$SortedT(l) = True \Rightarrow SortedT(ListUpTo(l,\, t)) = True$$

Dans la preuve avec SPIKE intégrant la procédure de décision pour l'arithmétique linéaire on utilise 2 opérations Generate, 51 opérations de réécriture et 10 opérations de subsomption. La preuve commence par une Generate qui instancie la variable l par Nil, $Cons(t_1/e_1,\, nil)$ et $Cons(t_2/e_2,\, Cons(y,\, z))$ et puis les réécrit avec des règles de réécriture conditionnelles obtenues par l'orientation des axiomes de gauche à droit. L'ordre sur les termes et clauses est construit à partir des précédences entre les symboles de fonction donnés dans le champ `less` de la spécification SPIKE.

Il y a une seule conjecture qui ne peut être éliminée pendant le processus de simplification :

$$SortedT(cons(y,\, z)) = True,\, Time(y) \leq t_2 = True,\, t_2 \leq t = False,$$
$$Time(y) \leq t = False \Rightarrow SortedT(cons(y,\, ListUpTo(z,\, t))) = True$$

SPIKE applique Generate sur elle. Cette fois-ci, toutes les conjectures dérivées sont éliminées.

Notons que la même version de SPIKE, mais sans intégrer la procédure de décision pour l'arithmétique linéaire, diverge sur cet exemple.

8.7 Conclusions

Notre objectif a été de dériver la première preuve automatisée d'un algorithme incrémental et idéal qui vérifie la conformité du protocole ABR. Une preuve « à la main » a été conçue auparavant dans [Rabadan et Klay, 1997]. Pourtant, ce type de preuves manuelles n'est pas convaincant en général car des cas limite ou des arguments apparemment triviaux sont souvent omis, ceci constituant des sources permanentes d'erreurs. Dans cette direction, notre preuve automatisée présente plus de confiance dans la correction de l'algorithme car chaque étape a été vérifiée par PVS.

D'une part, la spécification de l'algorithme dans PVS comme fonction récursive s'est avéré être une tâche relativement facile. D'autre part, la preuve qu'on a obtenue est difficile. Elle

demande la définition et la preuve d'approximativement 80 lemmes intermédiaires et l'introduction des fonctions auxiliaires. Contrairement à d'autres preuves formelles, qui servent à valider des preuves manuelles en suivant de près leur squelette, on a été contraint, en raison de notre approche de premier ordre, de renoncer à la preuve de [Rabadan et Klay, 1997] et de construire une autre preuve sans avoir une ligne directrice générale établie. L'interactivité avec l'utilisateur, spécifique à PVS, nous a permis d'isoler des cas critiques afin de se concentrer sur eux. La plupart des lemmes intermédiaires ont été testés sur des cas limite avant d'être prouvés formellement. Ces tests ont été simulés avec un programme écrit en Ocaml [Leroy *et al.*, 2000] qui met en œuvre les algorithmes Acr et Acr1. Les tests ont été effectués sur des configurations concrètes de la liste de cellules, avant de prouver formellement les lemmes avec PVS.

Nous avons réussi avec les techniques de preuve par ensembles couvrants contextuels de SPIKE, intégrant une procédure de décision pour l'arithmétique linéaire comme décrit dans le chapitre 6, de prouver complètement automatiquement plus de 60% des lemmes donnés par l'utilisateur. Ceci montre qu'une coopération entre PVS et SPIKE aurait réduit l'interaction avec l'utilisateur dans la preuve globale à plus d'un tiers.

Sommaire et perspectives

Le travail de la thèse est partagé en deux parties : théorie et applications. La partie théorique est centrée autour d'un nouveau concept, celui d'ensemble couvrant contextuel (ECC), qui caractérise complètement, d'une part des schémas de récurrence explicite comme les ensembles couvrants et d'autre part des techniques de simplification comme celles spécifiques à l'approche de la preuve par cohérence. Le noyau du principe de preuve par récurrence avec ECC est reflété par un système d'inférence abstrait. Le système est descriptif et générique et établit des normes pour l'usage des ECC à la construction des règles d'inférences. Par rapport à d'autres systèmes abstraits d'inférences similaires, notre système contient des règles comprenant des conditions d'applications plus faibles et généralise virtuellement toutes les procédures de récurrence implicite existantes basées sur des ensembles couvrants. D'une part, il peut être facilement adapté pour montrer des propriétés inductives, initiales et observationnelles dans les spécifications conditionnelles, pour faire des preuves dans les spécifications conditionnelles paramétrées ou bien dans les spécifications conditionnelles de type positif/négatif. D'autre part, il peut aussi être adapté à prouver si une spécification est suffisamment complète ou un système de réécriture est convergent ou confluent sur les termes clos.

En pratique, les ECC élémentaires sont engendrés par des modules de raisonnement, représentant des implantations des techniques de raisonnement. Grâce à leurs propriétés de composition, on peut engendrer de nouveaux ECC. Les modules de raisonnement sont intégrés dans le prouveur selon un schéma qui présente l'avantage d'utiliser des éléments du contexte courant en tant que prémisses. Comme étude de cas, nous avons défini un ensemble de modules de raisonnement à partir d'un ensemble de techniques de réécriture adéquates aux théories conditionnelles, comme la réécriture inductive ou la réécriture par cas. Sur cette base, nous avons spécifié le système d'inférence du prouveur SPIKE comme une instance de notre système abstrait adaptée à la vérification des propriétés initiales. Le potentiel offert par le cadre de travail de la récurrence avec ECC a permis non seulement des modifications faciles et des généralisations de ses règles d'inférence, ou un schéma de parallélisation du système d'inférence, mais aussi des extensions modulaires et incrémentales du système d'inférence en introduisant de nouvelles techniques de raisonnement. Par exemple, nous avons montré comment SPIKE, à l'aide d'une nouvelle technique de raisonnement sur des clauses (la subsomption sémantique inductive) et intégrant une procédure de semi-décision pour l'arithmétique linéaire, a réussi la preuve de correction de l'algorithme MJRTY en utilisant une combinaison de raisonnement inductif et arithmétique.

A court terme, j'envisage d'apporter à SPIKE les améliorations et les fonctionnalités que j'ai mentionné dans la thèse. En priorité, le travail sera porté sur l'intégration des techniques qui permettront plus d'automatisation et d'efficacité des preuves. En particulier, j'envisage le remplacement de la procédure de semi-décision pour l'arithmétique linéaire avec une procédure de décision. De plus, jqe vais étudier les conditions nécessaires à l'intégration d'autres procédures de décision, par exemple pour des listes, vecteurs de bits, tableaux, ...

L'interface parallèle SPIKEPAR que j'ai conçu pour SPIKE implémente un schéma de parallélisation au niveau des clauses. Il serait intéressant d'étudier le gain d'efficacité de SPIKE en utilisant des schémas plus fins, permettant le parallélisme au niveau des termes [Kirchner et Viry, 1992; Bündgen *et al.*, 1996], ou des stratégies de déduction [Bonacina et Hsiang, 1994].

J'envisage aussi la recherche d'autres applications qui peuvent bénéficier de notre cadre de preuve par ECC. Je crois que des systèmes d'inférence mettant en œuvre des techniques de raisonnement comme la résolution ordonnée [Loveland, 1978] ou des techniques basées sur l'arithmétique, comme celles présentées dans l'algorithme de Buchberger [Bachmair et Ganzinger, 1994] pour le calcul de bases de Gröbner [Becker et Weispfenning, 1993] dans des anneaux polynômiaux dans les algèbres, sont de bons candidats. Juste pour donner une idée, voici un exemple d'utilisation de la résolution ordonnée dans la génération des ECC.

Soient $C_1 \equiv l \vee r_1$ et $C_2 \equiv \neg l \vee r_2$ deux clauses telles que l est un littéral maximal des deux clauses. Par la résolution de C_1 et C_2, on obtient la clause plus petite (par rapport à l'ordre sur les clauses $\preceq_c$) $C_3 \equiv r_1 \vee r_2$. Par conséquent, $\{C_3\}$ est un ECC strict de C_1 (resp. C_2) dans tout contexte permettant l'utilisation de C_2 (resp. C_1). Les modules de raisonnement mettant en œuvre ces techniques peuvent être ainsi intégrés dans toute instance opérationnelle du système d'inférence abstrait qui manipule des clauses.

Dans la partie pratique, nous avons présenté des applications de la méthode de preuve par récurrence avec ECC à la vérification des logiciels de télécommunications. Un premier exemple aborde le problème de la détection et la résolution des interactions de services téléphoniques. Nous avons proposé une méthodologie qui permet leur détection/résolution avec des techniques basées sur la réécriture conditionnelle et la récurrence implicite. Comme étude de cas, nous avons utilisé avec succès cette méthodologie, en nous appuyant sur le démonstrateur SPIKE, pour analyser l'interopérabilité des services de renvoi inconditionnel et de filtrage des appels à l'arrivée, aussi que celle des services de numérotation abrégée et filtrage des appels au départ. Dans le futur, j'envisage l'analyse d'autres combinaisons de services téléphoniques avec notre méthodologie.

Dans un autre exemple, nous avons obtenu la première preuve automatisée, à l'aide du démonstrateur PVS, de la conformité d'un algorithme incrémental qui calcule les débits admis par le protocole ABR dans les réseaux ATM. Avec notre méthode de preuve, nous avons réussi, par l'intermédiaire de SPIKE, à vérifier complètement automatiquement la majorité des lemmes de cette preuve. L'expérience que nous avons eu au long de la preuve avec PVS a montré l'importance de l'interactivité du prouveur avec l'utilisateur. Grâce à elle, nous avons réussi à guider, étape par étape, la construction des preuves et isoler les cas critiques afin de nous concentrer sur eux.

Je crois que le succès d'un prouveur dépend fortement de l'équilibre, pendant les preuves, entre l'interactivité et les dérivations complètement automatiques. A long terme, j'envisage de travailler dans les deux directions pour rapprocher ces deux fonctionnalités dans le cadre des preuves par ECC. D'une part, je vais étudier les conséquences de l'utilisation de formules arbitraires qui peuvent ainsi perturber l'ordre global sur les formules, nécessaire aux dérivations des preuves par récurrence implicite. Ceci permettra une liberté de choix plus large pour l'utilisateur pendant les analyses par cas ou lorsqu'il instancie des lemmes. D'autre part, je vais étudier la possibilité d'intégrer des techniques de preuve facilement automatisables, en particulier celles basées sur la récurrence implicite, dans des prouveurs interactifs comme PVS.

A

Spécification composée des services CFU et TCS

Cette annexe contient la spécification SPIKE de l'inter-fonctionnement des services CFU et $POTS$, présentés dans le chapitre 7, qui est conçue pour une configuration de réseau à trois utilisateurs. Elle a été obtenue par l'application de la méthodologie présentée dans la section 7.5.

specification : TCS_CFU_POTS

sorts $bool$, Nat^M, N^M, Com^M

constructors :

0	:		$\rightarrow$	Nat^M
1	:		$\rightarrow$	Nat^M
2	:		$\rightarrow$	Nat^M
3	:		$\rightarrow$	Nat^M
$True$	:		$\rightarrow$	$bool$
$False$	:		$\rightarrow$	$bool$
Tel^M	:	$Nat^M \times Nat^M \times Nat^M \times (Nat^M)^* \times bool$	$\rightarrow$	N^M
TCS^M	:	$Nat^M \times Nat^M$	$\rightarrow$	Com^M
CFU^M	:	$Nat^M \times Nat^M$	$\rightarrow$	Com^M
$COMM^M$	:	$Nat^M \times Nat^M$	$\rightarrow$	Com^M

defined functions :

% fonctions définies %

$\wedge$	:	$bool \times bool$	$\rightarrow$	$bool$
$member$	:	$Nat^M \times (Nat^M)^*$	$\rightarrow$	$bool$
$busy$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$bool$
$disconnect$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$

% –$CFU/POTS$– les fonctions utilisées pour spécifier le service CFU %

$busy^C$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$bool$
$disconnect^C$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$member^C$	:	$Nat^M \times (Nat^M)^*$	$\rightarrow$	$bool$
$transit^C$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	Nat^M
$Apply\_network^C$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$Apply\_aux^C$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$connect^C$	:	$Nat^M \times Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$App1\_network^C$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
App_tel^C	:	$Com^M \times N^M$	$\rightarrow$	N^M
$Accepts^C$	:	$(Com^M)^* \times (N^M)^*$	$\rightarrow$	$bool$
$Ok\_network^C$	:	$(N^M)^*$	$\rightarrow$	$bool$
$no\_selfcall$	:	N^M	$\rightarrow$	$bool$

% –$TCS/POTS$– les fonctions utilisées pour spécifier le service TCS %

$busy^T$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$bool$
$disconnect^T$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$member^T$	:	$Nat^M \times (Nat^M)^*$	$\rightarrow$	$bool$
$Apply\_network^T$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$connect^T$	:	$Nat^M \times Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$App1\_network^T$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
App_tel^T	:	$Com^M \times N^M$	$\rightarrow$	N^M
$Accepts^T$	:	$(Com^M)^* \times (N^M)^*$	$\rightarrow$	$bool$
$Ok\_network^T$	:	$(N^M)^*$	$\rightarrow$	$bool$
$no\_call\_black\_list$	:	N^M	$\rightarrow$	$bool$

% –$(CFU \oplus TCS)/POTS$– les fonctions de la spécification composée%

$busy^M$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$bool$
$disconnect^M$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$member^M$	:	$Nat^M \times (Nat^M)^*$	$\rightarrow$	$bool$
$transit^M$	:	$Nat^M \times (N^M)^*$	$\rightarrow$	Nat^M
$Apply\_network^M$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$Apply\_aux^M$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$connect^M$	:	$Nat^M \times Nat^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
$App1\_network^M$	:	$Com^M \times (N^M)^*$	$\rightarrow$	$(N^M)^*$
App_tel^M	:	$Com^M \times N^M$	$\rightarrow$	N^M
$Accepts^M$	:	$(Com^M)^* \times (N^M)^*$	$\rightarrow$	$bool$
$Ok\_network^M$	:	$(N^M)^*$	$\rightarrow$	$bool$
$no\_call\_bl\_himself$	:	N^M	$\rightarrow$	$bool$

axioms :

% fonctions partagées %

% **logical and** %

$True \wedge True$			$\rightarrow$	$True$
$False \wedge x$			$\rightarrow$	$False$
$x \wedge False$			$\rightarrow$	$False$

% **member(n,l)** %

		$member(n,[])$	$\rightarrow$	$False$
$n_1 = n_2$	$\Rightarrow$	$member(n_1,[n_2@l])$	$\rightarrow$	$True$
$n_1 \neq n_2$	$\Rightarrow$	$member(n_1,[n_2@l])$	$\rightarrow$	$member(n_1,l)$

