Logique Mathématique
Raisonnement Automatique
Vérification de Démonstration

Laboratoire d'accueil
L.A.C.L.
Université Paris XII

THÈSE

Pour l'obtention du titre de
DOCTEUR DE L'UNIVERSITÉ PARIS XII
Discipline : Informatique

Présentée par

Andriy PASKEVYCH

Méthodes de formalisation des connaissances et des raisonnements mathématiques : aspects appliqués et théoriques

COMPOSITION DU JURY

Gilles DOWEK,	Professeur École Polytechnique,	Rapporteur
Michaël RUSINOWITCH,	Directeur de Recherche LORIA,	Rapporteur
Sergei SOLOVIEV,	Professeur à l'I.R.I.T.,	Rapporteur
Patrick CÉGIELSKI,	Professeur Paris XII,	Examinateur
Anatol SLISSENKO,	Professeur Paris XII,	Examinateur
Konstantin VERCHININE,	Professeur Paris XII,	Directeur de thèse

Soutenue le 21 décembre 2007

Je tiens à remercier :

Monsieur Konstantin Verchinine, Professeur à l'Université Paris XII. Je considère comme une chance énorme d'avoir eu comme chef une personne vivement impliquée dans le travail sans imposer ses opinions, un directeur avec qui je pouvais être en désaccord, avoir des discussions passionnées — à qui je pouvais dire, parfois des mois après : «*Vous avez eu raison*». Je ne le remercierai jamais assez pour sa patience et son impatience, pour son aide et sa volonté de me laisser trouver mes propres solutions, pour être devenu mon maître et pour tant d'autres raisons.

Monsieur Alexander Lyaletski, Directeur de recherche à l'Université Nationale de Kiev et mon co-directeur de thèse, pour m'avoir initié à mon travail et pour mes premières leçons très importantes sur la recherche scientifique.

Monsieur Gilles Dowek, Professeur à l'École Polytechnique, Monsieur Michaël Rusinowitch, Directeur de recherche à LORIA et Monsieur Sergei Soloviev, Professeur à l'I.R.I.T. (Institut de Recherche en Informatique de Toulouse) pour avoir accepté de rapporter ce travail.

Monsieur Patrick Cégielski, Professeur à l'Université Paris XII et Monsieur Anatol Slissenko, Professeur à l'Université Paris XII pour avoir accepté de faire partie de mon jury.

La France, le pays et l'état, pour son hospitalité généreuse, pour toutes les opportunités et toute l'assistance qu'elle offre aux étudiants des autres pays.

L'ensemble des membres du L.A.C.L. (Laboratoire d'Algorithmique, Complexité et Logique) pour m'avoir accueilli en thèse et pour mes premiers contacts amicaux en France.

Mes anciens collègues au Département Informatique à l'IUT Fontainebleau pour leur sympathie permanente.

Madame Laurence Goblot pour m'avoir offert un havre pendant trois années, pour son amitié et pour son esprit latin qui m'est pourtant paru très parent.

Monsieur Vladimir Donchenko, Professeur à l'Université Nationale de Kiev et Monsieur Patrick Cégielski, une deuxième fois, pour leur soutien ferme, sans lequel ce travail n'aurait pu ni aboutir ni même commencer.

Mes anciens enseignants à la Faculté de Cybernétique à l'Université Nationale de Kiev : Monsieur Andrei Stavrovsky, Monsieur Leonid Lisovik, Monsieur Vladimir Redko, Monsieur Igor Protasov, Madame Olga Shkaravskaya, Monsieur Yuriy Koval.

Mon condisciple Dmitry Astapov pour m'avoir initié à beaucoup de sujets en informatique qui n'étaient pas enseignés à l'époque dans les classes universitaires.

Ma famille pour son soutien illimité, sa patience et sa confiance inébranlable durant ces six derniers années. J'espère pouvoir donner un jour autant qu'on m'a donné.

Table des matières

Table des figures

Introduction

Dans la thèse, nous étudions les moyens de présentation des connaissances mathématiques ainsi que les schémas du raisonnement mathématique. Notre recherche est destinée à la construction d'un système de vérification automatique de textes mathématiques formalisés.

Notre approche est basée sur les principes suivants :

- Le texte à vérifier est écrit dans un langage formel qui imite la langue et le style naturels des publications mathématiques. La vérification consiste en la démonstration que le texte est, premièrement, «sensé», c'est-à-dire que toutes les fonctions et relations sont employées dans leurs domaines, conformément aux définitions (on dit d'un tel texte qu'il est «ontologiquement correct») ; et, deuxièmement, «fondé», c'est-à-dire que toutes les affirmations sont déductibles de leurs prémisses dans le texte (on dit d'un tel texte qu'il est «logiquement correct»).
- La recherche de preuve, nécessaire pour la vérification, est effectuée à deux niveaux. Le niveau bas est un démonstrateur automatique puissant, basé sur une procédure combinatoire cohérente et complète en logique classique du premier ordre. (L'existence de démonstrateurs puissants pour cette logique est une raison pour laquelle elle est choisie comme notre logique de base). Le niveau haut est un «raisonneur», le dirigeant d'un assortiment des méthodes heuristiques, dont le devoir est de transformer : filtrer, simplifier, décomposer une tâche de preuve avant de la passer au démonstrateur.

Ce «raisonneur» est le cœur du système. L'idée est d'exploiter les indices qui nous sont donnés par la forme humaine du problème à travailler (ce qu'une procédure combinatoire ne fait jamais) : traiter les définitions autrement que les axiomes, les démonstrations par analyse de cas autrement que les autres schémas de preuve, l'appartenance à une classe d'objets mathématiques autrement que les relations ordinaires et ainsi de suite.

En somme, nous tenons à un assistant de preuve fort : fort grâce à un démonstrateur puissant au fond, et fort grâce à un raisonneur sophistiqué devant le démonstrateur.

Les résultats principaux de notre travail sont les suivants :

1. Le développement de ForTheL, un langage des théories formelles. La syntaxe du langage est définie sous forme de règles BNF. La sémantique des propositions ForTheL est définie par une procédure de traduction dans le langage du premier ordre.
2. La définition formelle d'un texte correct dans le langage ForTheL. La définition est donnée en termes d'un calcul séquentiel. La cohérence de ce calcul est démontrée.
3. L'introduction de la notion de validité locale et de transformations qui préservent les propositions localement valides. Cela forme le fondement théorique pour la procédure de vérification et pour les algorithmes du raisonneur.
4. Une variante cohérente et complète de la méthode d'élimination de modèles (tableaux de connexions) en logique du premier ordre avec égalité.
5. L'implantation des principes et des résultats ci-dessus en logiciel : dans un assistant de preuve appelé SAD («System for Automated Deduction»). Le système SAD est un logiciel libre, distribué sur le site du projet `http://nevidal.org.ua`.
6. Un certain nombre de textes mathématiques formalisés en ForTheL et vérifiés avec SAD : les théorèmes de Ramsey infini et fini, le théorème des restes chinois, le théorème de Tarski-Knaster et d'autres.

Programme «Evidence Algorithm»

Au début des années 1960 V. Glushkov a initié une série de recherches destinée au développement d'un système qui pourrait assister un mathématicien pratiquant dans la vérification de démonstrations longues mais, dans un certain sens, évidentes, telles que, par exemple, des chaînes de transformations algébriques et d'applications de définitions.

D'après la conception initiale, les trois composantes principales du système étaient : une procédure déductive qui implante le niveau basique de l'évidence ; une collection d'outils qui la renforcent ; un langage formel qui doit être proche de la langue naturelle des textes mathématiques. Une proposition est dite «évidente» si elle peut être démontrée par le système sans intervention de l'utilisateur. Ainsi, plus puissants sont la procédure déductive et les outils de support, plus large est le «domaine d'évidence». Le système évolue au fur et à mesure de l'apparition de nouvelles méthodes de renforcement ainsi que par l'accumulation de connaissances mathématiques importantes.

Le programme de recherche a pris le nom d'«Algoritm Ochevidnosti» (algorithme d'évidence). Ses principes sont exposés en détail dans les travaux [24, 26]. Le développement du programme est décrit dans l'article [13], dont la présentation ci-dessous est un extrait.

Les premières publications, consacrées à une méthode de démonstration automatique en théorie des groupes, sont apparues en 1966 [2]. Par la suite cette méthode s'est développée en une procédure déductive universelle [1]. Un langage formel de théories mathématiques (proche du langage du premier ordre) a été aussi proposé [27].

En 1970, la deuxième équipe de recherche a été formée. Son travail a été destiné au développement d'un système intégré de traitement de textes mathématiques. Un langage original de théories formelles proche de la langue naturelle a été développé [28] et sa sémantique définie en termes d'une théorie spéciale des ensembles [73].

En même temps, des systèmes déductifs variés étaient introduits et étudiés : un calcul séquentiel orienté-but sans skolémisation préliminaire [14] ; des calculs résolutionnels munis de stratégies différentes de traitement d'égalité [12] ; des méthodes de démonstration «de haut niveau» basées sur l'application de propositions auxiliaires [74, 3].

Les travaux d'implantation ont commencé en 1976. Le premier prototype, programmé en PL/I pour les machines ES (clonées de l'IBM 360), a été complété en 1978 [35]. En 1980, la première version du système SAD (Systema Avtomatizatsiyi Dokazatelstv) [25] est sortie.