% **busy(n,r)** %

		$busy(n,[])$	$\rightarrow$	$False$
$n = n_1 \wedge n_2 \neq 0$	$\Rightarrow$	$busy(n,[Tel^M(n_1,n_2,n_3,ln,f)@r])$	$\rightarrow$	$True$
$n \neq n_1$	$\Rightarrow$	$busy(n,[Tel^M(n_1,n_2,n_3,ln,f)@r])$	$\rightarrow$	$busy(n,r)$
$n_2 = 0$	$\Rightarrow$	$busy(n,[Tel^M(n_1,n_2,n_3,ln,f)@r])$	$\rightarrow$	$busy(n,r)$

% **disconnect(n,r)** %

		$disconnect(n,[]) \rightarrow []$
$n = n_1$	$\Rightarrow$	$disconnect(n,[Tel^M(n_1,n_2,n_3,ln,f)@r]) \rightarrow$ $[Tel^M(n_1,0,n_3,ln,False)@disconnect(n,r)]$
$n = n_2$	$\Rightarrow$	$disconnect(n,[Tel^M(n_1,n_2,n_3,ln,f)@r]) \rightarrow$ $[Tel^M(n_1,0,n_3,ln,False)@disconnect(n,r)]$
$n \neq n_1 \wedge n \neq n_2$	$\Rightarrow$	$disconnect(n,[Tel^M(n_1,n_2,n_3,ln,f)@r]) \rightarrow$ $[Tel^M(n_1,n_2,n_3,ln,f)@disconnect(n,r)]$

% $CFU/POTS$ %

% **busy$^{\mathbf{C}}$(n,r)** %

$busy^C(n,r) \rightarrow busy(n,r)$

% **disconnect$^{\mathbf{C}}$(n,r)** %

$disconnect^C(n,r) \rightarrow disconnect(n,r)$

% **member$^{\mathbf{C}}$(n,l)** %

$member^C(n,l) \rightarrow member(n,l)$

% **transit$^{\mathbf{C}}$(n,r)** %

		$transit^C(n,[])$	$\rightarrow$	0
$n = n_1$	$\Rightarrow$	$transit^C(n,[Tel^M(n_1,n_2,n_3,ln,f)@r])$	$\rightarrow$	n_3
$n \neq n_1$	$\Rightarrow$	$transit^C(n,[Tel^M(n_1,n_2,n_3,ln,f)@r])$	$\rightarrow$	$transit^C(n,r)$

% **Apply_network^{C}(c,l)** %

$$
\begin{array}{lcl}
 & & Apply\_network^{C}(TCS^{M}(n_1,n_2),r) \rightarrow r \\
 & & Apply\_network^{C}(CFU^{M}(n_1,n_2),r) \rightarrow \\
 & & Apply\_aux^{C}(CFU^{M}(n_1,n_2),r) \\
transit^{C}(n_2,r) \neq 0 \wedge n_1 \neq n_2 & \Rightarrow & Apply\_network^{C}(COMM^{M}(n_1,n_2),r) \rightarrow \\
 & & Apply\_aux^{C}(COMM^{M}(n_1,transit^{C}(n_2,r)),r) \\
n_1 = n_2 & \Rightarrow & Apply\_network^{C}(COMM^{M}(n_1,n_2),r) \rightarrow r \\
transit^{C}(n_2,r) = 0 \wedge n_1 \neq n_2 & \Rightarrow & Apply\_network^{C}(COMM^{M}(n_1,n_2),r) \rightarrow \\
 & & Apply\_aux^{C}(COMM^{M}(n_1,n_2),r)
\end{array}
$$

% **Apply_aux^{C}(c,r)** %

$$
\begin{array}{lcl}
 & & Apply\_aux^{C}(TCS^{M}(n_1,n_2),r) \rightarrow r \\
 & & Apply\_aux^{C}(c,[]) \rightarrow [] \\
n_1 \neq 0 \wedge \neg busy^{C}(n_1,[t@r]) & \Rightarrow & Apply\_aux^{C}(CFU^{M}(n_1,n_2),[t@r]) \rightarrow \\
 & & [App\_tel^{C}(CFU^{M}(n_1,n_2),t)@Apply\_aux^{C}(CFU^{M}(n_1,n_2),r)] \\
n_1 = 0 & \Rightarrow & Apply\_aux^{C}(CFU^{M}(n_1,n_2),[t@r]) \rightarrow [t@r] \\
busy^{C}(n_1,[t@r]) & \Rightarrow & Apply\_aux^{C}(CFU^{M}(n_1,n_2),[t@r]) \rightarrow [t@r] \\
n_1 \neq 0 \wedge n_2 \neq 0 & \Rightarrow & Apply\_aux^{C}(COMM^{M}(n_1,n_2),[t@r]) \rightarrow \\
 & & connect^{C}(n_1,n_2,[t@r]) \\
n_1 = 0 & \Rightarrow & Apply\_aux^{C}(COMM^{M}(n_1,n_2),[t@r]) \rightarrow [t@r] \\
n_1 \neq 0 \wedge n_2 = 0 & \Rightarrow & Apply\_aux^{C}(COMM^{M}(n_1,n_2),[t@r]) \rightarrow \\
 & & disconnect^{C}(n_1,[t@r])
\end{array}
$$

% **connect^{C}($\mathbf{n_1}$,$\mathbf{n_2}$,r)** %

$$
\begin{array}{lcl}
 & & connect^{C}(n_1,n_2,[]) \rightarrow \\
 & & App1\_network^{C}(COMM^{M}(n_1,n_2),[]) \\
\neg busy^{C}(n_1,[t@r]) \wedge \neg busy^{C}(n_2,[t@r]) & \Rightarrow & connect^{C}(n_1,n_2,[t@r]) \rightarrow \\
 & & App1\_network^{C}(COMM^{M}(n_1,n_2),[t@r]) \\
busy^{C}(n_1,[t@r]) & \Rightarrow & connect^{C}(n_1,n_2,[t@r]) \rightarrow [t@r] \\
busy^{C}(n_2,[t@r]) & \Rightarrow & connect^{C}(n_1,n_2,[t@r]) \rightarrow [t@r]
\end{array}
$$

% **App1_network^{C}(c,r)** est toujours appelé ayant $COMM^{M}$ comme argument. Les autres commandes sont filtrées. %

$$
\begin{array}{lcl}
App1\_network^{C}(TCS^{M}(n_1,n_2),r) & \rightarrow & [] \\
App1\_network^{C}(CFU^{M}(n_1,n_2),r) & \rightarrow & [] \\
App1\_network^{C}(COMM^{M}(n_1,n_2),[]) & \rightarrow & [] \\
App1\_network^{C}(COMM^{M}(n_1,n_2),[t@r]) & \rightarrow & [App\_tel^{C}(COMM^{M}(n_1,n_2),t)@ \\
 & & App1\_network^{C}(COMM^{M}(n_1,n_2),r)]
\end{array}
$$

% **App_tel^{C}(c,t)** %

$$
\begin{array}{lcl}
 & & App\_tel^{C}(TCS^{M}(n_1,n_2),Tel^{M}(n_3,n_4,n_5,ln,f)) \rightarrow \\
 & & Tel^{M}(n_3,n_4,n_5,ln,t) \\
n_3 = n_1 & \Rightarrow & App\_tel^{C}(CFU^{M}(n_1,n_2),Tel^{M}(n_3,n_4,n_5,ln,f)) \rightarrow \\
 & & Tel^{M}(n_3,n_4,n_2,ln,f) \\
n_3 \neq n_1 & \Rightarrow & App\_tel^{C}(CFU^{M}(n_1,n_2),Tel^{M}(n_3,n_4,n_5,ln,f)) \rightarrow \\
 & & Tel^{M}(n_3,n_4,n_5,ln,f) \\
n_2 = n_3 & \Rightarrow & App\_tel^{C}(COMM^{M}(n_1,n_2),Tel^{M}(n_3,n_4,n_5,ln,f)) \rightarrow \\
 & & Tel^{M}(n_3,n_1,n_5,ln,f) \\
n_1 = n_3 & \Rightarrow & App\_tel^{C}(COMM^{M}(n_1,n_2),Tel^{M}(n_3,n_4,n_5,ln,f)) \rightarrow \\
 & & Tel^{M}(n_3,n_2,n_5,ln,f) \\
n_2 \neq n_3 \wedge n_1 \neq n_3 & \Rightarrow & App\_tel^{C}(COMM^{M}(n_1,n_2),Tel^{M}(n_3,n_4,n_5,ln,f)) \rightarrow \\
 & & Tel^{M}(n_3,n_4,n_5,ln,f)
\end{array}
$$

% **Accepts^{C}(lc,r)** %

$$
\begin{array}{lcl}
 & Accepts^{C}([],r) & \rightarrow \quad Ok\_network^{C}(r) \\
Ok\_network^{C}(r) \Rightarrow & Accepts^{C}([c@lc],r) & \rightarrow \quad Accepts^{C}(lc,Apply\_aux^{C}(c,r)) \\
\neg Ok\_network^{C}(r) \Rightarrow & Accepts^{C}(lc,r) & \rightarrow \quad False
\end{array}
$$

% **Ok_network^{C}(r)** %

$$
\begin{array}{lcl}
Ok\_network^{C}([]) & \rightarrow & True \\
Ok\_network^{C}([t@r]) & \rightarrow & no\_selfcall(t) \wedge Ok\_network^{C}(r))
\end{array}
$$

% **no_selfcall(t)** %

$$
\begin{array}{lcl}
n_1 \neq n_2 & \Rightarrow & no\_selfcall(Tel^{M}(n_1,n_2,n_3,ln,f)) \rightarrow True \\
n_1 = n_2 & \Rightarrow & no\_selfcall(Tel^{M}(n_1,n_2,n_3,ln,f)) \rightarrow False
\end{array}
$$

% $TCS/POTS$ %

% $\mathbf{busy^T(n,r)}$ %

$busy^T(n,r) \rightarrow busy(n,r)$

% $\mathbf{disconnect^T(n,r)}$ %

$disconnect^T(n,r) \rightarrow disconnect(n,r)$

% $\mathbf{member^T(n,l)}$ %

$member^T(n,l) \rightarrow member(n,l)$

% $\mathbf{Apply\_network^T(c,r)}$ %

		$Apply\_network^T(CFU^M(n_1,n_2),r) \rightarrow r$
		$Apply\_network^T(c,[]) \rightarrow []$
$n_1 \neq 0 \wedge n_2 \neq 0 \wedge \neg busy^T(n_2,[t@r])$	$\Rightarrow$	$Apply\_network^T(TCS^M(n_1,n_2),[t@r]) \rightarrow$ $[App\_tel^T(TCS^M(n_1,n_2),t)@$ $Apply\_network^T(TCS^M(n_1,n_2),r)]$
$n_1 = 0$	$\Rightarrow$	$Apply\_network^T(TCS^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_2 = 0$	$\Rightarrow$	$Apply\_network^T(TCS^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$busy^T(n_2,[t@r])$	$\Rightarrow$	$Apply\_network^T(TCS^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_1 \neq 0 \wedge n_1 \neq n_2 \wedge n_2 \neq 0$	$\Rightarrow$	$Apply\_network^T(COMM^M(n_1,n_2),[t@r]) \rightarrow$ $connect^T(n_1,n_2,[t@r])$
$n_1 = n_2$	$\Rightarrow$	$Apply\_network^T(COMM^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_1 = 0$	$\Rightarrow$	$Apply\_network^T(COMM^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_1 \neq 0 \wedge n_2 = 0$	$\Rightarrow$	$Apply\_network^T(COMM^M(n_1,n_2),[t@r]) \rightarrow$ $disconnect^T(n_1,[t@r])$

% $\mathbf{connect^T(n_1,n_2,r)}$ %

		$connect^T(n_1,n_2,[]) \rightarrow$ $App1\_network^T(COMM^M(n_1,n_2),[])$
$\neg busy^T(n_1,[t@r]) \wedge \neg busy^T(n_2,[t@r])$	$\Rightarrow$	$connect^T(n_1,n_2,[t@r]) \rightarrow$ $App1\_network^T(COMM^M(n_1,n_2),[t@r])$
$busy^T(n_1,[t@r])$	$\Rightarrow$	$connect^T(n_1,n_2,[t@r]) \rightarrow [t@r]$
$busy^T(n_2,[t@r])$	$\Rightarrow$	$connect^T(n_1,n_2,[t@r]) \rightarrow [t@r]$

% $\mathbf{App1\_network^T(c,r)}$ %

$App1\_network^T(CFU^M(n_1,n_2),r)$	$\rightarrow$	$[]$
$App1\_network^T(TCS^M(n_1,n_2),r)$	$\rightarrow$	$[]$
$App1\_network^T(COMM^M(n_1,n_2),[])$	$\rightarrow$	$[]$
$App1\_network^T(COMM^M(n_1,n_2),[t@r])$	$\rightarrow$	$[App\_tel^T(COMM^M(n_1,n_2),t)\ @App1\_network^T(COMM^M(n_1,n_2),r)]$

% $\mathbf{App\_tel^T(c,t)}$ %

		$App\_tel^T(CFU^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,ln,f)$
$n_2 = n_3 \wedge n_1 \neq n_4 \wedge \neg member^T(n_1,ln)$	$\Rightarrow$	$App\_tel^T(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,[n_1@ln],f)$
$n_2 \neq n_3$	$\Rightarrow$	$App\_tel^T(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,ln,f)$
$n_1 = n_4$	$\Rightarrow$	$App\_tel^T(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,ln,f)$
$member^T(n_1,ln)$	$\Rightarrow$	$App\_tel^T(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,ln,f)$
$n_1 \neq n_2 \wedge n_2 = n_3$	$\Rightarrow$	$App\_tel^T(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_1,n_5,ln,False)$
$n_1 \neq n_2 \wedge n_1 = n_3$	$\Rightarrow$	$App\_tel^T(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_2,n_5,ln,True)$
$n_2 \neq n_3 \wedge n_1 \neq n_3$	$\Rightarrow$	$App\_tel^T(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,ln,f)$
$n_1 = n_2$	$\Rightarrow$	$App\_tel^T(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow$ $Tel^M(n_3,n_4,n_5,ln,f)$

% $\mathbf{Accepts^T(lc,r)}$ %

		$Accepts^T([],r)$	$\rightarrow$	$Ok\_network^T(r)$
$Ok\_network^T(r)$	$\Rightarrow$	$Accepts^T([c@lc],r)$	$\rightarrow$	$Accepts^T(lc,Apply\_network^T(c,r))$
$\neg Ok\_network^T(r)$	$\Rightarrow$	$Accepts^T(lc,r)$	$\rightarrow$	$False$

% **Ok_network$^{\mathbf{T}}$(r)** %

$Ok\_network^T([]) \quad \rightarrow \quad True$
$Ok\_network^T([t@r]) \quad \rightarrow \quad no\_call\_black\_list(t) \wedge Ok\_network^T(r))$

% **no_call_black_list(t)** %

$\neg member^T(n_2,ln) \quad \Rightarrow \quad no\_call\_black\_list(Tel^M(n_1,n_2,n_3,ln,f)) \rightarrow True$
$member^T(n_2,ln) \quad \Rightarrow \quad no\_call\_black\_list(Tel^M(n_1,n_2,n_3,ln,f)) \rightarrow f$

% $CFU \oplus TCS/POTS$ %

% **busy$^{\mathbf{M}}$(n,r)** %

$busy^M(n,r) \rightarrow busy(n,r)$

% **disconnect$^{\mathbf{M}}$(n,r)** %

$disconnect^M(n,r) \rightarrow disconnect(n,r)$

% **member$^{\mathbf{M}}$(n,l)** %

$member^M(n,l) \rightarrow member(n,l)$

% **transit$^{\mathbf{M}}$(n,r)** %

$transit^M(n,r) \rightarrow transit^C(n,r)$

% **Apply_network$^{\mathbf{M}}$(c,r)** %

$Apply\_network^M(TCS^M(n_1,n_2),r) \rightarrow$
$Apply\_aux^M(TCS^M(n_1,n_2),r)$
$transit^M(n_2,r) \neq 0 \wedge n_1 \neq n_2 \quad \Rightarrow \quad Apply\_network^M(COMM^M(n_1,n_2),r) \rightarrow$
$Apply\_aux^M(COMM^M(n_1,transit^M(n_2,r)),r)$
$n_1 = n_2 \quad \Rightarrow \quad Apply\_network^M(COMM^M(n_1,n_2),r) \rightarrow r$
$transit^M(n_2,r) = 0 \wedge n_1 \neq n_2 \quad \Rightarrow \quad Apply\_network^M(COMM^M(n_1,n_2),r) \rightarrow$
$Apply\_aux^M(COMM^M(n_1,n_2),r)$
$Apply\_network^M(CFU^M(n_1,n_2),r) \rightarrow$
$Apply\_aux^M(CFU^M(n_1,n_2),r)$

% **Apply_aux$^{\mathbf{M}}$(c,r)** %

$Apply\_aux^M(c,[]) \rightarrow []$
$n_1 \neq 0 \wedge \neg busy^M(n_1,[t@r]) \quad \Rightarrow \quad Apply\_aux^M(CFU^M(n_1,n_2),[t@r]) \rightarrow$
$[App\_tel^M(CFU^M(n_1,n_2),t)@$
$Apply\_aux^M(CFU^M(n_1,n_2),r)]$
$n_1 = 0 \quad \Rightarrow \quad Apply\_aux^M(CFU^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$busy^M(n_1,[t@r]) \quad \Rightarrow \quad Apply\_aux^M(CFU^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_1 \neq 0 \wedge n_2 \neq 0 \wedge \neg busy^M(n_2,[t@r]) \quad \Rightarrow \quad Apply\_aux^M(TCS^M(n_1,n_2),[t@r]) \rightarrow$
$[App\_tel^M(TCS^M(n_1,n_2),t)@$
$Apply\_aux^M(TCS^M(n_1,n_2),r)]$
$n_1 = 0 \quad \Rightarrow \quad Apply\_aux^M(TCS^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_2 = 0 \quad \Rightarrow \quad Apply\_aux^M(TCS^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$busy^M(n_2,[t@r]) \quad \Rightarrow \quad Apply\_aux^M(TCS^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_1 \neq 0 \wedge n_2 \neq 0 \quad \Rightarrow \quad Apply\_aux^M(COMM^M(n_1,n_2),[t@r]) \rightarrow$
$connect^M(n_1,n_2,[t@r])$
$n_1 = 0 \quad \Rightarrow \quad Apply\_aux^M(COMM^M(n_1,n_2),[t@r]) \rightarrow [t@r]$
$n_1 \neq 0 \wedge n_2 = 0 \quad \Rightarrow \quad Apply\_aux^M(COMM^M(n_1,n_2),[t@r]) \rightarrow$
$disconnect^M(n_1,[t@r])$

% **connect$^{\mathbf{M}}$($\mathbf{n_1}$,$\mathbf{n_2}$,r)** %

$connect^M(n_1,n_2,[]) \rightarrow$
$App1\_network^M(COMM^M(n_1,n_2),[])$
$\neg busy^M(n_1,[t@r]) \wedge \neg busy^M(n_2,[t@r]) \quad \Rightarrow \quad connect^M(n_1,n_2,[t@r]) \rightarrow$
$App1\_network^M(COMM^M(n_1,n_2),[t@r])$
$busy^M(n_1,[t@r]) \quad \Rightarrow \quad connect^M(n_1,n_2,[t@r]) \rightarrow [t@r]$
$busy^M(n_2,[t@r]) \quad \Rightarrow \quad connect^M(n_1,n_2,[t@r]) \rightarrow [t@r]$

% **App1_network$^{\mathbf{M}}$(c,r)** %

$App1\_network^M(TCS^M(n_1,n_2),r) \quad \rightarrow \quad []$
$App1\_network^M(CFU^M(n_1,n_2),r) \quad \rightarrow \quad []$
$App1\_network^M(COMM^M(n_1,n_2),[]) \quad \rightarrow \quad []$
$App1\_network^M(COMM^M(n_1,n_2),[t@r]) \quad \rightarrow \quad [App\_tel^M(COMM^M(n_1,n_2),t)$
$@App1\_network^M(COMM^M(n_1,n_2),r)]$

% **App_tel$^{\mathbf{M}}$(c,t)** %

$n_3 = n_1 \Rightarrow App\_tel^M(CFU^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_2,ln,f)$
$n_3 \neq n_1 \Rightarrow App\_tel^M(CFU^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_5,ln,f)$
$n_2 = n_3 \wedge n_1 \neq n_4 \wedge \neg member^M(n_1,ln) \Rightarrow App\_tel^M(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_5,[n_1@ln],f)$
$n_2 \neq n_3 \Rightarrow App\_tel^M(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_5,ln,f)$
$n_1 = n_4 \Rightarrow App\_tel^M(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_5,ln,f)$
$member^M(n_1,ln) \Rightarrow App\_tel^M(TCS^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_5,ln,f)$
$n_2 = n_3 \Rightarrow App\_tel^M(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_1,n_5,ln,f)$
$n_1 = n_3 \Rightarrow App\_tel^M(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_2,n_5,ln,f)$
$n_2 \neq n_3 \wedge n_1 \neq n_3 \Rightarrow App\_tel^M(COMM^M(n_1,n_2),Tel^M(n_3,n_4,n_5,ln,f)) \rightarrow Tel^M(n_3,n_4,n_5,ln,f)$

% **Accepts$^{\mathbf{M}}$(lc,r)** %

$Accepts^M([],r) \rightarrow Ok\_network^M(r)$
$Ok\_network^M(r) \Rightarrow Accepts^M([c@lc],r) \rightarrow Accepts^M(lc,Apply\_aux^M(c,r))$
$\neg Ok\_network^M(r) \Rightarrow Accepts^M(lc,r) \rightarrow False$

% **Ok_network$^{\mathbf{M}}$(r)** %

$Ok\_network^M([]) \rightarrow True$
$Ok\_network^M([t@r]) \rightarrow no\_call\_bl\_himself(t) \wedge Ok\_network^M(r))$

% **no_call_bl_himself(t)** %

$n_1 = n_2 \Rightarrow no\_call\_bl\_himself(Tel^M(n_1,n_2,n_3,ln,f)) \rightarrow False$
$n_1 \neq n_2 \wedge \neg member^M(n_2,ln) \Rightarrow no\_call\_bl\_himself(Tel^M(n_1,n_2,n_3,ln,f)) \rightarrow True$
$n_1 \neq n_2 \wedge member^M(n_2,ln) \Rightarrow no\_call\_bl\_himself(Tel^M(n_1,n_2,n_3,ln,f)) \rightarrow f$

Afin d'obtenir la spécification composée pour une configuration de réseau à deux utilisateurs, il suffit d'effacer de la partie **constructors** la ligne :

$3 : \rightarrow Nat^M$

B

Spécification PVS et SPIKE de l'algorithme ABR

B.1 Spécification PVS

```
atm : Theory
  BEGIN

                                        % Déclarations globales %
    tuptype : TYPE  =  [nonneg_real,nonneg_real]

    tcrt,t2,t3,t,to,e,e1,ta,tb : VAR nonneg_real
    l,p : VAR list[tuptype]
    O,o1,oa : VAR tuptype
                                        % Fonctions communes %
    Time(t,e) : nonneg_real  =  t

    Er(t,e) : nonneg_real  =  e

    SortedT(l) :
     RECURSIVE bool  =  CASES l OF
       null : TRUE,
       cons(o1,p1) :
         (CASES p1 OF
           null : TRUE,
           cons(o2,p2) :
             IF Time(o1) ≥ Time(o2) THEN SortedT(cons(o2,p2))
              ELSE FALSE
              ENDIF
           ENDCASES)
       ENDCASES
     MEASURE length

                                        % Fonctions propres à Acr %
    Wind(l,tcrt,t2,t3) :
     RECURSIVE list[tuptype]  =  CASES l OF
       null : null,
       cons(O,p) :
```

```
        (IF Time(O) + t3 > tcrt THEN Wind(p,tcrt,t2,t3)
          ELSIF Time(O) + t2 ≤ tcrt THEN cons(O,null)
          ELSE cons(O,Wind(p,tcrt,t2,t3))
          ENDIF)
      ENDCASES
    MEASURE length(l)

  MaxEr(l) :
   RECURSIVE nonneg_real = CASES l OF
      null : 0,
      cons(O,p) : (IF MaxEr(p) ≤ Er(O) THEN Er(O) ELSE MaxEr(p) ENDIF)
      ENDCASES
    MEASURE length

  Acr(l,tcrt,t2,t3) :
       nonneg_real = IF SortedT(l) ∧ (t2 > t3) THEN MaxEr(Wind(l,tcrt,t2,t3)) ELSE 0 ENDIF
                                  % Fonctions propres à Acr1 %
  InsAt(l,t,e) :
   RECURSIVE list[tuptype] = CASES l OF
      null : cons((t,e),null),
      cons(O,p) :
         (IF Time(O) ≤ t THEN cons((t,e),cons(O,p))
           ELSE InsAt(p,t,e)
           ENDIF)
      ENDCASES
    MEASURE length(l)

  InsIn(l,t,e) :
   RECURSIVE list[tuptype] = CASES l OF
      null : cons((t,e),null),
      cons(O,p) :
         (IF Er(O) ≤ e THEN InsIn(p,Time(O),e)
           ELSE cons((t,e),cons(O,p))
           ENDIF)
      ENDCASES
    MEASURE length(l)

  ProgAt(l,tcrt) :
   RECURSIVE nonneg_real = CASES l OF
      null : 0,
      cons(O,p) : (IF Time(O) ≤ tcrt THEN Er(O) ELSE ProgAt(p,tcrt) ENDIF)
      ENDCASES
    MEASURE length(l)

  Prog(l,t2,t3) :
   RECURSIVE list[tuptype] = CASES l OF
      null : null,
      cons(O,p) :
         (IF ProgAt(Prog(p,t2,t3),(Time(O) + t3)) ≤ Er(O)
            THEN InsAt(Prog(p,t2,t3),(Time(O) + t3),Er(O))
           ELSE InsIn(Prog(p,t2,t3),(Time(O) + t2),Er(O))
           ENDIF)
      ENDCASES
    MEASURE length(l)
```

```
Acr1(l,tcrt,t2,t3) :
      nonneg_real = IF SortedT(l) ∧ (t2 > t3) THEN ProgAt(Prog(l,t2,t3),tcrt) ELSE 0 ENDIF
                                % Fonctions auxiliaires %

Timel(l) : nonneg_real = CASES l OF null : 0,cons(O,p) : Time(O) ENDCASES

Erl(l) : nonneg_real = CASES l OF null : 0,cons(O,p) : Er(O) ENDCASES

MemberC(O,l) :
 RECURSIVE bool = CASES l OF
    null : FALSE,
    cons(o1,p1) : (IF O = o1 THEN TRUE ELSE MemberC(O,p1) ENDIF)
    ENDCASES
  MEASURE length(l)

MemberT(tcrt,l) :
 RECURSIVE bool = CASES l OF
    null : FALSE,
    cons(o1,p1) : (IF tcrt = Time(o1) THEN TRUE ELSE MemberT(tcrt,p1) ENDIF)
    ENDCASES
  MEASURE length(l)

MemberE(e,l) :
 RECURSIVE bool = CASES l OF
    null : FALSE,
    cons(o1,p1) : (IF e = Er(o1) THEN TRUE ELSE MemberE(e,p1) ENDIF)
    ENDCASES
  MEASURE length(l)

SortedE(l) :
 RECURSIVE bool = CASES l OF
    null : TRUE,
    cons(o1,p1) :
       (CASES p1 OF
          null : TRUE,
          cons(o2,p2) :
             IF Er(o1) < Er(o2) THEN SortedE(cons(o2,p2))
              ELSE FALSE
              ENDIF
         ENDCASES)
    ENDCASES
  MEASURE length

ListUpTo(l,t) :
 RECURSIVE list[tuptype] = CASES l OF
    null : null,
    cons(O,p) :
       (IF Time(O) ≤ t THEN cons(O,null)
         ELSE cons(O,ListUpTo(p,t))
         ENDIF)
    ENDCASES
  MEASURE length(l)
```

TimeAt(l,tcrt) :
RECURSIVE nonneg_real = CASES l OF
null : 0,
cons(O,p) : (IF Time(O) $\leq$ tcrt THEN Time(O) ELSE TimeAt(p,tcrt) ENDIF)
ENDCASES
MEASURE length(l)

FirstAt(l,tcrt) :
RECURSIVE tuptype = CASES l OF
null : (0,0),
cons(O,p) : (IF Time(O) $\leq$ tcrt THEN O ELSE FirstAt(p,tcrt) ENDIF)
ENDCASES
MEASURE length(l)

ListAt(l,tcrt) :
RECURSIVE list[tuptype] = CASES l OF
null : null,
cons(O,p) :
(IF Time(O) $\leq$ tcrt THEN cons(O,p) ELSE ListAt(p,tcrt) ENDIF)
ENDCASES
MEASURE length(l)