Le programme a été abandonné vers le début des années 1990 suite au décès de Glushkov et aux perturbations globales dans l'Union Soviétique. Grâce aux efforts permanents de A. Lyaletski, la recherche s'est restaurée quelques années après. Le projet a pris le nom d'«Evidence Algorithm». La présente thèse fait partie de ce projet.

Approches existantes

Il nous paraît convenable de considérer le domaine de l'assistance mathématique (plus précisément, l'assistance à la démonstration) du point de vue de trois caractéristiques de base : le style de formalisation qu'un tel assistant suggère, le mode de démonstration qu'il exige et la «granularité» de démonstrations qu'il accepte. Nous allons établir la place de notre approche dans cet espace.

Style de formalisation. Par cela nous entendons le choix des faits préliminaires, la forme des définitions (doivent-elles être récursives), le genre du raisonnement (est-il constructif ou rigoureusement typé ou basé sur les calculs) et ainsi de suite. Il est évident que ce style est surtout influencé par le choix de la logique de base et des théories fondamentales qu'on utilise dans la formalisation.

Aujourd'hui, deux courants principaux consistent en l'adoption soit de la logique d'ordre supérieur (la théorie des types) soit de la logique du premier ordre (la théorie des ensembles). Les types sont utilisés dans la majorité des assistants mathématiques bien connus tels que Isabelle/HOL [56], Coq [7], HOL [29], ΩMEGA [65], PVS [57] et d'autres. L'approche basée sur la théorie des types favorise le raisonnement par induction et les définitions récursives ;

cette approche convient parfaitement à la formalisation dans le domaine de programmation ou d'ingénierie. Elle n'est, peut-être, pas aussi naturelle pour les mathématiques traditionnelles [38]. Toutefois la plupart des systèmes mentionnés offrent une collection vaste des théories purement mathématiques.

De l'autre côté nous voyons le système Mizar [69] qui est basé sur la logique classique du premier ordre et la théorie des ensembles de Tarski-Grothendieck. Cette approche correspond bien au style traditionnel de présentation mathématique et la Librairie Mathématique de Mizar (Mizar Mathematical Library, MML) constitue, à notre connaissance, la collection la plus large de textes mathématiques formels aujourd'hui.

La logique classique du premier ordre est aussi notre choix. Nous n'adoptons aucune théorie particulière des ensembles (ni d'autres théories fondamentales). Nous préférons définir un ensemble des faits préliminaires pour le problème en question et choisir les concepts basiques au niveau d'abstraction approprié. Cette approche est parfois classée comme le raisonnement dans les «petites théories», contrairement au développement d'une «grande théorie» universelle comme celle de Mizar.

Bien sûr, il y a des assistants de preuve qui n'appartiennent à aucune des deux catégories. Tel est le système ACL2 [36], qui travaille en logique du premier ordre sans quantificateurs avec induction. Ce formalisme ressemble à l'approche de la théorie des types, il impose les définitions récursives et les preuves par induction, mais il est aussi plus strict est plus adapté à la démonstration automatique. Le projet Theorema [9] est basé sur la logique des prédicats d'ordre supérieur.

Une classe particulière des systèmes, dits «logical frameworks» [60], offrent leur logique de base pour spécifier des formalismes déductifs. C'est dans ces logiques-objets qu'on développe des formalisations actuelles. Le premier projet de ce genre est Automath [54] par N. de Bruijn. Le système Isabelle engendre une série de systèmes déductifs : Isabelle/HOL, Isabelle/ZF etc.

Mode de démonstration. Une autre caractéristique importante d'un assistant de preuve est la sorte d'entrée qu'il accepte. Dans les systèmes interactifs, le plus souvent, une proposition est démontrée par une série d'instructions données au système. Ces instructions, dites «tactiques», peuvent être primitives, e.g. appliquer une règle d'inférence, ou bien complexes, e.g. appliquer une procédure de décision ou de simplification. Parmi les systèmes de ce genre, appelons-les «impératifs», sont Isabelle, PVS, Coq, HOL et d'autres.

Les assistants de preuve «déclaratifs», au contraire, acceptent des preuves rédigées dans le même langage que les propositions à prouver. Bien sûr, ce langage doit être muni de moyens pour organiser les propositions en preuves. Ensuite, l'assistant est obligé de vérifier chaque pas dans la démonstration. Mizar est le système le plus connu de ce genre. Isabelle, accompagnée par Isar [77] (un langage des preuves qui imite le style naturel), peut être aussi considérée déclarative. Le système SAD appartient clairement à cette catégorie.

Nous classons déclaratifs les démonstrateurs automatiques tels que Nqthm et ACL2 et les systèmes qui utilisent la planification des preuves : λClam [62], Ωmega, IsaPlanner [18]. En effet, les procédures autonomes, même si elles n'acceptent pas de preuves comme telles, s'appuient sur des prémisses (lemmes) données pour construire une dérivation. Donner de telles prémisses n'est qu'une manière d'écrire une preuve déclarative.

La distinction entre les deux catégories est assez vague. Si les pas de preuve dans un système déclaratif doivent être accompagnés par des indices détaillés pour le vérificateur, ce système peut être bien vu comme impératif (voir Isar). Par ailleurs, si, dans un assistant de preuve impératif, on peut construire des preuves n'employant que la tactique d'introduction d'un but intermédiaire et la tactique de finalisation automatique de but, de telles preuves sont déclaratives. Nous invitons le lecteur à se référer à l'article [31] pour un analyse approfondi de styles de preuve.

Granularité de démonstration. La puissance déductive d'un assistant de preuve peut être estimée par le nombre de pas de preuve que l'utilisateur est obligé de fournir au système : le nombre d'instructions pour les systèmes impératifs ou la longueur des preuves rédigées pour les systèmes déclaratifs. Nous pouvons distinguer les «contrôleurs de preuve» et les «chercheurs

de preuve». Les premiers n'acceptent que des démonstrations où chaque pas consiste en une application d'une règle d'inférence. Autrement dit, une preuve doit être absolument détaillée. Par exemple, Automath est un contrôleur de preuve par excellence. Mizar est un autre système de ce genre, bien que l'ensemble des règles d'inférence dans Mizar soit large et élaboré.

Les systèmes que nous appelons «chercheurs de preuve» appliquent les méthodes de démonstration automatique ou de planification de preuve et essaient de construire ou compléter les preuves par leurs propres efforts. Theorema, ACL2, ΩMEGA, aussi que le système SAD, tombent dans cette catégorie.

Les assistants impératifs ont d'habitude l'ensemble de tactiques augmentable. Par cela, leur «puissance de raisonnement» n'est pas inhérente à un système comme tel, et presque tous les systèmes impératifs implantent des procédures déductives assez fortes pour qu'ils puissent être qualifiés de chercheurs de preuve.

Présentation des travaux

Le premier chapitre est consacré à ForTheL («Formal Theory Language»), un langage formel des textes mathématiques [75, 46]. Un texte en ForTheL est composé de définitions, axiomes et théorèmes, hypothèses et affirmations, et de preuves des affirmations. ForTheL est un langage naturel contrôlé : sa syntaxe suit les règles de la grammaire anglaise. Les phrases typiques de ForTheL sont «`Let S be a finite set.`», «`Assume that m is less than n.`», «`Take an even prime number X.`», «`If p divides n-p then p divides n.`». La sémantique de propositions de ForTheL est définie par une série de transformations qui convertissent une proposition dans une formule du premier ordre, «l'image-formule» de cette proposition. On définit aussi les images-formules des phrases, des sections composées et des textes mêmes.

Les affirmations dans le texte peuvent être accompagnées avec une démonstration. ForTheL supporte les schémas de preuve différents, tels que *reductio ad absurdum*, l'analyse de cas et l'induction générale. Le langage n'impose aucune contrainte sur le niveau de détail dans une démonstration. Les pas de raisonnement peuvent être aussi grands que le vérificateur peut surmonter. Le processus de formalisation d'un texte mathématique en ForTheL est illustré par un exemple élaboré.

Le deuxième chapitre introduit la notion de texte correct : ontologiquement et logiquement. Répétons que dans un texte ontologiquement correct toutes les fonctions et relations sont employées dans leurs domaines, conformément aux définitions ; et que dans un texte logiquement correct toutes les affirmations sont déductibles de leurs prémisses dans le texte.

Pour pouvoir exprimer et démontrer des propriétés des occurrences particulières à l'intérieur d'une formule (ce qui est nécessaire pour la définition de la correction ontologique), nous développons et étudions la notion de proposition localement valide [59]. Nous justifions cette notion en démontrant que les lois de la logique classique (tels que *modus ponens*) peuvent être appliqués localement, et que l'équivalence localement valide est une condition suffisante pour le remplacement équivalent d'une sous-formule.

Ensuite, nous étudions les transformations des formules qui préservent les propriétés locales des sous-formules. Cela nous assure que les procédures du raisonneur (qui sont basées sur telles transformations) ne faussent pas la correction ontologique.

Sur ce fondement mathématique, nous définissons la correction ontologique et logique d'un texte formel en termes d'un calcul des textes corrects **CTC**. Ce calcul donne la sémantique précise des différents schémas de preuve admis dans ForTheL. En particulier, il définit comment «la thèse courante», c'est-à-dire la proposition à démontrer, est transformée au cours d'une démonstration.