% Lemmes introduits par l'utilisateur %

firstat_timeat : THEOREM Time(FirstAt(l,t)) = TimeAt(l,t)

firstat_progat : THEOREM Er(FirstAt(l,t)) = ProgAt(l,t)

sorted_sorted : THEOREM SortedT(cons(O,p)) = TRUE $\Rightarrow$ SortedT(p) = TRUE

sorted_insat1 : THEOREM SortedT(l) $\Rightarrow$ SortedT(InsAt(l,t,e))

sorted_insin2 : THEOREM SortedT(l) $\wedge$ Timel(l) $\leq t \Rightarrow$ SortedT(InsIn(l,t,e))

sorted_e_two : THEOREM SortedE(cons(O,p)) $\Rightarrow$ SortedE(p)

sorted_e_insin : THEOREM SortedE(l) $\Rightarrow$ SortedE(InsIn(l,t,e))

sorted_e_member : THEOREM SortedE(l) $\wedge$ MemberC(O,l) $\Rightarrow$ Erl(l) $\leq$ Er(O) $\vee$ null?(l)

member_t_insin : THEOREM MemberT(tcrt,InsIn(l,t,e)) $\Rightarrow$ tcrt = $t \vee$ MemberT(tcrt,l)

member_t_insat : THEOREM MemberT(tcrt,InsAt(l,t,e)) $\Rightarrow$ tcrt = $t \vee$ MemberT(tcrt,l)

member_less : THEOREM SortedT(l) $\wedge$ MemberC(O,l) $\Rightarrow$ Time(O) $\leq$ Timel(l)

member_insin_time : THEOREM
SortedT(l) $\wedge$ MemberC(o_1,l) $\wedge$ e < Er(o_1) $\wedge$ $t \geq$ Timel(l) $\Rightarrow$
Timel(InsIn(l,t,e)) $\geq$ Time(o_1)

member_firstat : THEOREM cons?(ListAt(l,tcrt)) $\Rightarrow$ MemberC(FirstAt(l,tcrt),l)

timel_insat_t : THEOREM Timel(InsAt(l,t,e)) = t

timel_insin : THEOREM SortedT(l) $\wedge$ Timel(l) $\leq$ tcrt $\Rightarrow$ Timel(InsIn(l,tcrt,e)) $\leq$ tcrt

erl_insin : THEOREM Erl(InsIn(l,t,e)) = e

erl_insat : THEOREM Erl(InsAt(l,t,e)) = e

erl_prog : THEOREM Erl(Prog(l,t_2,t_3)) = Erl(l)

time_progat_er : THEOREM Timel(l) $\leq$ tcrt $\Rightarrow$ ProgAt(l,tcrt) = Erl(l)

timeat_tcrt : THEOREM SortedT(l) $\Rightarrow$ TimeAt(l,tcrt) $\leq$ tcrt

timeat_timel : THEOREM SortedT(l) $\Rightarrow$ TimeAt(l,tcrt) $\leq$ Timel(l)

timel_timeat_max : THEOREM SortedT(l) $\wedge$ $t \geq$ Timel(l) $\Rightarrow$ TimeAt(l,t) = Timel(l)

sorted_timeat : THEOREM SortedT(l) $\wedge$ $t_2 \geq t_3 \Rightarrow$ TimeAt(l,t_2) $\geq$ TimeAt(l,t_3)

sorted_timel_timeat : THEOREM SortedT(l) $\wedge$ Timel(l) $\leq$ TimeAt(l,t) $\Rightarrow$ $t \geq$ Timel(l)

timel_prog_conj1 : THEOREM
SortedT(l) $\wedge$ Timel(l) $\leq$ tcrt $\wedge$ $t_2 > t_3 \wedge$ MemberT(Time(o_1),Prog(l,t_2,t_3)) $\Rightarrow$
(Time(o_1) $\leq$ tcrt + t_2)

time_dec : THEOREM SortedT(l) $\wedge$ $t_2 > t_3 \Rightarrow$ SortedT(Prog(l,t_2,t_3))

timel_prog : THEOREM SortedT(l) $\wedge$ $t_2 > t_3 \Rightarrow$ (Timel(Prog(l,t_2,t_3)) $\leq$ Timel(l) + t_2)

null_insat : THEOREM null?(InsAt(l,t,e)) $\Rightarrow$ null?(l)

null_insin : THEOREM null?(InsIn(l,t,e)) $\Rightarrow$ null?(l)

null_prog : THEOREM null?(Prog(l,t_2,t_3)) $\Rightarrow$ null?(l)

null_listat : THEOREM null?(ListAt(l,t)) $\Rightarrow$ ProgAt(l,t) = 0

null_listat1 : THEOREM null?(l) $\Rightarrow$ null?(ListAt(l,t))

cons_insat : THEOREM cons?(InsAt(l,t,e))

cons_listat : THEOREM cons?(ListAt(l,t)) $\Rightarrow$ cons?(l)

progat_two_timeat : THEOREM SortedT(l) $\Rightarrow$ ProgAt(l,TimeAt(l,t)) = ProgAt(l,t)

progat_timel_erl : THEOREM SortedT(l) $\Rightarrow$ ProgAt(l,Timel(l)) = Erl(l)

progat_insat : THEOREM SortedT(l) $\wedge$ $t >$ tcrt $\Rightarrow$ ProgAt(InsAt(l,t,e),tcrt) = ProgAt(l,tcrt)

progat_insat1 : THEOREM SortedT(l) $\wedge$ $t \leq$ tcrt $\Rightarrow$ ProgAt(InsAt(l,t,e),tcrt) = e

progat_insin_timeat : THEOREM
SortedT(l) $\wedge$ ProgAt(l,t) $> e \wedge$ to $\geq$ Timel(l) $\wedge$ $t <$ to $\Rightarrow$ Timel(InsIn(l,to,e)) $> t$

progat_insin : THEOREM

SortedT(l) ∧ Timel(InsIn(l,t,e)) > tcrt ⇒ ProgAt(InsIn(l,t,e),tcrt) = ProgAt(l,tcrt)

listat_insin_tcrt : THEOREM
SortedT(l) ∧ Timel(InsIn(l,t,e)) > tcrt ⇒ ListAt(InsIn(l,t,e),tcrt) = ListAt(l,tcrt)

progat_insin_t : THEOREM
SortedT(l) ∧ Timel(InsIn(l,t,e)) ≤ tcrt ⇒ ProgAt(InsIn(l,t,e),tcrt) = e

listupto1_erl : THEOREM
SortedT(l) ∧ t ≤ to ⇒
Erl(ListUpTo(l,t)) = Erl(ListUpTo(l,to))∨
null?(ListUpTo(l,to)) ∨ null?(ListUpTo(l,t))

null_listupto : THEOREM t ≤ to ∧ SortedT(l) ∧ null?(ListUpTo(l,t)) ⇒ null?(ListUpTo(l,to))

listupto_t_insat : THEOREM SortedT(l) ⇒ ListUpTo(InsAt(l,t,e),t) = cons((t,e),null)

listupto_insin_tcrt : THEOREM
SortedT(l) ∧ tcrt < Timel(InsIn(l,t,e)) ⇒
ListUpTo(InsIn(l,t,e),tcrt) = InsIn(ListUpTo(l,tcrt),t,e)

sorted_e_listupto : THEOREM
SortedT(l) ∧ t ≤ to ∧ SortedE(ListUpTo(l,t)) ⇒ SortedE(ListUpTo(l,to))

timel_listupto : THEOREM Timel(ListUpTo(l,t)) = Timel(l)

sorted_listupto : THEOREM SortedT(l) ⇒ SortedT(ListUpTo(l,t))

progat_listupto : THEOREM SortedT(l) ∧ ta ≥ tb ⇒ ProgAt(ListUpTo(l,tb),ta) = ProgAt(l,ta)

leftmax : THEOREM
SortedT(l) ∧ t_2 > t_3 ∧ tcrt ≥ Timel(l) + t_2 ⇒ ProgAt(Prog(l,t_2,t_3),tcrt) = Erl(l)

leftmax_max : THEOREM
SortedT(l) ∧ t_2 > t_3 ∧ tcrt ≥ Timel(l) + t_2 ⇒ MaxEr(Wind(l,tcrt,t_2,t_3)) = Erl(l)

right_prog : THEOREM
SortedT(cons(o_1,p)) ∧ Time(o_1) + t_3 > tcrt ∧ t_2 > t_3 ⇒
ProgAt(Prog(cons(o_1,p),t_2,t_3),tcrt) = ProgAt(Prog(p,t_2,t_3),tcrt)

right_wind : THEOREM
SortedT(cons(o_1,p)) ∧ Time(o_1) + t_3 > tcrt ∧ t_2 > t_3 ⇒
MaxEr(Wind(cons(o_1,p),tcrt,t_2,t_3)) = MaxEr(Wind(p,tcrt,t_2,t_3))

time_listat : THEOREM SortedT(l) ∧ t ≥ Timel(l) ⇒ ListAt(l,t) = l

listat_listupto : THEOREM
SortedT(l) ∧ ta ≤ tb ∧ cons?(ListAt(l,ta)) ⇒ cons?(ListAt(ListUpTo(l,ta),tb))

sorted_cons_listat : THEOREM SortedT(l) ∧ cons?(ListAt(l,t_3)) ∧ t_2 ≥ t_3 ⇒ cons?(ListAt(l,t_2))

progat_member_time : THEOREM
cons?(ListAt(l,t))∧
ProgAt(l,t) = 0∧

SortedT(l) ∧ SortedE(l) ∧ MemberC(O,l) ∧ Time(O) > t ⇒
Er(O) = 0

sorted_cons_two : THEOREM
SortedT(cons(o_1,p)) ∧ t_2 > t_3 ⇒ cons?(ListAt(Prog(cons(o_1,p),t_2,t_3),Time(o_1) + t_3))

sorted_cons_two_nil : THEOREM
SortedT(l) ∧ t_2 > t_3 ⇒ cons?(ListAt(Prog(l,t_2,t_3),Timel(l) + t_3)) ∨ null?(l)

sorted_e_progat_prog_two : THEOREM
SortedT(cons(o_1,p)) ∧ t_2 > t_3 ⇒ SortedE(ListUpTo(Prog(cons(o_1,p),t_2,t_3),Time(o_1) + t_3))

sorted_e_progat_prog_two_nil : THEOREM
SortedT(l) ∧ t_2 > t_3 ⇒ SortedE(ListUpTo(Prog(l,t_2,t_3),Timel(l) + t_3)) ∨ null?(l)

sorted_e_member_two : THEOREM
SortedT(l) ∧ SortedE(l) ∧ $t_2 \geq t_3$ ⇒
ProgAt(l,t_2) ≤ ProgAt(l,t_3) ∨ ProgAt(l,t_3) = 0

new_listat_prog : THEOREM
SortedT(l) ∧ SortedE(l) ∧ cons?(ListAt(l,t)) ∧ ProgAt(l,t) ≤ e ∧ $t \leq$ tcrt ⇒
ProgAt(l,tcrt) ≤ e

progat_prog_two : THEOREM
SortedT(cons(o_1,p))∧
t_2 > t_3∧
ProgAt(Prog(p,t_2,t_3),Time(o_1) + t_3) ≤ Er(o_1)∧
tcrt ≥ Time(o_1) + t_3 ⇒
ProgAt(Prog(p,t_2,t_3),tcrt) ≤ Er(o_1)

null_wind1 : THEOREM t_2 > t_3 ∧ Timel(l) + $t_3 \leq$ tcrt ∧ null?(Wind(l,tcrt,t_2,t_3)) ⇒ null?(l)

null_wind2 : THEOREM
t_2 > t_3 ∧ Time(o_1) + $t_3 \leq$ tcrt ∧ null?(Wind(l,tcrt,t_2,t_3)) ∧ SortedT(cons(o_1,l)) ⇒
null?(l)

member_t_timel : THEOREM MemberT(Timel(InsIn(l,t,e)),l) ∨ Timel(InsIn(l,t,e)) = t

timeat_greater : THEOREM SortedT(l) ∧ MemberT(t,l) ∧ t > TimeAt(l,tcrt) ⇒ t > tcrt

timel_insin1 : THEOREM e < Erl(l) ⇒ Timel(InsIn(l,t,e)) = t

null_listupto1 : THEOREM null?(ListUpTo(l,t)) ⇒ null?(l)

sorted_e_cons : THEOREM SortedE(cons(O,l)) ⇒ Er(O) < Erl(l) ∨ null?(l)

erl_cons : THEOREM Erl(ListUpTo(l,t)) = Erl(l)

no_time : THEOREM ProgAt(l,t) = e ∧ $t \geq$ ta ∧ ta ≥ TimeAt(l,t) ⇒ ProgAt(l,ta) = e

prev_time : THEOREM
SortedT(l)∧
Timel(InsIn(l,t_2,e)) > t_3∧
$t_2 \geq$ Timel(l)∧

$t_2 > \text{tcrt} \wedge \text{tcrt} \geq t_3 \wedge t_3 \geq \text{TimeAt}(l,\text{tcrt}) \Rightarrow$
$\text{Timel}(\text{InsIn}(l,t_2,e)) > \text{tcrt} \vee \text{null?}(l)$

monoton_e : THEOREM
$\text{SortedT}(l) \wedge$
$\text{SortedE}(\text{ListUpTo}(l,t_3)) \wedge$
$t_2 > \text{tcrt} \wedge$
$\text{tcrt} \geq t_3 \wedge$
$t_3 < \text{TimeAt}(l,\text{tcrt}) \wedge$
$e \geq \text{ProgAt}(l,\text{tcrt}) \wedge t_2 \geq \text{Timel}(l) \Rightarrow$
$\text{Timel}(\text{InsIn}(l,t_2,e)) \leq \text{TimeAt}(l,\text{tcrt}) \vee \text{null?}(l)$

monoton_e1 : THEOREM
$\text{SortedT}(l) \wedge$
$\text{SortedE}(\text{ListUpTo}(l,t_3)) \wedge$
$t_2 > \text{tcrt} \wedge$
$\text{tcrt} \geq t_3 \wedge$
$t_3 < \text{TimeAt}(l,\text{tcrt}) \wedge$
$e < \text{ProgAt}(l,\text{tcrt}) \wedge t_2 \geq \text{Timel}(l) \Rightarrow$
$\text{Timel}(\text{InsIn}(l,t_2,e)) > \text{TimeAt}(l,\text{tcrt}) \vee \text{null?}(l)$

main_conj : THEOREM $\text{SortedT}(l) \wedge t_2 > t_3 \Rightarrow \text{ProgAt}(\text{Prog}(l,t_2,t_3),\text{tcrt}) = \text{MaxEr}(\text{Wind}(l,\text{tcrt},t_2,t_3))$

final : THEOREM $\text{Acr}(l,\text{tcrt},t_2,t_3) = \text{Acr1}(l,\text{tcrt},t_2,t_3)$