Le troisième chapitre décrit le système SAD, son architecture et ses algorithmes [45, 46, 72]. Nous présentons les composantes principales du système. *L'analyseur syntaxique* de ForTheL construit la représentation interne du texte soumis. Le *vérificateur* contre-applique les règles du calcul **CTC** pour réduire la vérification d'un texte aux taches de preuve séparées. Le *générateur d'évidences* accumule des propriétés locales des termes considérés ce qui facilite considérable-

ment le contrôle ontologique. La procédure *motivatrice* maintient la thèse courante. Finalement, le *raisonneur* s'occupe des tâches de preuve particulières en s'appuyant sur le *démonstrateur* automatique externe.

Nous considérons les techniques du raisonneur en détail. Les procédures actuellement implantées sont les suivantes : décomposer des buts conjonctifs, les simplifier à l'aide de propriétés locales accumulées, filtrer et «alléger» des prémisses selon des suggestions dans le texte, appliquer des définitions.

Nous montrons, en utilisant les résultats du chapitre 2, que toutes les transformations de formules effectuées dans SAD sont cohérentes et qu'elles préservent les propriétés locales des occurrences particulières.

Le quatrième chapitre développe l'approche de la démonstration automatique associée au projet «Evidence Algorithm» [14, 15]. Nous démontrons la relation entre un calcul orienté but qui a été développé dans les travaux précédents dans le cadre de ce projet et la méthode de «tableaux de connexions» aussi connue sous le nom d'«élimination de modèles». En particulier, nous lions la notion de substitution admissible [43] à la technique traditionnelle de skolémisation.

Ensuite nous proposons une variante des tableaux de connexions cohérente et complète en logique classique du premier ordre avec égalité. Cette tâche n'est pas triviale car la règle habituelle de paramodulation intégrée aux tableaux de connexions produit un calcul qui est incomplet en absence des axiomes de réflexivité fonctionnelle. Or, l'utilisation de ces axiomes rendrait la recherche de preuve désespérément inefficace.

Notre calcul **LPCT** [58] emploie la paramodulation paresseuse [22, 66] où les contraintes d'unification de termes sont traitées comme les sous-buts à démontrer. Cette technique n'a pas été appliquée auparavant dans le cadre des calculs des tableaux. Outre cela, **LPCT** utilise les paramodulations basiques et les contraintes d'ordre, ce qui est aussi inhabituel dans les calculs orientés but.

Chapitre 1

Langage ForTheL

1.1 Introduction

ForTheL, un acronyme pour «Formal Theory Language», est un langage formel de textes mathématiques, qui imite la langue anglaise utilisée par les mathématiciens. Nous voyons deux raisons principales de poursuivre le style naturel «verbeux» contre une notation concise d'un langage traditionnel de logique.

Premièrement, un texte composé avec des phrases correctes de la langue naturelle est probablement plus lisible qu'une collection de formules construites des quantificateurs, parenthèses, connecteurs logiques, lambdas, et caetera. Par notre expérience, tel texte est aussi plus agréable à écrire. Alors, la première raison est donner à notre système une interface «amicale».

Deuxièmement, nous observons dans le texte humain beaucoup d'information en dehors de la logique qui disparaît habituellement dans la traduction. Dans un discours naturel, il y a des noms communs qui désignent des classes d'entités ; des adjectifs et des verbes qui agissent comme les attributs et restreignent les classes ; des adjectifs et des verbes qui agissent comme prédicats et peuvent lier les entités. Dans un texte mathématique traditionnel, il y a des définitions et des axiomes, des théorèmes importants et des lemmes auxiliaires, il y a des schémas divers de raisonnement. Mais là où la langue humaine fait des distinctions, le langage de la logique formelle unifie : les classes, les attributs, les prédicats — sont tous traduits par les symboles de prédicats ; les axiomes, les définitions, les théorèmes — deviennent tous des formules prémisses ; l'analyse de cas, *reductio ad absurdum*, l'application de définitions et de lemmes — sont tous transformés en inférences par *modus ponens*, par la résolution ou par les règles de Beth.

Nous sommes convaincus que le choix des fragments de raisonnement à mettre dans les lemmes séparés, le choix des nouvelles notions à introduire par les définitions, le choix du schéma de preuve et le reste, tout ce qui donne la structure au discours mathématique — porte des connaissances importantes sur le problème en question et doit être pris en compte avec le contenu purement logique. Notre intention est de préserver les distinctions, la structure fine du texte naturel, dans la formalisation. Nous voulons comprendre comment notre raison se sert de cette structure fine : comment nous traitons les définitions différemment des axiomes ordinaires, les noms des classes différemment des adjectifs, les démonstrations par analyse de cas différemment des démonstrations par induction, et ainsi de suite. Sur la base de telles observations, nous voulons améliorer les capacités déductives de l'ordinateur, implanter des procédures heuristiques qui vont diriger la recherche de démonstration ou restreindre l'espace de recherche en s'appuyant sur les connaissances extraites d'un texte formel mais «anthropomorphe».

Le langage ForTheL peut être considéré comme un langage naturel contrôlé. Il a beaucoup en commun avec Attempto Controlled English (ACE) [20, 21] (bien que la sémantique de phrase est différente dans ACE) et surtout avec Common Logic Controlled English (CLCE) [67]. En effet, beaucoup de propositions en ForTheL sont des propositions bien-formées en CLCE, ayant le même sens, et vice versa. Cependant, le concept de texte, qui embrasse le raisonnement dans le style naturel, est bien plus élaboré en ForTheL. Au niveau du texte, ForTheL ressemble aux langages des systèmes Mizar [69] et Isar [77].

Une autre approche à la formalisation de la langue mathématique commune est présentée

dans Mathematical Vernacular par de Bruijn [54] et son développement, Weak Type Theory (WTT) [34]. Il faut noter que les notions («noms») et attributs («adjectifs») en WTT sont considérés comme des types faibles et participent dans les inférences des types.

Dans les sections suivantes nous décrivons ForTheL, sa syntaxe et sa sémantique. Notre exposé va de bas en haut par rapport à la structure du langage :

1. Nous commençons par le niveau lexique qui détermine comment une séquence des caractères ASCII se transforme en une chaîne des *lexèmes* de ForTheL.
2. Au niveau suivant nous introduisons les *unités* : termes, notions, prédicats, propositions. Les unités sont composées de *primitives syntaxiques* (prédéfinies dans le langage ou introduites dans le texte en ForTheL) conformément aux règles de la grammaire de ForTheL. La sémantique d'une proposition de ForTheL est définie en termes du langage du premier ordre.
3. Ensuite nous considérons les *sections* de ForTheL. La section primaire (c'est-à-dire, non-composée) est la *phrase*, qui n'est, effectivement, qu'une proposition enveloppée. Des chaînes des phrases forment les démonstrations et les sections haut-niveau, telles que axiomes, définitions, extensions de signature et théorèmes. La sémantique d'une section est définie par un nombre des propriétés caractéristiques : son genre, la traduction dans le premier ordre, l'ensemble des variables déclarées, etc. Ces propriétés sont déterminées par le contenu de la section ainsi que par son entourage : l'ensemble des sections qui la précèdent logiquement dans le texte.
4. Enfin, un *texte* en ForTheL est une chaîne de sections haut-niveau. La sémantique d'un texte dépend de la tâche que l'on veut effectuer avec lui. Notre travail se concentre sur la vérification mathématique, et la notion correspondante d'un texte correct sera introduite et étudiée dans le chapitre suivant.

1.2 Structure lexicale

Nous utilisons les formes de Backus-Naur pour présenter la grammaire. Les non-terminaux sont écrit en italiques (e.g. *attribute*) et les terminaux en caractères typographiques (e.g. *therefore*). Les règles sont de la forme :

$$nonterm \;\rightarrow\; alt_1 \,|\, alt_2 \,|\, \ldots \,|\, alt_n$$

et les conventions suivantes sont adoptées :

$pat_1 \,|\, pat_2$	choix	$(\,patron\,)$	groupement
$[\,patron\,]$	facultatif	$\{\,patron\,\}$	répétitions, zéro ou plus

La structure lexique de ForTheL est défini comme suit :

$$\begin{aligned}
lexeme &\rightarrow mot \mid symbole \\
mot &\rightarrow alphanum\,\{\,alphanum\,\} \\
alphanum &\rightarrow alpha \mid numerique \mid \texttt{_} \\
alpha &\rightarrow minuscule \mid majuscule \\
minuscule &\rightarrow \texttt{a} \mid \ldots \mid \texttt{z} \\
majuscule &\rightarrow \texttt{A} \mid \ldots \mid \texttt{Z} \\
numerique &\rightarrow \texttt{0} \mid \ldots \mid \texttt{9} \\
symbole &\rightarrow \texttt{(} \mid \texttt{)} \mid \texttt{[} \mid \texttt{]} \mid \texttt{\{} \mid \texttt{\}} \mid \texttt{<} \mid \texttt{>} \mid \texttt{`} \mid \texttt{'} \mid \texttt{"} \mid \texttt{/} \mid \texttt{\textbackslash} \mid \texttt{|} \mid \texttt{\%} \\
&\quad \mid \texttt{!} \mid \texttt{?} \mid \texttt{@} \mid \texttt{\$} \mid \texttt{\&} \mid \texttt{\%} \mid \texttt{\~{}} \mid \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{=} \mid \texttt{:} \mid \texttt{;} \mid \texttt{,} \mid \texttt{.} \\
espaceblanc &\rightarrow blanc\,\{\,blanc\,\} \\
blanc &\rightarrow espace \mid finligne \mid commentaire \\
commentaire &\rightarrow \texttt{\#}\,\{\text{tout caractère sauf fin de ligne}\}\,finligne \\
espace &\rightarrow \text{espace} \mid \text{tabulation} \\
finligne &\rightarrow \text{fin de ligne}
\end{aligned}$$

Dans l'analyse lexicale, les lexèmes sont pris aussi longs que possible selon les règles de la grammaire. En particulier, un lexème *mot* n'est jamais suivi par un autre lexème *mot* sans être séparé par un lexème *symbole* ou *espaceblanc*.

L'espace blanc sert à délimiter des autres lexèmes et n'intervient pas dans les règles suivantes. Il y a un cas d'exception où la présence ou absence de l'espace blanc est prise en compte : l'analyse des primitives symboliques (la section 1.3.1) qui contiennent un *token*[1] composé de plusieurs symboles, tel que `->` ou `!=`. D'après les règles ci-dessus, le deux lignes se composent de deux lexèmes consécutifs. Pourtant, les mêmes lexèmes séparés par un espace : `- >` ne seront pas reconnus comme le token `->`.

Les variables en ForTheL sont dénotées par des lettres latines.

$$variable \rightarrow alpha$$

La case est importante : `x` et `X` sont deux variables différentes.

1.3 Syntaxe des propositions

Pour nous initier aux propositions de ForTheL, commençons par quelques exemples. Pour chaque proposition nous donnons son sens dans le langage du premier ordre introduit dans la section 2.2.1.

ForTheL : `every bird is a feathery animal`
Sens : $\forall\, x\, (x \,\varepsilon\, \mathrm{Bird} \supset (x \,\varepsilon\, \mathrm{Animal} \wedge \mathrm{isFeathery}(x)))$

ForTheL : `every subset of some set S is equal to S`
Sens : $\exists\, S\, (S \,\varepsilon\, \mathrm{Set} \wedge \forall\, x\, (x \,\varepsilon\, \mathrm{SubsetOf}(S) \supset x = S))$

ForTheL : `every natural number m greater than 0 divides m !`
Sens : $\forall\, m\, ((m \,\varepsilon\, \mathrm{Number} \wedge m \,\varepsilon\, \mathrm{Natural} \wedge \wedge\, \mathrm{isGreaterThan}(m, \mathrm{Zero})) \supset \mathrm{Divides}(m, \mathrm{Factorial}(m)))$

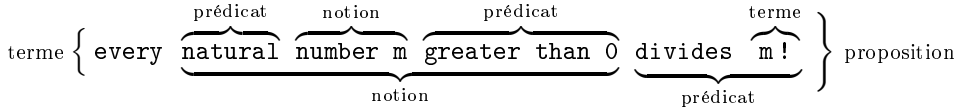
ForTheL : `if X and Y are close to Z then X and Y are close`
Sens : $(\mathrm{isCloseTo}(X, Z) \wedge \mathrm{isCloseTo}(Y, Z)) \supset \mathrm{isCloseTo}(X, Y)$

ForTheL : `no equation in G has a positive solution`
Sens : $\forall\, x\, ((x \,\varepsilon\, \mathrm{Equation} \wedge \mathrm{isIn}(x, G)) \supset \supset \neg\exists\, y\, (y \,\varepsilon\, \mathrm{SolutionOf}(x) \wedge \mathrm{isPositive}(y)))$

Les propositions de ForTheL sont composées des *unités* qui sont de quatre genres principaux :

- *notion* dénote une classe d'objets, peut-être paramétrée : `natural number`, `element of S`, `series that converges to N`.
- *terme* dénote un objet, soit en indiquant une valeur concrète : `N`, `the complement of S`, `X * Y`; soit en quantifiant la classe désignée par une notion : `every set`, `some divisor of M`.
- *prédicat* dénote une propriété d'un objet : `empty`, `divides N`, `is a subset of S`. Appliqué à un terme, le prédicat forme une proposition ; appliqué à une notion, il forme une autre notion, plus restreinte.
- *proposition* dénote une expression logique qui peut être vraie ou fausse : `X > Y`, `every divisor of N is a natural number`, `there exists a countable set`.

La troisième proposition ci-dessus se décompose comme suit :



[1] nous désignons par ce terme les composantes des primitives syntaxiques. D'habitude, un token s'exprime avec un lexème, mais des tokens symboliques peuvent en contenir plusieurs.

Comme toute langue naturelle, ForTheL admet des expressions ambiguës. Par exemple, dans la proposition :

```
some point of any straight line that crosses L lies on L
```

le prédicat `crosses L` peut être appliqué à «some point», comme à «`any straight line`». Sans certaines capacités de raisonnement, l'analyseur syntaxique n'a aucun moyen pour choisir la lecture correcte. Aussi, telles unités sont rejetées. Généralement, il n'est pas difficile de trouver une reformulation non-ambiguë :

```
every straight line that crosses L has a point that lies on L
```

et, en tout cas, on peut recourir aux parenthèses :

```
some point of (any straight line that crosses L) lies on L
```

1.3.1 Primitives syntaxiques

Les *primitives syntaxiques* sont les briques des unités. À chaque moment de l'analyse syntaxique nous disposons d'une collection des primitives qui réfléchit la signature de la portion du texte qu'on à lu pour le moment. Nous appelons cette collection le *vocabulaire courant* pour indiquer qu'elle est propre au texte considéré et qu'elle est dynamique. Il y a six groupes des primitives syntaxiques, appelés *primitives de base* :

- *noms-classes*, pour former les notions : `element of` arg_1
- *noms-objets*, pour former les termes : `zero`, `power set of` arg_1
- *adjectifs* et *verbes*, pour former les prédicats : `converges`, `equal to` arg_1
- *opérateurs* symboliques, y compris les constantes : `0`, arg_1 `+` arg_2, `min` arg_1
- *relations* symboliques : arg_1 `<=` arg_2, arg_1 `:` arg_2 `->` arg_3

Les primitives de base sont introduites directement dans le texte à l'aide de définitions, extensions de signature et déclarations de synonymes. La syntaxe de ces formes est décrite dans la section 1.3.6. Il y a aussi des groupes de primitives supplémentaires qui sont dérivées automatiquement des primitives de base. Par exemple, les primitives-noms avec une place d'argument après `of`, tel que `subset of` arg_1 ou `complement of` arg_1 `to` arg_2 produisent des primitives spéciales à employer dans les prédicats possessifs : `has an element`, `of an infinite cardinality`. Les primitives dérivées seront introduites, groupe par groupe, dans les sections suivantes.

Pour introduire une primitive de base, l'auteur doit donner un *patron* qui détermine la syntaxe. Un patron est une chaîne de *tokens* (qui définissent les terminaux de nouvelle primitive) alternés par des variables (qui indiquent les places d'arguments) :

$$
\begin{aligned}
\mathit{patron} &\rightarrow \mathit{token}\ \{\,\mathit{token}\,\}\ [\,\mathit{variable}\ \{\,\mathit{token}\ \{\,\mathit{token}\,\}\ \mathit{variable}\,\}\,] \\
\mathit{token} &\rightarrow \mathit{minuscule}\ \{\,\mathit{minuscule}\,\} \\
\mathit{symbPatron} &\rightarrow [\,\mathit{variable}\,]\ \mathit{symbToken}\ \{\,\mathit{variable}\ \mathit{symbToken}\,\}\ [\,\mathit{variable}\,] \\
&\quad \mid \mathit{mot}\ \texttt{(}\ \mathit{variable}\ \{\,\texttt{,}\ \mathit{variable}\,\}\ \texttt{)} \\
&\quad \mid \mathit{mot}\ [\,\mathit{variable}\,] \\
\mathit{symbToken} &\rightarrow \mathit{symbole}\ \{\,\mathit{symbole}\,\}
\end{aligned}
$$

Les tokens (ordinaires et symboliques) ne peuvent pas contenir des espaces. Les articles `a`, `an`, `the`, les formes du verbe «to be» et les mot à une lettre ne sont pas acceptés comme tokens.

Comme noms et verbes prennent en anglais des formes différentes au singulier et au pluriel, ForTheL permet d'«égaliser» des tokens en les joignant dans un *groupe de tokens* :

$$\mathit{groupeTokens} \rightarrow \texttt{[}\ \mathit{token}\ \texttt{/}\ \mathit{token}\ \{\,\texttt{/}\ \mathit{token}\,\}\ \texttt{]}$$

Par exemple, on peut introduire les groupes suivants :

```
[formula/formulas/formulae]
[cross/crosses]
```

Un groupe de tokens doit apparaître dans le texte avant le patron qui utilise un token de ce groupe.