END atm

B.2 Spécification SPIKE

```
Specification: AlgoConformNat

use : nats; % utilisation de LA

Sorts: OBJ PLAN;

Constructors:


_/_     : nat nat -> OBJ;

Nil     :           -> PLAN;
cons__  : OBJ  PLAN -> PLAN;

Defined Functions:

eqlv__  : PLAN PLAN  -> bool;
eqv__   : OBJ  OBJ   -> bool;
eq__    : nat  nat   -> bool;
Time_   : OBJ        -> nat;
Er_     : OBJ        -> nat;

Timel_           : PLAN             -> nat;
Erl_             : PLAN             -> nat;
MemberC__        : OBJ PLAN         -> bool;
MemberT__        : nat PLAN         -> bool;
MemberE__        : nat PLAN         -> bool;
SortedT_         : PLAN             -> bool;
SortedE_         : PLAN             -> bool;
```

```
ListUpTo__       : PLAN nat          -> PLAN;
Wind____         : PLAN nat nat nat  -> PLAN;
MaxEr_           : PLAN              -> nat;
Acr____          : PLAN nat nat nat  -> nat;

InsAt___         : PLAN nat nat      -> PLAN;
InsIn___         : PLAN nat nat      -> PLAN;
ProgAt__         : PLAN nat          -> nat;
TimeAt__         : PLAN nat          -> nat;
FirstAt__        : PLAN nat          -> OBJ;
ListAt__         : PLAN nat          -> PLAN;
Prog___          : PLAN nat nat      -> PLAN;
Acr1____         : PLAN nat nat nat  -> nat;

Axioms:

Time(t/e) = t;
Er(t/e)   = e;

Timel(Nil) = 0;
Timel(cons(o,p)) = Time(o);

Erl(Nil) = 0;
Erl(cons(o,p)) = Er(o);

MemberC(o, Nil) = false;
eqv(o1, o2) = True => MemberC(o1,cons(o2,p)) = True;
eqv(o1, o2) = False => MemberC(o1,cons(o2,p)) = MemberC(o1, p);

MemberT(t, Nil) = false;
eq(t1, t2) = True => MemberT(t1,cons((t2/e),p)) = True;
eq(t1, t2) = False => MemberT(t1,cons((t2/e),p)) = MemberT(t1, p);

MemberE(e, Nil) = false;
eq(e1, e2) = True => MemberE(e1,cons((t/e2),p)) = True;
eq(e1, e2) = False => MemberE(e1,cons((t/e2),p)) = MemberE(e1, p);

SortedT(Nil) = true;
SortedT(cons(o,Nil)) = true;
Time(o2) <= Time(o1) = true
               => SortedT(cons(o1,cons(o2,p))) = SortedT(cons(o2,p));
Time(o2) <= Time(o1) = false
               => SortedT(cons(o1,cons(o2,p))) = false;

SortedE(Nil) = true;
SortedE(cons(o,Nil)) = true;
Er(o2) <= Er(o1) = false
               => SortedE(cons(o1,cons(o2,p))) = SortedE(cons(o2,p));
Er(o2) <= Er(o1) = true
               => SortedE(cons(o1,cons(o2,p))) = false;

ListUpTo(Nil, t) = Nil;
Time(o) <= t = True => ListUpTo(cons(o,p),t) = cons(o, Nil);
Time(o) <= t = False => ListUpTo(cons(o,p),t) = cons(o, ListUpTo(p,t));


Wind(Nil, t, t1, t2) = Nil;
(Time(o)+t2) <= t = false
       => Wind(cons(o,p), t, t1, t2) = Wind(p, t, t1, t2);
(Time(o)+t1) <= t = true
       => Wind(cons(o,p), t, t1, t2) = cons(o,Nil);
(Time(o)+t2) <= t = true,
(Time(o)+t1) <= t = false
       => Wind(cons(o,p), t, t1, t2) = cons(o,Wind(p, t, t1, t2));

MaxEr(Nil) = 0;
MaxEr(p) <= Er(o) = true  =>  MaxEr(cons(o,p)) = Er(o);
MaxEr(p) <= Er(o) = false =>  MaxEr(cons(o,p)) = MaxEr(p);
```

```
SortedT(p) = true, t2 <= t3 = false
                 => Acr(p,t,t2,t3) = MaxEr(Wind(p, t, t2, t3));
SortedT(p) = false => Acr(p,t,t2,t3) = 0;
t2 <= t3  = true  => Acr(p,t,t2,t3) = 0;

InsAt(Nil, t, e) = cons((t/e), Nil);
Time(o) <= t = false  => InsAt(cons(o,pg), t, e)
                         =
                         InsAt(pg, t, e);
Time(o) <= t = true => InsAt(cons(o,pg), t, e)
                         =
                         cons((t/e),cons(o, pg));

InsIn(Nil, t, e) = cons((t/e), Nil);
Er(o) <= e = true  => InsIn(cons(o,pg), t, e)
                      =
                      InsIn(pg, Time(o), e);
Er(o) <= e = false => InsIn(cons(o,pg), t, e)
                      =
                      cons((t/e),cons(o, pg));

ProgAt(Nil,t) = 0;
Time(o) <= t = true  => ProgAt(cons(o, pg), t) = Er(o);
Time(o) <= t = false => ProgAt(cons(o, pg), t) = ProgAt(pg, t);

TimeAt(Nil,t) = 0;
Time(o) <= t = true  => TimeAt(cons(o, pg), t) = Time(o);
Time(o) <= t = false => TimeAt(cons(o, pg), t) = TimeAt(pg, t);

FirstAt(Nil,t) = (0/0);
Time(o) <= t = true  => FirstAt(cons(o, pg), t) = o;
Time(o) <= t = false => FirstAt(cons(o, pg), t) = FirstAt(pg, t);

ListAt(Nil,t) = Nil;
Time(o) <= t = true  => ListAt(cons(o, pg), t) = cons(o, pg);
Time(o) <= t = false => ListAt(cons(o, pg), t) = ListAt(pg, t);

Prog(Nil, t2, t3) = Nil;
ProgAt(Prog(p, t2, t3) , Time(o)+t3) <= Er(o) = true
                => Prog(cons(o,p), t2, t3)
                   =
                   InsAt(Prog(p, t2, t3), Time(o)+t3, Er(o));

ProgAt(Prog(p, t2, t3), Time(o)+t3) <= Er(o) = false
                => Prog(cons(o,p), t2, t3)
                   =
                   InsIn(Prog(p, t2, t3), Time(o)+t2, Er(o));

SortedT(p) = true,
t2 <= t3  = false => Acr1(p,t,t2,t3)
                     =
                     ProgAt(Prog(p, t2, t3), t);
SortedT(p) = false => Acr1(p,t,t2,t3) = 0;
t2 <= t3  = true  => Acr1(p,t,t2,t3) = 0;

%eqlv
eqlv(Nil, Nil) = True;
eqlv(cons(t,l), Nil) = False;
eqlv(Nil, cons(t,l)) = False;
eqv(t1,t2) = True => eqlv(cons(t1,l1),cons(t2,l2)) = eqlv(l1,l2);
eqv(t1,t2) = False => eqlv(cons(t1,l1),cons(t2,l2)) = False;

%eqv
eq(t1,t2) = True, eq(e1,e2) = True => eqv((t1/e1),(t2/e2)) = True;
eq(t1,t2) = False => eqv((t1/e1),(t2/e2)) = False;
eq(e1,e2) = False => eqv((t1/e1),(t2/e2)) = False;
```

```
%eq
eq(0,0) = True;
eq(0,s(x)) = False;
eq(s(x),0) = False;
eq(s(x),s(y)) = eq(x,y);

less:
true false 0 s / Nil cons eq eqv eqlv + <= Time Er Timel Erl MemberC
MemberT MemberE SortedT SortedE ListUpTo Wind MaxEr Acr InsAt InsIn
ProgAt TimeAt FirstAt ListAt Prog Acr1;


Properties:
system_is_sufficiently_complete ;
system_is_ground_convergent ;

Strategy:

delete_set=try(tautology, negative_clash, subsumption);

decomposition_set=try(positive_decomposition,negative_decomposition,positive_clash);

eliminate_set=try(eliminate_redundant_literal,eliminate_trivial_literal);

simplify=try(delete_set, eliminate_set, decomposition_set, auto_simplification) ;

rec_strategy_rewriting_level0 = repeat_plus(repeat_plus(delete_set),
                                            conditional_rewriting(normalize, R, *));

rec_strategy_induction = try (print_goals,
                              tautology,
                              total_case_rewriting (delete, R, *),
                              conditional_rewriting (normalize, R, *));

main = repeat (try ((simplify, print_goals (t)),
                        conditional_rewriting (normalize, R&L, *),
                        equational_rewriting (*),
                        contextual_rewriting (rec_strategy_rewriting_level0, r|e|h|l, *),
                        partial_case_rewriting (R, *),
                        total_case_rewriting (delete_set, R, *),
                        print_goals,
                        induction
                    )) ;


start_with: main

Lemmas:

% lemmes arithmétiques à être utilisés comme des règles de réécriture

0+x = x;
s(x)+y = s(x+y);

(0    <= x   ) = true;
(s(x) <= 0   ) = false;
(s(x) <= s(y)) = x <= y;

% lemmes définis par l'utilisateur

%  firstat_timeat: %ok
 Time(FirstAt(l,t)) = TimeAt(l,t);
%  firstat_progat: %ok
 Er(FirstAt(l,t)) = ProgAt(l,t);
%  sorted_sorted: %ok
 SortedT(cons(o, p)) = TRUE => SortedT(p) = TRUE;
%  sorted_insat1: %ok
 SortedT(l) = True => SortedT(InsAt(l, t, e)) = True;
```

```
%  sorted_insin2: %ok
 SortedT(l) = True, Timel(l) <= t = True => SortedT(InsIn(l, t, e)) = True;
%  sorted_e_two: %ok
 SortedE(cons(o,p)) = True => SortedE(p) = True;
%  sorted_e_insin: %ok
 SortedE(l) = True => SortedE(InsIn(l,t,e)) = True;
%  sorted_e_member:
 SortedE(l) = True, MemberC(o,l)=True, Erl(l) <= Er(o) = False => l = Nil;
%  member_t_insin: %ok -- preuve longue
 MemberT(tcrt,InsIn(l,t,e)) =True, MemberT(tcrt,l) = False => t= tcrt ;
%  member_t_insat:
 MemberT(tcrt,InsAt(l,t,e)) = True,  MemberT(tcrt,l) = False => tcrt = t ;
%  member_less:
 SortedT(l)= True, MemberC(o,l)=True => Time(o) <= Timel(l)=True;
%  member_insin_time:
 SortedT(l)=True, MemberC(o1,l)=True, s(e) <= Er(o1)=True, Timel(l)<=t= True =>
   Time(o1) <=Timel(InsIn(l,t,e))=True;
%  member_firstat: %ok
 eqlv(ListAt(l,tcrt),Nil)=False => MemberC(FirstAt(l,tcrt),l)=True;
%  timel_insat_t: %ok
 Timel(InsAt(l,t,e)) = t;
%  timel_insin:
 SortedT(l) = True, Timel(l) <= tcrt = True => Timel(InsIn(l,tcrt,e)) <= tcrt = True;
%  erl_insin: %ok
 Erl(InsIn(l,t,e)) = e;
%  erl_insat: %ok
 Erl(InsAt(l,t,e)) = e;
%  erl_prog: %ok
 Erl(Prog(l,t2,t3)) = Erl(l);
%  time_progat_er: %ok
 Timel(l) <= tcrt =True => ProgAt(l,tcrt) = Erl(l);
%  timeat_tcrt: %ok
 SortedT(l) = True => TimeAt(l,tcrt) <= tcrt = True;
%  timeat_timel:
 SortedT(l)=True => TimeAt(l,tcrt) <= Timel(l) =True;
%  timel_timeat_max: %ok
 SortedT(l) = True,  Timel(l)<=t = True => TimeAt(l,t) = Timel(l);
%  sorted_timeat:
 SortedT(l) = True, t3 <= t2 = True =>  TimeAt(l,t3) <= TimeAt(l,t2) = True;
%  sorted_timel_timeat:
 SortedT(l) = True,  Timel(l) <= TimeAt(l,t) = True => Timel(l) <= t = True;
%  timel_prog_conj1:
 SortedT(l) = True, Timel(l) <=tcrt = True,   s(t3)  <= t2 = True,
   MemberT(Time(o1), Prog(l,t2,t3)) = True => (Time(o1) <= (tcrt + t2)) = True;
%  time_dec:
 SortedT(l) = True,  s(t3) <= t2 = True => SortedT(Prog(l,t2,t3))= True;
%  timel_prog:
 SortedT(l)=True, t3 <= t2=True => (Timel(Prog(l,t2,t3)) <= (Timel(l) + t2))=True;
%  null_insat: %ok
 InsAt(l,t,e)=Nil => Nil=l;
%  null_insin: %ok
 InsIn(l,t,e) = Nil => Nil = l;
%  null_prog:
 Prog(l,t2,t3) = Nil => l = Nil;
%  null_listat: %ok
 ListAt(l,t)=Nil => ProgAt(l,t) = 0 ;
%  null_listat1: %ok
 l = Nil => ListAt(l,t) = Nil;
%  cons_insat: %ok
 InsAt(l,t,e) = Nil =>;
%  cons_listat: %ok (version contraposée de null_listat1)
 l =  Nil => ListAt(l,t) = Nil;
%  progat_two_timeat: %ok
 SortedT(l) = True => ProgAt(l, TimeAt(l,t)) = ProgAt(l,t);
%  progat_timel_erl: %ok
 SortedT(l)= True => ProgAt(l,Timel(l)) = Erl(l);
%  progat_insat: %ok
 SortedT(l)=True, t<= tcrt =false => ProgAt(InsAt(l,t,e),tcrt) = ProgAt(l,tcrt);
```

```
%  progat_insat1: %ok
 SortedT(l)=True, t<= tcrt=True => ProgAt(InsAt(l,t,e),tcrt) = e;
%  progat_insin_timeat:
 SortedT(l)=True, ProgAt(l,t) <= e = false,  Timel(l)<=to = true, to<= t= false =>
   Timel(InsIn(l,to,e)) <= t = false;
%  progat_insin: %ok
 SortedT(l)=true, Timel(InsIn(l,t,e)) <= tcrt =false => ProgAt(InsIn(l,t,e),tcrt) = ProgAt(l,tcrt);
%  listat_insin_tcrt: %ok
 SortedT(l)=true, Timel(InsIn(l,t,e)) <= tcrt = false => ListAt(InsIn(l,t,e),tcrt) = ListAt(l,tcrt);
%  progat_insin_t: %ok
 SortedT(l)=True, Timel(InsIn(l,t,e)) <= tcrt=true => ProgAt(InsIn(l,t,e),tcrt) = e;
%  listupto1_erl: %ok
 SortedT(l)=True, t <= to =true=> Erl(ListUpTo(l,t)) = Erl(ListUpTo(l,to)),
   ListUpTo(l,to)=Nil, ListUpTo(l,t)=Nil;
%  null_listupto: %ok
 t<=to=True, SortedT(l)=True, ListUpTo(l,t)=Nil => ListUpTo(l,to)=Nil;
%  listupto_t_insat: %ok
 SortedT(l)=True => ListUpTo(InsAt(l,t,e),t) = cons((t/e),Nil);
%  listupto_insin_tcrt:
 SortedT(l)=True,  Timel(InsIn(l,t,e)) <=tcrt = False =>
   ListUpTo(InsIn(l,t,e),tcrt) = InsIn(ListUpTo(l,tcrt),t,e);
%  sorted_e_listupto:
 SortedT(l)=True, t <= to=True, SortedE(ListUpTo(l,t))=True => SortedE(ListUpTo(l,to))=True;
%  timel_listupto: %ok
 Timel(ListUpTo(l,t)) = Timel(l);
%  sorted_listupto: %ok
 SortedT(l)=True => SortedT(ListUpTo(l,t))=True;
%  progat_listupto: %ok
 SortedT(l)=True, tb <=ta=True => ProgAt(ListUpTo(l,tb),ta) = ProgAt(l,ta);
%  leftmax: %ok
 SortedT(l)=True, t2 <= t3=false, Timel(l) + t2 <= tcrt = true => ProgAt(Prog(l,t2,t3),tcrt) = Erl(l);
%  leftmax_max: %ok
 SortedT(l)=True, t2<=t3 = false, Timel(l) + t2 <= tcrt = true => MaxEr(Wind(l, tcrt, t2, t3)) = Erl(l);
%  right_prog:
 SortedT(cons(o1,p))=True, Time(o1) + t3 <= tcrt= false, t2 <= t3 = false =>
   ProgAt(Prog(cons(o1,p),t2,t3),tcrt) = ProgAt(Prog(p,t2,t3),tcrt);
%  right_wind: %ok
 SortedT(cons(o1,p))=True, Time(o1) + t3 <= tcrt=false, t2 <= t3 =false =>
   MaxEr(Wind(cons(o1,p),tcrt,t2,t3)) = MaxEr(Wind(p,tcrt,t2,t3));
%  time_listat: %ok
 SortedT(l)=True, Timel(l) <= t = True => ListAt(l,t) = l;
%  listat_listupto: %ok
 SortedT(l)=True, ta<=tb=True, ListAt(ListUpTo(l,ta),tb)= Nil => ListAt(l,ta) = Nil;
%  sorted_cons_listat: %ok
 SortedT(l)=True, ListAt(l,t2)=Nil, t3 <= t2 = True => ListAt(l,t3)=Nil;
%  progat_member_time:
 ProgAt(l,t) = 0, SortedT(l) = True, SortedE(l)= True, MemberC(o,l) = True,
   Time(o) <= t = false => Er(o) = 0, ListAt(l,t) = Nil;
%  sorted_cons_two:
 SortedT(cons(o1,p))=True, ListAt(Prog(cons(o1,p),t2,t3),Time(o1) + t3) = Nil => t2<=t3=true;
%  sorted_cons_two_nil:
 SortedT(l) = True,  t2<=t3=false, ListAt(Prog(l,t2,t3),Timel(l) + t3) = Nil => l =Nil;
%  sorted_e_progat_prog_two:
 SortedT(cons(o1,p))=True, t2 <= t3 =false =>
   SortedE(ListUpTo(Prog(cons(o1,p),t2,t3),Time(o1) + t3))=True;
%  rate_inc:
 SortedT(l)=True, t2 <= t3=false => SortedE(ListUpTo(Prog(l,t2,t3),Timel(l) + t3))=true;
%  sorted_e_member_two:
 SortedT(l)=True, SortedE(l)=True, t3<= t2=true => ProgAt(l,t2) <= ProgAt(l,t3)=true, ProgAt(l,t3) = 0;
%  new_listat_prog:
 SortedT(l) = True, SortedE(l)=True, ProgAt(l,t) <= e=True, t <= tcrt = True =>
   ListAt(l,t)= Nil, ProgAt(l,tcrt)<= e = True;
%  progat_prog_two:
 SortedT(cons(o1,p)) = True, t2 <= t3=false, ProgAt(Prog(p,t2,t3),Time(o1) + t3) <= Er(o1) =true,
   Time(o1) + t3 <=tcrt=True=> ProgAt(Prog(p,t2,t3),tcrt) <= Er(o1) = True;
%  null_wind1: %ok
 t2<=t3=false, Timel(l) + t3 <= tcrt = true, Wind(l,tcrt,t2,t3)=Nil => l=Nil;
%  null_wind2: %ok
```

```
 t2<=t3=false, Time(o1) + t3 <=tcrt=true, Wind(l,tcrt,t2,t3)=Nil,
%  member_t_timel:
 MemberT(Timel(InsIn(l,t,e)),l)=True, Timel(InsIn(l,t,e)) = t;
%  timeat_greater:
 SortedT(l)=True, MemberT(t,l)=True, t<= TimeAt(l,tcrt)=false => t<= tcrt=False;
%  timel_insin1: %ok
 Erl(l)<= e = false => Timel(InsIn(l,t,e)) = t;
%  null_listupto1: %ok
 ListUpTo(l,t)=Nil => l=Nil;
%  sorted_e_cons: %ok
 SortedE(cons(o,l))=true => Erl(l) <= Er(o)=false, l=Nil;
%  erl_cons: %ok
 Erl(ListUpTo(l,t)) = Erl(l);
%  no_time:  %ok
 ProgAt(l,t) = e, ta <= t = true, TimeAt(l,t)<= ta =true => ProgAt(l,ta) = e;
%  prev_time:
 SortedT(l)=True, Timel(InsIn(l,t2,e)) <= t3=false, Timel(l)<=t2=true, t2 <= tcrt=false,
   t3<=tcrt=true,  TimeAt(l,tcrt)<=t3=true  => Timel(InsIn(l,t2,e)) <= tcrt=false, l=Nil;
%  monoton_e:
 SortedT(l)=True, SortedE(ListUpTo(l,t3))=True, t2 <= tcrt=false, t3<=tcrt=true,  TimeAt(l,tcrt)<=t3=false,
   ProgAt(l,tcrt)<=e = true, Timel(l) <=t2 = true => Timel(InsIn(l,t2,e)) <= TimeAt(l,tcrt)=true, l = Nil;
%  monoton_e1:
 SortedT(l)=True, SortedE(ListUpTo(l,t3))=True, t2 <= tcrt=false, t3<=tcrt =true, TimeAt(l,tcrt)<=t3=false,
   ProgAt(l,tcrt)<=e=false, Timel(l)<=t2=true => Timel(InsIn(l,t2,e)) <= TimeAt(l,tcrt) = false, l = Nil;
%  main_conj:
 SortedT(l)=True, t2 <= t3=false => ProgAt(Prog(l,t2,t3),tcrt) = MaxEr(Wind(l,tcrt,t2,t3));

Goals:

%  final: %ok
 Acr(l,tcrt,t2,t3) = Acr1(l,tcrt,t2,t3);
```

Bibliographie

[Abrial, 1996] J.-R. Abrial. *The B-Book*. Cambridge University Press, 1996. ISBN 0-521-49619-5.

[Aggoun et Combes, 1997] I. Aggoun et P. Combes. Observers in the SCE and the SEE to detect and resolve service interactions. Dans *Feature Interactions in Telecommunication Systems*, pages 192–212. IOS Press, 1997.

[Armando et Ranise, 1998] A. Armando et S. Ranise. Constraint contextual rewriting. Dans *FTP'98*, 1998.

[Armando et Ranise, 2000] A. Armando et S. Ranise. Termination of constraint contextual rewriting. Dans *FROCOS'2000*, 2000.

[Arnold, 1990] A. Arnold. MEC : A system for constructing and analysing transition systems. Dans *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, rédacteur J. Sifakis, numéro 407 dans Lecture Notes in Computer Science, pages 117–132. Springer Verlag, 1990.

[Aubin, 1979] R. Aubin. Mechanizing structural induction. *Theoretical Computer Science*, 9 :329–362, 1979.

[Avenhaus et Madlener, 1997] J. Avenhaus et K. Madlener. Theorem proving in hierarchical clausal specifications. *Advances in Algorithms, Languages, and Complexity*, pages 1–51, 1997.

[Baader et Nipkow, 1998] F. Baader et T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Bachmair et Ganzinger, 1994] L. Bachmair et H. Ganzinger. Buchberger's algorithm : A constraint-based completion procedure. Dans *Constraints in Computational Logic, First International Conference CCL'94*, volume 845 de *Lecture Notes in Computer Science*, pages 285–301. Springer-Verlag, 1994.

[Bachmair, 1988] L. Bachmair. Proof by consistency in equational theories. Dans *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pages 228–233, 1988.

[Barras *et al.*, 1997] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi et B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Rapport Technique 0203, INRIA, Août 1997.

[Becker et Weispfenning, 1993] T. Becker et V. Weispfenning. *Gröbner Bases : A Computational Approach to Commutative Algebras*, volume 141 de *Graduate Texts in Mathematics*. Springer-Verlag, 1993.

[Becker, 1996] K. Becker. How to Prove Ground Confluence. SEKI-report SR-96-02, Universität Kaiserslautern, 1996.

[Bengtsson *et al.*, 1996] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson et W. Yi. UPPAAL : a tool suite for the automatic verification of real-time systems. Dans *Hybrid Systems*

III, rédacteurs R. Alur, T. A. Henzinger et E.D. Sontag, numéro 1066 dans Lecture Notes in Computer Science, pages 232–243, 1996.

[Berger *et al.*, 1995] A. Berger, F. Bonomi et K. Fendick. Proposed TM baseline text on an ABR conformance definition. Rapport Technique 95-0212R1, ATM Forum Traffic Management Group, 1995.

[Bernot *et al.*, 1994] G. Bernot, M. Bidoit et T. Knapik. Behavioural approaches to algebraic specifications : A comparative study. *Acta Informatica*, 31(7) :651–671, 1994.

[Berregeb *et al.*, 1998] N. Berregeb, A. Bouhoula et M. Rusinowitch. Observational proofs with critical contexts. Dans *Fundamental Approaches to Software Engineering (FASE'98)*, volume 1382 de *Lecture Notes in Computer Science*, pages 38–53. Springer Verlag, March–April 1998.

[Bevers et Lewi, 1990] E. Bevers et J. Lewi. Proof by consistency in conditional equational theories. Dans *Conditional and Typed Rewriting Systems, 2nd International Workshop*, volume 516 de *Lecture Notes in Computer Science*, pages 194–205, 1990.

[Birkhoff, 1935] G. Birkhoff. On the structure of abstract algebras. Dans *Proc. Cambridge Philos. Soc.*, volume 31, pages 417–429, 1935.

[Bjørner et others, 1995] N. Bjørner et al. *STeP. The Stanford Temporal Prover*, Novembre 1995. Version 1.0.

[Bjørner, 1998] N. Bjørner. Private communication, 1998.

[Bjørner, 1998] N. Bjørner. *Integrating decision procedures for temporal verification.* Thèse de doctorat, Stanford University, Novembre 1998.

[Bledsoe et Shostak, 1979] W.W. Bledsoe et R. Shostak. A prover for general inequalities. Dans *Proceedings of the 6th IJCAI*, pages 66–69, 1979.

[Bledsoe, 1975] W. Bledsoe. A new method for proving certain Presburger formulas. Dans *Proceedings of the 4th IJCAI*, pages 15–21, 1975.

[Blom *et al.*, 1994] J. Blom, B. Jonsson et L. Kempe. Using temporal logic for modular specification of telephone services. Dans Bouma et Velthuijsen [1994], pages 197–216.

[Bündgen *et al.*, 1996] R. Bündgen, M. Göbel et W. Küchlin. Strategy compliant multi-threaded term completion. *Journal of Symbolic Computation*, 5 :1–30, 1996.

[Bündgen et Eckhardt, 1992] R. Bündgen et H. Eckhardt. A fast algorithm for ground normal form analysis. Dans *Proc of the 3rd International Conference on Algebraic and Logic Programming*, rédacteurs H. Kirchner et G. Levi, volume 632 de *Lecture Notes in Computer Science*, pages 291–305. Springer Verlag, 1992.

[Bonacina et Hsiang, 1994] M.-P. Bonacina et J. Hsiang. Parallelization of deduction strategies : an analytical study. *Journal of Automated Reasoning*, 13 :1–33, 1994.

[Boström et Engstedt, 1995] M. Boström et M. Engstedt. Feature interaction detection and resolution in the Delphi framework. Dans Cheng et Ohta [1995], pages 157–172.

[Bouhoula *et al.*, 1995] A. Bouhoula, E. Kounalis et M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5) :631–668, 1995.

[Bouhoula et Rusinowitch, 1993] A. Bouhoula et M. Rusinowitch. Automatic case analysis in proof by induction. Dans *13th IJCAI Conf.*, volume 1, pages 88–94, 1993.

[Bouhoula et Rusinowitch, 1995a] A. Bouhoula et M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2) :189–235, 1995.

[Bouhoula et Rusinowitch, 1995b] A. Bouhoula et M. Rusinowitch. *SPIKE-User Manual*, Décembre 1995.

[Bouhoula, 1994] A. Bouhoula. *Preuves automatiques par récurrence dans les théories conditionnelles.* Thèse de doctorat, Université Nancy I, Mars 1994.

[Bouhoula, 1996] A. Bouhoula. Using induction and rewriting to verify and complete parameterized specifications. *Theoretical Computer Science*, 1-2(170) :245–276, 1996.

[Bouhoula, 1997] A. Bouhoula. Automated theorem proving by test set induction. *Journal of Symbolic Computation*, 23 :47–77, 1997.

[Bouhoula, 1999] A. Bouhoula. A new procedure for simultaneously checking completeness and ground confluence. Rapport Technique 99-R-051, LORIA, 1999. A paraître dans 15th IEEE International Conference on Automated Software Engineering, Grenoble (France).

[Bouma et Velthuijsen, 1994] rédacteurs L. G. Bouma et H. Velthuijsen. *Feature Interactions in Telecommunications Systems*. IOS Press, 1994.

[Bouma et Zuidweg, 1993] W. Bouma et H. Zuidweg. Formal analysis of feature interactions by model checking. Technical Report TI-PU-93-868, PTT Research, 1993.

[Boumezbeur et Logrippo, 1993] R. Boumezbeur et L. Logrippo. Specifying telephone systems in LOTOS. *IEEE Communication Magazine*, 31(8) :38–45, Août 1993.

[Bousdira et Rémy, 1990] W. Bousdira et J.-L. Rémy. On sufficient completeness of conditional specifications. Dans *Proceedings 2nd International Workshop on Conditional and Typed Rewriting Systems*, rédacteurs S. Kaplan et M. Okada, volume 516 de *Lecture Notes in Computer Science*, pages 272–283. Springer-Verlag, 1990.

[Bousdira, 1990] W. Bousdira. *Etude des propriétés des systèmes de réécriture conditionnelle. Mise en œuvre d'un algorithme de complétion.* Thèse de doctorat, Institut National Polytechnique de Lorraine, 1990.

[Boyer et Moore, 1979] R. S. Boyer et J S. Moore. *A Computational Logic*. Academic Press, New York, 1979.

[Boyer et Moore, 1985] R. S. Boyer et J S. Moore. Integrating decision procedures into heuristic theorem provers : A case study with linear arithmetic. ICSCA-CMP-44, University of Texas at Austin, 1985. Aussi publié dans *Machine Intelligence 11*, Oxford University Press, 1988.

[Boyer et Moore, 1991] R. S. Boyer et J S. Moore. MJRTY - a fast majority vote algorithm. Dans *Automated Reasoning : Essays in Honor of Woody Bledsoe*, rédacteur R. S. Boyer, volume 1 de *Automated Reasoning*, pages 105–117. Kluwer Academic Publishers, 1991.

[Braithwaite et Atlee, 1994] K.H. Braithwaite et J.M. Atlee. Towards automated detection of feature interactions. Dans Bouma et Velthuijsen [1994], pages 36–59.