En convertissant un patron en une primitive syntaxique, l'analyseur syntaxique transforme les (groupes de) tokens en (choix groupés de) terminaux et remplace les variables par le non-terminal *terme* (*symbTerme*, dans les patrons symboliques) qui «lit» les arguments de la primitive. Dans les patrons de relations qui commencent avec par variable, cette variable est remplacée par le non-terminal *symbTermes*, pour permettre des expressions comme «`x,y <= z`». Dans les noms-classes, le non-terminal facultatif *noms* (voir la section 1.3.2) est inséré un token plus tôt que la première place d'argument ou à la fin du patron, s'il n'y a pas d'arguments. Considérons les exemples suivants de patrons avec les primitives correspondantes, une ligne pour un groupe de base. Nous présumons que les groupes de tokens nécessaires sont déjà introduits.

Patron	Primitive
`subset of S`	(`subset` | `subsets`) [*noms*] (`of`) *terme*
`power set of S`	(`power`) (`set` | `sets`) (`of`) *terme*
`infinite`	(`infinite`)
`divides n`	(`divides` | `divide`) *terme*
`Pow x`	(`Pow`) *symbTerme*
`x <= y`	*symbTermes* (`<=`) *symbTerme*

Dans les primitives non-symboliques, la case des tokens est ignorée. C'est-à-dire que `subset of S`, `Subset of S`, et `sUbSeT oF S` sont tous la même unité notion. Les tokens des primitives symboliques, au contraire, sont sensibles à la case : `sin x`, `Sin x`, `sin X` sont trois termes différents.

Les règles de la syntaxe responsables de l'analyse des primitives sont les seules qui sont modifiées au cours de la lecture. Dans les sections suivantes nous présentons ces règles comme des «instantanés» hypothétiques du vocabulaire courant dans un moment donné de la lecture.

1.3.2 Notions

Toute unité notion est construite à partir d'une *notion primaire* à laquelle un certain nombre *d'attributs* est appliqué. Deux genres de primitives syntaxiques sont employés pour former des notions primaires :

$$
\begin{aligned}
\mathit{primNomClasse} \rightarrow\ & (\ \texttt{set} \mid \texttt{sets}\ )\ [\ \mathit{noms}\ ] \\
\mid\ & (\ \texttt{element} \mid \texttt{elements}\ )\ [\ \mathit{noms}\ ]\ (\ \texttt{of}\ )\ \mathit{terme} \\
\mid\ & (\ \texttt{function} \mid \texttt{functions}\ )\ [\ \mathit{noms}\ ]\ (\ \texttt{from}\ )\ \mathit{terme}\ (\ \texttt{to}\ )\ \mathit{terme} \\
\mid\ & \ldots \\
\\
\mathit{primRelationClasse} \rightarrow\ & \mathit{noms}\ (\ \texttt{<<}\ )\ \mathit{symbTerme} \\
\mid\ & \mathit{noms}\ (\ \texttt{:}\ )\ \mathit{symbTerme}\ (\ \texttt{->}\ )\ \mathit{symbTerme} \\
\mid\ & \ldots \\
\\
\mathit{noms} \rightarrow\ & \mathit{variable}\ \{\ \texttt{,}\ \mathit{variable}\ \}
\end{aligned}
$$

Relations-classes sont les primitives dérivées. Elles sont générées d'une façon automatique à partir des relations descriptives (voir Sections 1.3.6 et 1.3.7). La première place d'argument dans une relation-classe est prise par le non-terminal *noms*, et le reste est le même que dans la relation correspondante. Les relations-classes ci-dessus pourraient être produites par les déclarations de synonymes comme suit :

```
Let x << y stand for (x is an element of y).
Let f : D -> R denote (f is a function from D to R).
```

Chaque unité notion est caractérisée par une liste de *noms* (à ne pas confondre avec des noms — primitives syntaxiques). Ces identificateurs se réfèrent aux objets particuliers pris dans la classe. Ils jouent le même rôle que les noms des variables attachés à un quantificateur. Nous avons vu les noms employés dans la proposition :

```
every natural number m greater than 0 divides m !
```

Les constructions avec plus qu'un nom sont aussi utiles :

```
for all real numbers X,Y (X*Y) is a real number
```

Dans les noms-classes, la liste de noms peut être vide :

```
every even natural number greater than 2 is compound
L,M are parallel straight lines
```

Les *attributs* servent à restreindre la classe désignée par une notion. Du point de vue syntaxique, les attributs sont basés sur les unités prédicats et unités propositions :

$$\textit{attributGauche} \rightarrow \textit{primAdjectifSimple} \mid \textit{primAdjectifSimpleM}$$

$$\begin{aligned} \textit{attributDroit} \rightarrow\ & \textit{predicatIs}\ \{\,\texttt{and}\ \textit{predicatIs}\,\} \\ \mid\ & \texttt{that}\ \textit{predicatDoes}\ \{\,\texttt{and}\ \textit{predicatDoes}\,\} \\ \mid\ & \texttt{such that}\ \textit{proposition} \end{aligned}$$

Les adjectifs simples qui forment des attributs gauches sont aussi des primitives dérivées. Ils sont produits par les adjectifs et m-adjectifs (voir la section 1.3.4) qui n'ont pas d'arguments :

$$\textit{primAdjectifSimple} \rightarrow (\,\texttt{prime}\,) \mid (\,\texttt{empty}\,) \mid (\,\texttt{natural}\,) \mid \ldots$$

$$\textit{primAdjectifSimpleM} \rightarrow (\,\texttt{equal}\,) \mid (\,\texttt{parallel}\,) \mid (\,\texttt{disjoint}\,) \mid \ldots$$

Maintenant, définissons la syntaxe des notions :

$$\textit{notion} \rightarrow \textit{nomClasse} \mid \textit{relationClasse}$$

$$\textit{nomClasse} \rightarrow \{\,\textit{attributGauche}\,\}\ \textit{primNomClasse}\ [\,\textit{attributDroit}\,]$$

$$\textit{relationClasse} \rightarrow \{\,\textit{attributGauche}\,\}\ [\,\texttt{(}\,]\ \textit{primRelationClasse}\ [\,\texttt{)}\,]\ [\,\textit{attributDroit}\,]$$

Les expressions suivantes sont des notions bien formées :

```
cyclic group G
injective f : Nat -> Nat that maps 0 to 0
real number greater than 0 and less than 1
natural numbers q,r such that n = (q * m) + r and r < m
```

Notez que la dernière unité définit en fait deux classes différentes : une pour `q` et une pour `r`.

1.3.3 Termes

Il y a deux genres d'unités termes en ForTheL : termes fixes et notions quantifiées :

$$\begin{aligned} \textit{terme} \rightarrow\ & [\,\texttt{(}\,]\ \textit{notionQuantifiee}\ [\,\texttt{)}\,] \\ \mid\ & \textit{termeFixe} \end{aligned}$$

Les *notions quantifiées* permettent de formuler des propositions génériques sur tout objet de la classe désignée par une notion, ou d'affirmer qu'il y a un objet dans la classe qui possède une certaine propriété :

$$\begin{aligned} \textit{notionQuantifiee} \rightarrow\ & (\,\texttt{every} \mid \texttt{each} \mid \texttt{all} \mid \texttt{any}\,)\ \textit{notion} \\ \mid\ & \texttt{some}\ \textit{notion} \\ \mid\ & \texttt{no}\ \textit{notion} \end{aligned}$$

Remarque. Les notions quantifiées intervenant comme termes-arguments dans les primitives syntaxiques doivent avoir un nom au maximum. Ainsi, les unités `N divides some numbers X,Y,Z`, `a subset of finite sets A,B`, `the orders of groups G,H` sont interdites.

Un *terme fixe* est formé à partir d'un nom-objet ou d'un opérateur symbolique, par application d'une fonction aux termes-arguments. Nous adoptons les conventions suivantes sur la priorité et l'associativité des opérateurs :

- les opérateurs *postfixes*, dont les primitives finissent par un token, ont la priorité la plus haute ;
- les opérateurs *préfixes*, dont les primitives commencent par un token et finissent par une place d'argument, ont la priorité moyenne ;
- les opérateurs *infixes*, dont les primitives commencent et finissent par une place d'argument, ont la priorité la plus basse et associent à droite.

On peut se servir des parenthèses pour réordonner une expression symbolique.

$$\begin{aligned}
\mathit{termeFixe} &\rightarrow [\,\texttt{(}\,][\,\texttt{the}\,]\ \mathit{primNomObjet}\ [\,\texttt{)}\,] \\
&\mid\ \mathit{symbTerme}
\end{aligned}$$

$$\mathit{symbTerme} \rightarrow \mathit{primOperateurInfixe} \mid \mathit{symbTermePrefixe}$$

$$\mathit{symbTermePrefixe} \rightarrow \mathit{primOperateurPrefixe} \mid \mathit{symbTermePostfixe}$$

$$\mathit{symbTermePostfixe} \rightarrow \mathit{primOperateurPostfixe} \mid \texttt{(}\ \mathit{symbTerme}\ \texttt{)} \mid \mathit{variable}$$

Voici les primitives typiques des nom-objets et opérateurs :

$$\begin{aligned}
\mathit{primNomObjet} &\rightarrow (\,\texttt{zero}\,|\,\texttt{zeroes}\,) \\
&\mid\ (\,\texttt{order}\,|\,\texttt{orders}\,)\ (\,\texttt{of}\,)\ \mathit{terme} \\
&\mid\ \ldots
\end{aligned}$$

$$\begin{aligned}
\mathit{primOperateurInfixe} &\rightarrow \mathit{symbTermePrefixe}\ (\,\texttt{*}\,)\ \mathit{symbTerme} \\
&\mid\ \ldots
\end{aligned}$$

$$\begin{aligned}
\mathit{primOperateurPrefixe} &\rightarrow (\,\texttt{min}\,)\ \mathit{symbTermePrefixe} \\
&\mid\ \ldots
\end{aligned}$$

$$\begin{aligned}
\mathit{primOperateurPostfixe} &\rightarrow (\,0\,) \\
&\mid\ (\,\texttt{exp}\,)\ (\,\texttt{(}\,)\ \mathit{symbTerme}\ (\,\texttt{,}\,)\ \mathit{symbTerme}\ (\,\texttt{)}\,) \\
&\mid\ \mathit{symbTermePostfixe}\ (\,\texttt{(}\,)\ \mathit{symbTerme}\ (\,\texttt{)}\,) \\
&\mid\ \ldots
\end{aligned}$$

Notez la dernière primitive dans le groupe d'opérateurs postfixes. Il introduit l'application fonctionnelle comme opération binaire abstraite en utilisant la même syntaxe que pour les fonctions habituelles, telles que `exp`.

Un *terme primordial* est un terme qui ne contient pas de notions quantifiées. Autrement dit, un terme primordial est un terme fixe dont les arguments sont aussi des termes fixes :

$$\begin{aligned}
\mathit{termePrimord} &\rightarrow [\,\texttt{(}\,][\,\texttt{the}\,]\ \mathit{primNomPrimord}\ [\,\texttt{)}\,] \\
&\mid\ \mathit{symbTerme}
\end{aligned}$$

$$\begin{aligned}
\mathit{primNomPrimord} &\rightarrow (\,\texttt{zero}\,|\,\texttt{zeroes}\,) \\
&\mid\ (\,\texttt{order}\,|\,\texttt{orders}\,)\ (\,\texttt{of}\,)\ \mathit{termePrimord} \\
&\mid\ \ldots
\end{aligned}$$

Chaque nom-objet produit automatiquement une primitive jumelle *primNomPrimord* en remplaçant dans les places d'arguments le non-terminal *terme* par *termePrimord*.

1.3.4 Prédicats

Les termes nous donnent des objets ; les *prédicats* expriment leurs propriétés et relations qui les lient. Il y a trois genres de *prédicats primaires* (c.-à-d. non-composés) : ceux construits avec une primitive de verbe ou d'adjectif, ceux qui servent à exprimer l'appartenance à la classe désignée par une notion («is a»-prédicats), et ceux qui servent à exprimer l'existence d'un certain objet subordonné au sujet («has»-prédicats). Les prédicats primaires peuvent être mis sous négation et composés dans les conjonctions :

$$
\begin{aligned}
\mathit{predicatDoes} \rightarrow\ & [\,\texttt{does} \mid \texttt{do}\,]\ [\,\texttt{not}\,]\ \mathit{primVerbe} \\
\mid\ & [\,\texttt{does} \mid \texttt{do}\,]\ [\,\texttt{not}\,]\ [\,\texttt{pairwise}\,]\ \mathit{primVerbeM} \\
\mid\ & (\,\texttt{has} \mid \texttt{have}\,)\ \mathit{predicatHas} \\
\mid\ & (\,\texttt{is} \mid \texttt{are} \mid \texttt{be}\,)\ \mathit{predicatIs}\ \{\,\texttt{and}\ \mathit{predicatIs}\,\} \\
\mid\ & (\,\texttt{is} \mid \texttt{are} \mid \texttt{be}\,)\ \mathit{predicatIsA}\ \{\,\texttt{and}\ \mathit{predicatIsA}\,\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{predicatIs} \rightarrow\ & [\,\texttt{not}\,]\ \mathit{primAdjectif} \\
\mid\ & [\,\texttt{not}\,]\ [\,\texttt{pairwise}\,]\ \mathit{primAdjectifM} \\
\mid\ & (\,\texttt{with} \mid \texttt{of} \mid \texttt{having}\,)\ \mathit{predicatHas}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{predicatIsA} \rightarrow\ & [\,\texttt{not}\,]\ [\,\texttt{a} \mid \texttt{an}\,]\ \mathit{nomClasse} \\
\mid\ & [\,\texttt{not}\,]\ \mathit{termeFixe}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{predicatHas} \rightarrow\ & [\,\texttt{a} \mid \texttt{an} \mid \texttt{the}\,]\ \mathit{nomPossede}\ \{\,\texttt{and}\ [\,\texttt{a} \mid \texttt{an} \mid \texttt{the}\,]\ \mathit{nomPossede}\,\} \\
\mid\ & \texttt{no}\ \mathit{nomPossede}
\end{aligned}
$$

Les adjectifs et verbes primitifs sont comme suit :

$$
\begin{aligned}
\mathit{primVerbe} \rightarrow\ & (\,\texttt{converges} \mid \texttt{converge}\,) \\
\mid\ & (\,\texttt{divides} \mid \texttt{divide}\,)\ \mathit{terme} \\
\mid\ & (\,\texttt{belongs} \mid \texttt{belong}\,)\ (\,\texttt{to}\,)\ \mathit{terme} \\
\mid\ & (\,\texttt{joins} \mid \texttt{join}\,)\ \mathit{terme}\ (\,\texttt{with}\,)\ \mathit{terme} \\
\mid\ & \ldots
\end{aligned}
$$

$$
\begin{aligned}
\mathit{primAdjectif} \rightarrow\ & (\,\texttt{prime}\,) \\
\mid\ & (\,\texttt{dividing}\,)\ \mathit{terme} \\
\mid\ & (\,\texttt{equal}\,)\ (\,\texttt{to}\,)\ \mathit{terme} \\
\mid\ & (\,\texttt{less}\,)\ (\,\texttt{than}\,)\ \mathit{terme} \\
\mid\ & \ldots
\end{aligned}
$$

L'adjectif primitif `equal to` est prédéfini dans ForTheL. Notez que nous pouvons utiliser le présent continu d'un verbe primitif en le déguisant en adjectif. Cela doit être fait explicitement dans le texte, avec une déclaration de synonyme, telle que «`Let x is dividing y stand for (x divides y).`», car le système ne pourrait pas déterminer de lui-même la forme correcte du verbe.

Verbes et adjectifs primitifs peuvent produire automatiquement des *multisujets primitives* ou *m-primitives*. La règle de production est la suivante. Prenons une primitive de $\mathit{primVerbe}$ ou $\mathit{primAdjectif}$ ayant au moins un argument. Si la première place d'argument est précédée par une préposition et n'est pas suivie par le token (`and`), nous enlevons alors cette première place d'argument avec le token précédent et nous mettons la nouvelle primitive dans $\mathit{primVerbeM}$ (respectivement dans $\mathit{primAdjectifM}$).

Ainsi, l'adjectif primitif :

$$(\,\texttt{parallel}\,)\ (\,\texttt{to}\,)\ \mathit{terme}$$

produit le m-adjectif :

$$(\,\texttt{parallel}\,)$$

qui s'ajoute dans *primAdjectifM* et *primAdjectifSimpleM* ; et le verbe primitif :

(`commutes` | `commute`) (`with`) *terme* (`wrt`) *terme*

produit le m-verbe :

(`commutes` | `commute`) (`wrt`) *terme*

Remarque. Les unités-prédicats qui sont construites de m-primitives (*m-prédicats*) doivent être employées avec plusieurs sujets : `X,Y,Z commute wrt N`, `parallel lines l,m`. Si un tel prédicat est employé dans un attribut, l'unité-notion ne peut pas avoir un seul nom. Ainsi, les unités `an equal number X`, `a line N that is parallel`, `a set S such that S is disjoint` sont interdites.

Les nouvelles m-primitives sont produites pour tous les adjectifs et verbes primitifs. La symétrie et la transitivité des prédicats multisujets sont présupposées. Ainsi, la proposition `A,B,C are equal` sera traduite dans la formule `A is equal to B and B is equal to C`. Pour les prédicats non-transitifs, l'adverbe special `pairwise` doit être employé. Alors la proposition `A,B,C are pairwise disjoint` sera correctement traduite comme `A is disjoint with B and A is disjoint with C and B is disjoint with C`.

$$
\begin{aligned}
\mathit{primVerbeM} \rightarrow\ & (\ \texttt{collides} \mid \texttt{collide}\ ) \\
\mid\ & (\ \texttt{commutes} \mid \texttt{commute}\ )\ (\ \texttt{wrt}\ )\ \mathit{terme} \\
\mid\ & \ldots \\
\\
\mathit{primAdjectifM} \rightarrow\ & (\ \texttt{equal}\ ) \\
\mid\ & (\ \texttt{adjacent}\ )\ (\ \texttt{in}\ )\ \mathit{terme} \\
\mid\ & \ldots
\end{aligned}
$$

À l'aide de *«is a»-prédicats* nous pouvons exprimer qu'un objet est soit «décrit» par une notion ou est égal à un terme fixe : `X is a natural number`, `X is the order of G`.

Remarque. Les nom-classes dans les «is a»-prédicats doivent avoir un nom au maximum.