[Bérard et Fribourg., 1999] B. Bérard et L. Fribourg. Automated verification of a parametric real-time program : the ABR conformance protocol. Dans *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99)*, numéro 1633 dans Lecture Notes in Computer Science, pages 96–107, Juillet 1999.

[Bredereke et Gotzhein, 1994] J. Bredereke et R. Gotzhein. A case study on specification, detection and resolution of IN feature interactions with Estelle. Rapport Technique 245/94, FBI Kaiserslautern, Mai 1994.

[Bredereke et Gotzhein, 1995] J. Bredereke et R. Gotzhein. Specification, detection and resolution of IN feature interactions with Estelle. Dans *Formal Description Techniques VII*, rédacteurs D. Hogrefe et S. Leue. Chapman & Hall, Mai 1995.

[Bredereke, 1995] J. Bredereke. Formal criteria for feature interactions in telecommunications systems. Dans *IFIP International Working Conference on Intelligent Networks, Proceedings*, rédacteurs V. B. Iversen et J. Nørgaard, pages 83–97. IFIP TC6, Août 1995.

[Bredereke, 1996] J. Bredereke. Automata-theoretic vs. property-oriented approaches for the detection of feature interactions in IN. Dans *International Workshop on Advanced Intelligent Networks 1996 – AIN'96, Proceedings*, rédacteur T. Margaria, 1996.

[Bronsard *et al.*, 1996] F. Bronsard, U. Reddy et R. W. Hasker. Induction using term orders. *Journal of Symbolic Computation*, 16 :3–37, 1996.

[Bronsard et Reddy, 1991] F. Bronsard et U.S. Reddy. Conditional rewriting in Focus. Dans *Proc. 2nd CTRS Workshop*, volume 449 de *Lecture Notes in Computer Science*, pages 2–13, 1991.

[Bündgen et Küchlin, 1989] R. Bündgen et W. Küchlin. Computing ground reducibility and inductively complete positions. Dans *Proceedings 3rd Conference on Rewriting Techniques and Applications, Chapel Hill (North Carolina, USA)*, rédacteur N. Dershowitz, volume 355 de *Lecture Notes in Computer Science*, pages 59–75. Springer-Verlag, Avril 1989.

[Bundy *et al.*, 1989] A. Bundy, F. van Harmelen, A. Smaill et A. Ireland. Extensions to the rippling-out tactic for guiding inductive proofs. Dans *10th International Conference on Automated Deduction*, volume 449 de *Lecture Notes in Computer Science*, pages 132–146, 1989.

[Burch *et al.*, 1990] J.R. Burch, E.M. Clarke, K.L. McMillan et D.L. Dill. Symbolic model checking : 10^{20} states and beyond. Dans *Proc. 5th LICS Symp., Philadelphia (Pa., USA)*, pages 428–439. Springer Verlag, 1990.

[Cansell et Méry, 2000] D. Cansell et D. Méry. Playing with abstraction and refinement for managing features interactions. Dans *ZB2000*, Lecture Notes in Computer Sciences. Springer Verlag, Août 2000.

[Charnois, 1997] T. Charnois. A natural language processing approach for avoidance of feature interactions. Dans *Feature Interactions in Telecommunication Networks*, volume 4, pages 347–363. IOS Press, 1997.

[Chen *et al.*, 1997] Y. L. Chen, S. Lafortune et F. Lin. Resolving feature interactions using modular supervisory control with priorities. Dans *Feature Interactions in Telecommunication Systems*, pages 108–122. IOS Press, 1997.

[Cheng et Ohta, 1995] rédacteurs K. E. Cheng et T. Ohta. *Feature Interactions in Telecommunications III*. IOS Press, Octobre 1995.

[Clocksin et Mellish, 1981] W. Clocksin et C. Mellish. *Programming in Prolog*. Springer Verlag, 1981.

[Coglio *et al.*, 1997] A. Coglio, F. Giunchiglia, P. Pecchiari et C. L. Talcott. A logic level specification of the NQTHM simplification process. Rapport Technique 9706-07, IRST, Juillet 1997.

[Combes et Pickin, 1994] P. Combes et S. Pickin. Formalisation of a user view of network and services for feature interaction detection. Dans Bouma et Velthuijsen [1994], pages 120–135.

[Comon et Jacquemard, 1997] H. Comon et F. Jacquemard. Ground reducibility is EXPTIME-complete. Dans *Proc. IEEE Symp. on Logic in Computer Science*. IEEE Comp. Soc. Press, Juin 1997.

[Comon, 1989] H. Comon. Inductive proofs by specification transformations. Dans *Proc. of 3rd Conference on Rewriting Techniques and Applications (RTA'89)*, rédacteur N. Dershowitz, volume 775 de *Lecture Notes in Computer Science*, pages 76–91. Springer Verlag, 1989.

[Cooper, 1972] D. C. Cooper. Theorem proving in arithmetic without multiplications. *Machine Intelligence*, 6 :43–59, 1972.

[Cyrluk *et al.*, 1996] D. Cyrluk, P. Lincoln et N. Shankar. On Shostak's decision procedure for combinations of theories. Dans *Automated Deduction—CADE-13*, rédacteurs M. A. McRobbie et J. K. Slaney, numéro 1104 dans Lecture Notes in Artificial Intelligence, pages 463–477, New Brunswick, NJ, Juillet–Août 1996. Springer-Verlag.

[Dankel *et al.*, 1994] D. D. Dankel, M. Schmalz, W. Walker, K. Nielsen, L. Muzzi et D. Rhodes. An architecture for defining features and exploring interactions. Dans Bouma et Velthuijsen [1994], pages 258–271.

[Dershowitz *et al.*, 1988] N. Dershowitz, M. Okada et G. Sivakumar. Canonical conditional rewrite systems. Dans *9th CADE Conference*, volume 310 de *Lecture Notes in Computer Science*. Springer Verlag, Mai 1988.

[Dershowitz et Jouannaud, 1990] N. Dershowitz et J.P. Jouannaud. *Rewrite systems*, volume B :Formal Methods and Semantics de *Handbook of Theoretical Computer Science*, chapitre 6, pages 243–320. North-Holland, Amsterdam, 1990.

[Dershowitz et Manna, 1979] N. Dershowitz et Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8) :465–476, 1979.

[Dershowitz, 1982] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3) :279–301, 1982.

[Dijkstra, 1976] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, 1976.

[Ehrig et Mahr, 1985] H. Ehrig et B. Mahr. *Fundamentals of Algebraic Specification 1. Equations and initial semantics*, volume 6 de *Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Encontre, 1989] V. Encontre. Geode : An industrial environment for designing real time distributed systems in SDL. Dans *Proceedings of the 4th SDL Forum*. North-Holland, 1989.

[Enderton, 1977] H. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1977.

[Faci et Logrippo, 1994] M. Faci et L. Logrippo. Specifying features and analysing their interactions in a LOTOS environment. Dans Bouma et Velthuijsen [1994], pages 136–151.

[Fernandez *et al.*, 1992] J.-C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez et J. Sifakis. A toolbox for the verification of LOTOS programs. Dans *Proceedings of the 14th International Conference on Software Engineering ICSE'14*. Melbourne, Australia, 1992.

[Fitting, 1990] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, 1990.

[Frappier *et al.*, 1996] M. Frappier, A. Mili et J. Desharnais. Defining and detecting feature interactions. Rapport Technique 181, Université de Sherbrooke, 1996.

[Fribourg, 1989] L. Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8(3) :253–276, Septembre 1989.

[Fribourg, 1998] L. Fribourg. A closed-form evaluation for extended timed automata. Rapport Technique LSV-98-2, Lab. Specification and Verification, ENS de Cachan, Mars 1998. 17 pages.

[Fritsche, 1995] N. Fritsche. Runtime resolution of feature interactions in architectures with separated call and feature control. Dans Cheng et Ohta [1995], pages 43–63.

[Garland et Guttag, 1988] S. J. Garland et J. V. Guttag. Inductive methods for reasoning about abstract data types. Dans *ACM Symposium on Principles of Programming Languages*, pages 219–228, 1988.

[Geist et others, 1994] A. Geist et al. *PVM : Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[Giunchiglia *et al.*, 1994] F. Giunchiglia, P. Pecchiari et C. L. Talcott. Reasoning theories : Towards an architecture for open mechanized reasoning systems. Rapport Technique 9409-15, IRST, Novembre 1994.

[Gordon et Melham, 1993] M.-J.-C. Gordon et T.-F. Melham. *Introduction to HOL : a theorem proving environment for higher order logic*. Cambridge University Press, 1993. ISBN 0-521-44189-7.

[Gramlich, 1989] B. Gramlich. UNICOM : a refined completion based inductive theorem prover. Dans *10th International Conference on Automated Deduction*, rédacteur M. E. Stickel, volume 449 de *Lecture Notes in Artificial Intelligence*, pages 655–656. Springer-Verlag, Juillet 1989.

[Griffeth et Velthuijsen, 1994] N. Griffeth et H. Velthuijsen. The negotiating agents approach to runtime feature interaction resolution. Dans Bouma et Velthuijsen [1994], pages 217–235.

[groupe PROTHEO, 1996 a 1998] Le groupe PROTHEO. Rapports trimestriels et annuels dans le cadre du contrat 96 1B 008 avec le CNET, 1996 à 1998.

[Grätzer, 1979] G. Grätzer. *Universal Algebra*. Springer, 1979. second edition.

[Guttag, 1975] J. Guttag. *The Specification and Application to Programming of Abstract Data Types*. Thèse de doctorat, University of Toronto, 1975.

[Henzinger *et al.*, 1997] T. A. Henzinger, P. H. Ho et H. Wong-Toi. HYTECH : A model checker for hybrid systems. Dans *CAV'97*, numéro 1254 dans Lecture Notes in Computer Science, pages 460–463. Springer-Verlag, 1997.

[Hofbauer et Huber, 1994] D. Hofbauer et M. Huber. Linearizing term rewriting systems using test set. *Journal of Symbolic Computation*, 17(1) :91–129, 1994.

[Holzmann et Peled, 1995] G.J. Holzmann et D. Peled. An improvement in formal verification. *Formal Description Techniques*, 7, Mai 1995.

[Howe, 1993] D. J. Howe. Reasoning about functional programs in Nuprl. Dans *Functional Programming, Concurrency, Simulation and Reasoning*, numéro 693 dans Lecture Notes in Computer Science, pages 145–164, 1993.

[Huet et Hullot, 1982] G. Huet et J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2) :239–266, Octobre 1982.

[Huet et Lankford, 1978] G. Huet et D. S. Lankford. On the uniform halting problem for term rewriting systems. Rapport Technique 283, Laboria, France, 1978.

[Huet et Oppen, 1980] G. Huet et D. Oppen. *Equations and Rewrite Rules : A Survey*, pages 349–405. Formal Language Theory : Perspectives and Open Problems. Academic Press, New York, 1980.

[Iraqi et Erradi, 1997] Y. Iraqi et M. Erradi. Method for supporting and elimination of feature interaction in a telecommunication system. Dans *Feature Interactions in Telecommunication Systems*, pages 298–312. IOS Press, 1997.

[ISO/TC 97/SC 21, 1988] ISO 8807 ISO/TC 97/SC 21. Information processing systems - Open Systems Interconnection - LOTOS : A formal description technique based on the temporal ordering of observational behaviour, 1988.

[ISO/TC 97/SC 21, 1989] ISO 9074 ISO/TC 97/SC 21. Information processing systems - Open Systems Interconnection - ESTELLE : A formal description technique based on an extended state transition model, 1989.

[ITU-T, 1993a] ITU-T. Distributed functional plane for intelligent network CS-1, 1993. Q-1214.

[ITU-T, 1993b] ITU-T. Message Sequence Charts (MSC), 1993. Q. 1201.

[Jackson, 1994] P. B. Jackson. *The Nuprl Proof Development System. Version 4.1 Reference Manual and User's Guide*. Cornell University, 1994.

[Jain, 1996] R. Jain. Congestion control and traffic management in ATM networks : Recent advances and a survey. *Computer Networks and ISDN Systems*, 28 :1723–1738, 1996. ftp ://ftp.netlab.ohio-state.edu/pub/jain/papers/cnis/index.html.

[Janičić *et al.*, 1999] P. Janičić, A. Bundy et I. Green. A framework for the flexible integration of a class of decision procedures into theorem provers. Dans *Proc. of the 16th International Converence on Automated Deduction (CADE-16)*, numéro 1632 dans Lecture Notes in Artificial Intelligence, pages 127–141. Springer Verlag, 1999.

[Jouannaud *et al.*, 1982] J.-P. Jouannaud, P. Lescanne et F. Reinig. Recursive decomposition ordering. Dans *Formal Description of Programming Concepts 2*, rédacteur D. Bjørner, pages 331–348, 1982.

[Jouannaud et Kounalis, 1989] J.-P. Jouannaud et E. Kounalis. Automatic proofs by induction in theories without constructors. *Information and Computation*, 82 :1–33, 1989.

[Kaplan et Choquer, 1986] S. Kaplan et M. Choquer. On the decidability of quasi-reducibility. *Bulletin of European Association for Theoretical Computer Science*, 28 :1–33, 1986.

[Kaplan, 1984] S. Kaplan. Fair conditional term rewriting systems : Unification, termination and confluence. Rapport technique, Université de Paris Sud, Orsay (France), 1984.

[Kaplan, 1987] S. Kaplan. Simplifying conditional term rewriting systems : Unification, termination and confluence. *Journal of Symbolic Computation*, 4(3) :295–334, Décembre 1987.

[Kapur *et al.*, 1987] D. Kapur, P. Narendran et H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24 :395–415, 1987.

[Kapur *et al.*, 1991] D. Kapur, P. Narendran et H. Zhang. Automating inductionless induction using test sets. *Journal of symbolic Computation*, 11 :83–112, 1991.

[Kapur *et al.*, 1994] D. Kapur, D.R. Musser et X. Nie. An overview of the tecton proof system. *Thoretical Computer Science*, 133, 1994.

[Kapur et Nie, 1994] D. Kapur et X. Nie. Reasoning about numbers in TECTON. Dans 8^{th} *Intl. Symp. Methodologies for Intelligent Systems*, pages 57–70, Octobre 1994.

[Kapur et Zhang, 1989] D. Kapur et H. Zhang. An overview of RRL : Rewrite Rule Laboratory. Dans *International Conference on Rewriting Techniques and its Applications(RTA-89)*, numéro 355 dans Lecture Notes in Computer Science, pages 513–529. Chapel Hill, 1989.

[Kapur, 1997] D. Kapur. Shostak's congruence closure as completion. Dans *RTA'97*, 1997.

[Käufl, 1988] T. Käufl. Reasoning about systems of linear inequalities. Dans *Proceedings of 9th International Conference on Automated Deduction*, pages 563–572, 1988.

[Kaufmann et Moore, 1999] M. Kaufmann et J S. Moore. *ACL2 Version 2.4 – The User's Manual*, 1999.