Pour les *«has»-prédicats*, nous maintenons encore un groupe des primitives dérivées des nom-classes et nom-objets :

$$
\begin{aligned}
\mathit{nomPossede} \rightarrow\ & \{\ \mathit{attributGauche}\ \}\ \mathit{primNomPossede}\ [\ \mathit{attributDroit}\ ] \\
\\
\mathit{primNomPossede} \rightarrow\ & (\ \texttt{element} \mid \texttt{elements}\ )\ [\ \mathit{noms}\ ] \\
\mid\ & (\ \texttt{solution} \mid \texttt{solutions}\ )\ [\ \mathit{noms}\ ] \\
\mid\ & (\ \texttt{order} \mid \texttt{orders}\ )\ [\ \mathit{noms}\ ] \\
\mid\ & \ldots
\end{aligned}
$$

Ces primitives sont produites automatiquement par la règle suivante. Prenons un nom primitif dans *primNomClasse* ou *primNomObjet* avec au moins un argument. Si la première place d'argument est précédée par la préposition (`of`) et n'est pas suivie par le token (`and`), nous enlevons alors cette première place d'argument avec le token précédent, nous ajoutons [*noms*] si nécessaire, et nous mettons la nouvelle primitive dans *primNomPossede*.

Alors le nom-classe :

(`ambassador` | `ambassadors`) [*noms*] (`of`) *terme* (`in`) *terme*

produit le nouveau nom possédé :

(`ambassador` | `ambassadors`) [*noms*] (`in`) *terme*

tandis que le nom-objet `union of _ and _` ne produit aucune nouvelle primitive à cause du token (`and`) après la première place d'argument.

Voici quelques exemples de propositions avec «has»-prédicats :

```
X has no elements
every set of a finite cardinality is finite
F has the domain D and the range R such that D is a subset of R
```

Dans la deuxième proposition le «has»-prédicat apparaît comme attribut d'une notion.

1.3.5 Propositions

Les *propositions* sont des homologues des formules de la logique. Considérons d'abord les propositions *primaires*, non-composées. Une telle proposition peut être d'une de quatre formes :

$$\begin{aligned} \mathit{propositionPrimaire} \rightarrow\ & \mathit{propositionSimple} \\ \mid\ & \mathit{propositionThereIs} \\ \mid\ & [\,\texttt{we have}\,]\ \mathit{symbProposition} \\ \mid\ & [\,\texttt{we have}\,]\ \mathit{propositionConst} \end{aligned}$$

Les *propositions simples* appliquent prédicats aux termes :

$$\mathit{propositionSimple} \rightarrow \mathit{termes}\ \mathit{predicatDoes}\ \{\,\texttt{and}\ \mathit{predicatDoes}\,\}$$

$$\mathit{termes} \rightarrow \mathit{terme}\ \{\,(\,\texttt{,}\mid\texttt{and}\,)\ \mathit{terme}\,\}$$

Les *«there is»-propositions* affirment ou nient l'existence de membres dans la classe désignée par une notion.

$$\begin{aligned} \mathit{propositionThereIs} \rightarrow\ & \texttt{there}\ (\,\texttt{exists}\mid\texttt{exist}\,)\ \mathit{notions} \\ \mid\ & \texttt{there}\ (\,\texttt{exists}\mid\texttt{exist}\,)\ \texttt{no}\ \mathit{notion} \end{aligned}$$

$$\mathit{notions} \rightarrow [\,\texttt{a}\mid\texttt{an}\,]\ \mathit{notion}\ \{\,(\,\texttt{,}\mid\texttt{and}\,)\ [\,\texttt{a}\mid\texttt{an}\,]\ \mathit{notion}\,\}$$

Les *propositions symboliques* sont composées à partir de relations et termes symboliques dans la syntaxe traditionnelle du premier ordre. Les propositions non-symboliques peuvent y intervenir aussi, mises entre parenthèses. Dans la règle pour *symbProposition*, `<=>` dénote l'équivalence, `=>` l'implication, `\/` la disjonction, et `/\` la conjonction. Les règles pour ces connecteurs propositionnels sont données dans l'ordre d'augmentation de priorité.

$$\begin{aligned} \mathit{symbProposition} \rightarrow\ & \texttt{forall}\ \mathit{relationClasse}\ \mathit{symbProposition} \\ \mid\ & \texttt{exists}\ \mathit{relationClasse}\ \mathit{symbProposition} \\ \mid\ & \mathit{symbProposition}\ \texttt{<=>}\ \mathit{symbProposition} \\ \mid\ & \mathit{symbProposition}\ \texttt{=>}\ \mathit{symbProposition} \\ \mid\ & \mathit{symbProposition}\ \texttt{\textbackslash/}\ \mathit{symbProposition} \\ \mid\ & \mathit{symbProposition}\ \texttt{/\textbackslash}\ \mathit{symbProposition} \\ \mid\ & \texttt{not}\ \mathit{symbProposition} \\ \mid\ & \texttt{(}\ \mathit{proposition}\ \texttt{)} \\ \mid\ & \mathit{primRelation} \end{aligned}$$

$$\begin{aligned} \mathit{primRelation} \rightarrow\ & \mathit{symbTermes}\ (\,\texttt{=}\,)\ \mathit{symbTerme} \\ \mid\ & \mathit{symbTermes}\ (\,\texttt{!=}\,)\ \mathit{symbTerme} \\ \mid\ & \mathit{symbTermes}\ (\,\texttt{-<-}\,)\ \mathit{symbTerme} \\ \mid\ & \mathit{symbTermes}\ (\ \texttt{:}\,)\ \mathit{symbTerme}\ (\,\texttt{->}\,)\ \mathit{symbTerme} \\ \mid\ & (\,\texttt{Nat}\,)\ (\,\texttt{(}\,)\ \mathit{symbTerme}\ (\,\texttt{)}\,) \\ \mid\ & \ldots \end{aligned}$$

$$\mathit{symbTermes} \rightarrow \mathit{symbTerme}\ \{\,\texttt{,}\ \mathit{symbTerme}\,\}$$

Les relations binaires `=` et `!=` sont prédéfinies dans ForTheL. Elles sont les synonymes pour, respectivement, la proposition d'égalité et sa négation. La relation binaire `-<-` a aussi une sémantique spéciale : elle dénote une relation bien-fondée abstraite qui est utilisée dans les démonstrations par induction (voir la section 1.5.3). Cette relation, pourtant, doit être introduit explicitement dans le texte.

Les *proposition constantes* `thesis` et `contrary` dénotent, respectivement, la thèse courante et sa négation. La notion de thèse courante sera introduite dans la section 2.3.2. La proposition constante `contradiction` dénote $\bot$, la contradiction logique.

$$\begin{aligned} \mathit{propositionConst} \rightarrow\ & [\,\texttt{the}\,]\ \texttt{thesis} \\ |\ & [\,\texttt{the}\,]\ \texttt{contrary} \\ |\ & [\,\texttt{a}\,|\,\texttt{an}\,]\ \texttt{contradiction} \end{aligned}$$

Les propositions primaires sont composées avec des prépositions et conjonctions :

$$\mathit{proposition} \rightarrow \mathit{propositionTete} \mid \mathit{propositionChaine}$$

$$\begin{aligned} \mathit{propositionTete} \rightarrow\ & \texttt{for}\ \mathit{notionQuantifiee}\ \{\,\texttt{and}\ \mathit{notionQuantifiee}\,\}\ \mathit{proposition} \\ |\ & \texttt{if}\ \mathit{proposition}\ \texttt{then}\ \mathit{proposition} \\ |\ & \texttt{it is wrong that}\ \mathit{proposition} \end{aligned}$$

$$\begin{aligned} \mathit{propositionChaine} \rightarrow\ & \mathit{andChaine}\ [\,\texttt{and}\ \mathit{propositionTete}\,] \\ |\ & \mathit{orChaine}\ [\,\texttt{or}\ \mathit{propositionTete}\,] \\ |\ & (\,\mathit{andChaine}\,|\,\mathit{orChaine}\,)\ \texttt{iff}\ \mathit{proposition} \end{aligned}$$

$$\mathit{andChaine} \rightarrow \mathit{propositionPrimaire}\ \{\,\texttt{and}\ \mathit{propositionPrimaire}\,\}$$

$$\mathit{orChaine} \rightarrow \mathit{propositionPrimaire}\ \{\,\texttt{or}\ \mathit{propositionPrimaire}\,\}$$

1.3.6 Propositions spéciales. Synonymes

Les *propositions spéciales* sont employées dans les définitions et extensions de signature (voir la section 1.5.1). Leur syntaxe est décrite par les règles suivantes :

$$\mathit{defProposition} \rightarrow \mathit{notionDef} \mid \mathit{fonctionDef} \mid \mathit{predicatDef}$$

$$\mathit{sigProposition} \rightarrow \mathit{notionSig} \mid \mathit{fonctionSig} \mid \mathit{predicatSig}$$

$$\mathit{notionDef} \rightarrow \mathit{teteNotion}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \mathit{nomClasse}$$

$$\begin{aligned} \mathit{notionSig} \rightarrow\ & \mathit{teteNotion}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \mathit{nomClasse} \\ |\ & \mathit{teteNotion}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \texttt{notion} \end{aligned}$$

$$\begin{aligned} \mathit{fonctionDef} \rightarrow\ & \mathit{teteFonction}\ \texttt{is}\ [\,\texttt{equal to}\,]\ \mathit{termePrimord} \\ |\ & \mathit{teteFonction}\ \texttt{is}\ [\,\texttt{equal to}\,]\ [\,\texttt{a}\,|\,\texttt{an}\,|\,\texttt{the}\,]\ \mathit{nomClasse} \end{aligned}$$

$$\begin{aligned} \mathit{fonctionSig} \rightarrow\ & \mathit{teteFonction}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \mathit{nomClasse} \\ |\ & \mathit{teteFonction}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ (\,\texttt{term}\,|\,\texttt{constant}\,) \end{aligned}$$

$$\mathit{predicatDef} \rightarrow \mathit{tetePredicat}\ \texttt{iff}\ \mathit{proposition}$$

$$\begin{aligned} \mathit{predicatSig} \rightarrow\ & \mathit{tetePredicat}\ \texttt{implies}\ \mathit{proposition} \\ |\ & \mathit{tetePredicat}\ \texttt{is}\ [\,\texttt{a}\,|\,\texttt{an}\,]\ \texttt{atom} \end{aligned}$$