[Küchlin, 1989] W. Küchlin. Inductive completion by ground proof transformation. Dans *Colloquium on the Resolution of Equations in Algebraic Structures, Volume 2 : Rewriting Techniques*, rédacteurs H. Aït-Kaci et M. Nivat, pages 211–244. Academic Press, 1989.

[Kühler, 1999] U. Kühler. *A Tactic-Based Inductive Theorem Prover for Data Types with Partial Operations*. Thèse de doctorat, Universität Kaiserslautern, 1999.

[Kirchner et Viry, 1992] C. Kirchner et P. Viry. Implementing parallel rewriting. Dans *Parallelization in Inference Systems*, rédacteurs B. Fronhöfer et G. Wrightson, volume 590 de *Lecture Notes in Artificial Intelligence*, pages 123–138. Springer Verlag, 1992.

[Kirchner, 1991] H. Kirchner. Proofs in parameterized specifications. Dans *RTA'4*, pages 174–187. Springer Verlag, 1991.

[Klay *et al.*, 1999] F. Klay, M. Rusinowitch et S. Stratulat. Analysing feature interactions with automated deduction systems. Dans *7th ICOTS - International Conference on Telecommunication Systems, Modelling and Analysis*, pages 542–554, Mars 1999.

[Klop, 1992] J. W. Klop. *Term Rewriting Systems*, volume 2 de *Handbook of Logic in Computer Science*, chapitre 1, pages 1–116. Clarendon Press, 1992.

[Knuth et Bendix, 1970] D. Knuth et P. Bendix. Simple word problems in universal algebras. Dans *Computational problems in abstract algebra*, rédacteur Leech, pages 263–297. Pergamon Press, 1970.

[Korver, 1993] H.P. Korver. Detecting feature interactions with CÆSAR/ALDÉBARAN. Rapport Technique CS-R9370, Centrum voor Wiskunde en Informatica, Amsterdam, Décembre 1993.

[Kounalis et Rusinowitch, 1990] E. Kounalis et M. Rusinowitch. Mechanizing inductive reasoning. *Bulletin of European Association for Theoretical Computer Science*, 41 :216–226, Juin 1990.

[Kounalis et Rusinowitch, 1991] E. Kounalis et M. Rusinowitch. Studies on the ground convergence property of conditional theories. Dans *Algebraic Methodology and Software Technology '91*, pages 363–376. Springer Verlag, 1991.

[Kounalis, 1985a] E. Kounalis. Completeness in data type specifications. Dans *Proceedings EUROCAL Conference*, volume 204 de *Lecture Notes in Computer Science*, pages 348–362, Linz(Austria), 1985.

[Kounalis, 1985b] E. Kounalis. *Validation de spécifications algébriques par complétion inductive.* Thèse de doctorat, Université de Nancy 1, 1985.

[Kounalis, 1992] E. Kounalis. Testing for the ground (co)-reducibility property in term rewriting systems. *Theoretical Computer Science*, 106 :87–117, 1992.

[Lamport, 1991] L. Lamport. The temporal logic of actions. Rapport Technique 79, Digital Equipment Corp., Palo Alto,California, USA, Décembre 1991.

[Lassez et Maher, 1992] J.-L. Lassez et M.J. Maher. On Fourier's algorithm for linear arithmetic constraints. *Journal of Automated Reasoning*, 9 :373–379, 1992.

[Leroy *et al.*, 2000] X. Leroy, D. Doligez, J. Garrigue, D. Rémy et J. Vouillon. *The Objective Caml system, release 3.00. Documentation and user's manual*, 2000.

[Lescanne, 1984] P. Lescanne. Term rewriting systems and algebra. Dans *Proceedings 7th International Conference on Automated Deduction*, rédacteur R. Shostak, Lecture Notes in Computer Science, 1984.

[Lin et Lin, 1994] F. J. Lin et Y.-J. Lin. A building block approach to detecting and resolving feature interactions. Dans Bouma et Velthuijsen [1994], pages 86–119.

[Loveland, 1978] D. Loveland. *Automated Theorem Proving : A Logical Basis.* North-Holland, Amsterdam, 1978.

[Mermet *et al.*, 1998] B. Mermet, D. Méry et D. Samborski. Spécification de services : une approche avec B. *Technique et Science Informatiques*, 17(9) :1157–1180, 1998.

[Monin et Klay, 1999] J.F. Monin et F. Klay. Correctness proof of the standardized algorithm for ABR conformance. Dans *Formal Methods (FM) '99*, rédacteurs J. Wing, J. Woodcock et J. Davies, numéro 1709 dans Lecture Notes in Computer Science, pages 662–681. Springer Verlag, 1999.

[Musser, 1980] D. R. Musser. On proving inductive properties of abstract data types. Dans *Proceedings 7th ACM Symposium on Principles of Programming Languages*, pages 154–162. ACM, 1980.

[Naidich, 1996] D. Naidich. On generic representation of implicit induction procedures. Rapport Technique CS-R9620, CWI, 1996.

[Navarro et Orejas, 1987] M. Navarro et F. Orejas. Parameterized Horn clause specifications : proof theory and correctness. Dans *Proceedings TAPSOFT Conference*, volume 249 de *Lecture Notes in Computer Science*. Springer Verlag, 1987.

[Necula, 1998] G. C. Necula. *Compiling with proofs*. Thèse de doctorat, Carnegie Mellon University, Septembre 1998.

[Nelson et Oppen, 1979] G. Nelson et D.C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2) :245–257, 1979.

[Nelson, 1981] G. Nelson. Techniques for program verification. Rapport Technique CSL-81-10, Xerox Palo Alto Research Center, 1981.

[Owre *et al.*, 1992] S. Owre, J. M. Rushby et N. Shankar. PVS : A prototype verification system. Dans *11th International Conference on Automated Deduction (CADE)*, rédacteur D. Kapur, volume 607 de *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer Verlag, 1992.

[Padawitz, 1988] P. Padawitz. *Computing in Horn Clauses*. Springer-Verlag, 1988.

[Padawitz, 1992] P. Padawitz. *Deduction and Declarative Programming*. Cambridge University Press, 1992.

[Plaisted, 1978] D. Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Rapport Technique R-78-943, Dept. of Computer Science, Univ. of Illinois, Urbana, 1978.

[Plaisted, 1985] D. Plaisted. Semantic confluence and completion method. *Information and Control*, 65 :182–215, 1985.

[Plaisted, 1993] D. Plaisted. *Equational Reasoning and Term Rewriting Systems*, volume 1 de *Handbook of Logic in Artificial Intelligence and Logic Programming*, chapitre 5, pages 273–364. Clarendon Press-Oxford, 1993.

[Rabadan et Klay, 1997] C. Rabadan et F. Klay. Un nouvel algorithme de contrôle de conformité pour la capacité de transfert "Available Bit Rate". Rapport Technique NT/CNET/5476, CNET, 1997.

[Rabadan, 1997] C. Rabadan. L'ABR et sa conformité. Rapport Technique NT DAC/ARP/034, CNET, 1997.

[Reddy, 1990] U. Reddy. Term rewriting induction. Dans *10th International Conference on Automated Deduction*, volume 814 de *Lecture Notes in Computer Science*, pages 162–177, 1990.

[Rémy, 1982] J. L. Rémy. *Etudes des systèmes de réécriture conditionnelles et applications aux types abstraits algébriques*. Thèse de doctorat, Université de Nancy I, 1982.

[Ruess et Shankar, 2001] H. Ruess et N. Shankar. Deconstructing Shostak. Submitted to LICS 2001, 2001.

[Rusinowitch *et al.*, 2000] M. Rusinowitch, S. Stratulat et F. Klay. Mechanical verification of an ideal incremental ABR conformance algorithm. Dans *Proceedings of 12th International Conference on Computer Aided Verification (CAV'2000)*, rédacteurs E. A. Emerson et A. P. Sistla, numéro 1855 dans Lecture Notes in Computer Science, pages 344–357. Springer Verlag, Juillet 2000.

[Rusinowitch, 1989] M. Rusinowitch. *Démonstration automatique-Techniques de réécriture*. InterEditions, 1989.

[Schmid et Fettig, 1995] K. Schmid et R. Fettig. Towards an efficient construction of test sets for deciding ground reducibility. Dans *Proc. of the 6th Conference on Rewriting Techniques and Applications (RTA'95)*, rédacteur J. Hsiang, volume 914 de *Lecture Notes in Computer Science*, pages 86–100. Springer Verlag, 1995.

[Shostak, 1977] R. Shostak. On the SUP-INF method for proving Presburger formulas. *JACM*, 24 :529–543, 1977.

[Shostak, 1979] R.E. Shostak. A practical decision procedure for arithmetic with function symbols. *ACM*, 2(26) :351–360, 1979.

[Shostak, 1984] R. Shostak. Deciding combination of theories. *Journal of the ACM*, 31(1) :1–12, 1984.

[Stepien et Logrippo, 1995] B. Stepien et L. Logrippo. Feature interaction detection by using backward reasoning with LOTOS. Dans *Protocol Specification, Testing and Verification XIV*, rédacteurs S.T. Vuong et S.T. Chanson, pages 71–86. Chapman & Hall, 1995.

[Stratulat, 1998a] S. Stratulat. Applying semantic subsumption rules in the context of inductive proofs. Dans *Workshop on Integration of Deductive Systems, CADE-15*, pages 85–95, 1998.

[Stratulat, 1998b] S. Stratulat. *SPIKEpar* : une interface parallèle du démonstrateur automatique SPIKE. Dans *Dixièmes Rencontres Francophones du Parallélisme (RenPar'10)*, pages 209–212, 1998.

[Stratulat, 1999] S. Stratulat. A general framework to build multi-logic implicit induction provers. Rapport Technique 99-R-310, LORIA, Novembre 1999.

[Stratulat, 2000] S. Stratulat. A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 2000. 43 pages (to appear).

[Thomas, 1998] M. Thomas. Modelling and analysing user views of telecommunications services. Dans *Fifth Feature Interaction Workshop*, 29 Septembre – 1 Octobre 1998.

[Tiwari, 2000] A. Tiwari. *Decision procedures in automated deduction.* Thèse de doctorat, State University of New York at Stony Brook, Mai 2000.

[Tsang et Magill, 1998] S. Tsang et E. H. Magill. Learning to detect and avoid run-time feature interactions in intelligent networks. *IEEE Transactions on Software Engineering*, pages 818–830, Octobre 1998.

[Turner, 1993] K.J. Turner. *Using formal description techniques - An introduction to ESTELLE, LOTOS and SDL.* Wiley, 1993.

[Vardi et Wolper, 1986] M. Vardi et P. Wolper. An automata-theoretic approach to automatic program verification. Dans *Proc. 1st LICS Symp., Cambridge (Mass., USA)*, pages 322–331, Juin 1986.

[Wall et others, 1997] L. Wall et al. *Programming Perl.* O'Reilly, 2nd édition, 1997.

[Walther, 1993] C. Walther. Combining induction axioms by machine. Dans *Proc. of the 12th International Joint Conference on Artificial Intelligence*, volume 1, pages 95–100, 1993.

[Wirsing, 1990] M. Wirsing. *Algebraic specification*, volume Handbook of Theoretical Computer Science B : Formal Methods and Semantics, chapitre 13. Elsevier, 1990.

[Wirth, 1997] C.-P. Wirth. *Positive/Negative-Conditional Equations : A Constructor-Based Framework for Specification and Inductive Theorem Proving.* Thèse de doctorat, University of Kaiserslautern, 1997.

[Wirth, 2000] C.-P. Wirth. Descente infinie + analytic deduction. Rapport technique, Universität Dortmund, Informatik V, Mai 2000.

[Zhang *et al.*, 1988] H. Zhang, D. Kapur et M. S. Krishnamoorthy. A mechanizable induction principle for equational specifications. Dans *Proceedings 9th International Conference on Automated Deduction, Aragonne (Illinois, USA)*, numéro 310 dans Lecture Notes in Computer Science, pages 162–181. Springer-Verlag, 1988.

Index

Glossaire

CFU : Call Forward Unconditional
TCS : Terminating Call Screening
ABD : Abbreviated Dialling
OCS : Originating Call Screening
POTS : Plain Old Telephone System
ATM : Asynchronous Transfer Mode
GCRA : Generic Cell Rate Algorithm
DGCRA : Dynamic GCRA
ABR : Available Bit Rate
CBR : Constant Bit Rate
VBR : Variable Bit Rate
ACR : Allowed Cell Rate
RM : Resource Management

Résumé

Le processus de certification de logiciels est dans la plupart des cas une tâche laborieuse et coûteuse qui nécessite aussi bien des méthodes mathématiques, pour exprimer sans ambiguïté et de façon structurée le comportement attendu du logiciel, que des outils automatiques pour vérifier ses propriétés. Parmi les techniques de preuve, la récurrence est parfaitement adaptée pour raisonner sur des structures de données infinies, comme les entiers et les listes, ou des systèmes paramétrés.

Cette thèse comprend deux parties, l'une théorique, l'autre applicative. La première partie est centrée autour d'un nouveau concept, l'*ensemble couvrant contextuel* (ECC). Le principe de preuve par récurrence avec ECC est exprimé par un système d'inférence abstrait qui introduit des conditions suffisantes pour son application correcte. La conception modulaire de règles d'inférence concrètes est un avantage de cette approche. Comme étude de cas, nous spécifions le système d'inférence du démonstrateur SPIKE en tant qu'instance de ce système.

Dans la deuxième partie, nous analysons tout d'abord le problème d'interactions de services téléphoniques. Nous proposons une méthodologie pour les détecter et les résoudre, reposant sur des techniques basées sur la réécriture conditionnelle et la récurrence. Dans une autre application, nous obtenons, à l'aide du démonstrateur PVS, la première preuve formelle de l'équivalence entre deux algorithmes de conformité du protocole ABR. Puis, nous utilisons SPIKE pour vérifier complètement automatiquement la majorité des 80 lemmes de cette preuve.

Mots-clés: ensembles couvrants contextuels, preuve par récurrence, intégration de modules de raisonnement, validation de logiciels de télécommunications

Abstract

The software certification process is in most of the cases a laborious and costly task that needs not only mathematical methods to express clearly and in a structured manner the software's expected behavior but also automatic tools to prove some of its properties. Among the proof techniques, induction is well-suited to reason on infinite data structures, like integers and lists, or parameterized systems.

This thesis contains a theoretical and an applicative part. The first one is centered around the new concept of contextual cover set (CCS). The principle of induction with CCSs is reflected by an abstract inference system introducing sufficient conditions for its sound usage. The modular design of concrete inference rules is an advantage of this approach. As a case study, we specify the SPIKE prover as an instance of this system.

In the second part, we first analyze the feature interaction problem in telecommunications. We propose a methodology for their detection and resolution by using techniques based on conditional rewriting and induction. In another application, we obtain the first formal proof of a generic incremental ABR conformance algorithm, by using the PVS prover. Then, we use SPIKE to verify completely automatically the most of the 80 user-defined lemmas.

Keywords: contextual cover sets, proof by induction, integration of reasoning modules, validation of telecommunications software