Dans toute proposition spéciale S, nous distinguons la *jonction principale* qui peut être le mot `is`, égalité, équivalence ou implication. L'unité à gauche de la jonction principale est appelée *l'unité de tête* de la proposition, de la phrase qui contient la proposition, et de la section (définition ou extension de signature) qui contient la phrase. L'unité à droite de la jonction principale est appelée *l'unité cible*. La syntaxe de l'unité de tête est comme suit :

$$
\begin{aligned}
\mathit{teteNotion} &\rightarrow [\,\texttt{a} \mid \texttt{an}\,]\ \mathit{primNomClasse} \\
&\mid \mathit{patronNotion} \\
\\
\mathit{patronNotion} &\rightarrow (\texttt{a} \mid \texttt{an})\ \mathit{patron} \\
\\
\mathit{teteFonction} &\rightarrow [\,\texttt{the}\,]\ \mathit{primNomObjet} \\
&\mid \mathit{primOperateurInfixe} \\
&\mid \mathit{primOperateurPrefixe} \\
&\mid \mathit{primOperateurPostfixe} \\
&\mid \mathit{patronFonction} \\
\\
\mathit{patronFonction} &\rightarrow \texttt{the}\ \mathit{patron} \\
&\mid \mathit{symbPatron} \\
\\
\mathit{tetePredicat} &\rightarrow \mathit{variable}\ \texttt{is}\ \mathit{primAdjectif} \\
&\mid \mathit{variable}\ \texttt{,}\ \mathit{variable}\ \texttt{are}\ \mathit{primAdjectifM} \\
&\mid \mathit{variable}\ \mathit{primVerbe} \\
&\mid \mathit{variable}\ \texttt{,}\ \mathit{variable}\ \mathit{primVerbeM} \\
&\mid \mathit{primRelation} \\
&\mid \mathit{patronPredicat} \\
\\
\mathit{patronPredicat} &\rightarrow \mathit{variable}\ \texttt{is}\ \mathit{patron} \\
&\mid \mathit{variable}\ \mathit{patron} \\
&\mid \mathit{symbPatron}
\end{aligned}
$$

Une proposition spéciale S bien formée doit satisfaire les conditions suivantes :

- dans l'unité de tête, tous les termes sont des variables (tête est plat) ;
- aucune variable n'intervient deux fois dans l'unité de tête (tête est linéaire) ;
- toute variable libre dans S (voir la section 1.4 pour la définition) intervient dans l'unité de tête ;
- si la primitive syntaxique dans l'unité de tête est présente dans le vocabulaire courant, elle n'a pas été introduite comme synonyme (voir ci-dessous). Par contre, elle peut être introduite par une définition ou extension de signature dans le texte précédent (c'est pourquoi nous permettons dans les unités de tête les primitives établies ainsi que les *patron*'s) ;
- l'unité de tête *teteNotion* a un nom au maximum ;
- la cible notion *nomClasse* a un nom au maximum.

Si l'unité de tête n'est pas basée sur une primitive déjà présente dans le vocabulaire courant, l'analyseur syntaxique construit et y ajoute une nouvelle primitive. Dans la section 1.3.1, nous avons expliqué comment les patrons sont convertis dans les primitives syntaxiques de base.

Une autre possibilité d'augmenter le vocabulaire courant est d'écrire une *déclaration de synonyme* :

$$\mathit{synonyme} \rightarrow \mathit{notionSyn} \mid \mathit{fonctionSyn} \mid \mathit{predicatSyn}$$

notionSyn → `let` *patronNotion* (`stand for` | `denote`) [`a` | `an`] *nomClasse* `.`

fonctionSyn → `let` *patronFonction* (`stand for` | `denote`) *termePrimord* `.`

predicatSyn → `let` *patronPredicat* (`stand for` | `denote`) *proposition* `.`

Le patron à gauche de la jonction principale (`stand for` | `denote`) est toujours appelé *l'unité de tête* et l'unité à droite, *l'unité-cible*. Comme dans les propositions spéciales, l'unité de tête dans les déclarations de synonymes doit être plat et linéaire et doit contenir toutes les variables qui interviennent librement dans la cible. Par contre, l'unité de tête dans une déclaration de synonyme ne peut pas coïncider avec une primitive déjà introduite.

Notez que les déclarations de synonymes ne sont pas les propositions, mais les instructions pour l'analyseur syntaxique, juste comme les groupes de tokens (section 1.3.1). On peut comparer les synonymes et les symboles définis avec, respectivement, les macros de préprocesseur et les routines dans un langage de programmation.

1.3.7 Description des variables

Nous disons qu'une proposition S *décrit* une variable v si S implique que v appartient à une classe désignée par quelque notion. Les définitions ci-dessous en donnent une explication formelle.

Un terme t est *positif* si t n'est pas une notion quantifiée de la forme `no` *notion* et les arguments de t sont aussi des termes positifs.

Une primitive syntaxique I est *descriptive* si l'une des conditions suivantes a lieu :

- I est l'adjectif prédéfini `equal to` arg_1 ;
- I est introduite comme synonyme-prédicat ; dans la déclaration de synonyme, le patron dans l'unité de tête commence avec une variable v et la proposition-cible décrit v (voir ci-dessous).

Une unité-prédicat P est *descriptive* si P n'est pas sous négation et l'une des conditions suivantes a lieu :

- P est un «is a»-prédicat formé par un terme fixe positif ;
- P est un «is a»-prédicat formé par une notion avec des termes positifs dans les arguments ;
- P est formé par un adjectif ou verbe descriptif avec des termes positifs dans les arguments ;
- P est composé de plusieurs prédicats primaires dont un est descriptif.

Finalement, une proposition S *décrit* une variable v si l'une des conditions suivantes a lieu :

- S est une proposition symbolique formée par une primitive descriptive de relation ; le premier argument de S est une séquence de variables qui contient v ;
- S est une proposition simple telle que le prédicat de S est descriptif et le sujet de S est une séquence de variables qui contient v ;
- S est une conjonction de plusieurs propositions dont une décrit v.

La descente récursive dans les définitions ci-dessus se termine parce que une primitive-synonyme ne peut intervenir ni dans son unité-cible ni au-dessus de la déclaration de synonyme dans le texte.

Par exemple, après le groupe de tokens et deux déclarations suivants :

```
[belongs/belong]
Let x belongs to y stand for (x is an element of y).
Let x << y stand for (x belongs to y).
```

la proposition :

```
u,v belong to some finite (s << W) and Q is a subset of no set
```

est bien-formée et décrit les variables `u`, `v` mais pas la variable `Q`. Notez que l'introduction de `<<` produit à la fois la primitive de relation et la primitive de relation-classe. C'est parce que la proposition-cible `x belongs to y` décrit la variable `x`.

1.4 L'image-formule des propositions

Au cours de l'analyse syntaxique d'une chaîne de lexèmes, nous identifions les fragments de la chaîne avec les non-terminaux conformément aux règles de la grammaire. Un fragment qui correspond à quelque non-terminal N sera appelé une *occurrence* de N. Remarquez que le contexte du fragment est important. Considérons deux propositions :

```
some natural number N divides 1
for some natural number N (N divides 1)
```

La ligne `some natural number N` est une occurrence du non-terminal *terme* dans la première proposition mais pas dans la deuxième. D'ailleurs, la ligne `natural number` n'est pas une occurrence de *notion* dans les deux propositions (bien que elle soit une notion bien formée en soi), parce que les occurrences actuelles du non-terminal *notion* comprennent aussi le nom de la notion `N`.

En utilisant la correspondance entre les fragments du texte et les non-terminaux, nous pouvons traduire une proposition S dans une formule du premier ordre $|S|$, appelée *l'image-formule* de S. Au cours de la traduction, nous appliquons les règles de transformation introduites dans les sections suivantes. Chaque règle doit être appliquée autant que possible avant que la règle suivante puisse être appliquée pour la première fois.

Nous définissons l'ensemble des variables libres d'une proposition S comme l'ensemble des variables libres de l'image-formule de S : $\mathcal{FV}(S) = \mathcal{FV}(|S|)$.

1.4.1 Normaliser la syntaxe

Pour simplifier la formulation des règles de transformation par la suite, nous commençons par une certaine normalisation de notre texte.

1. Dans toute occurrence de *primNomClasse* ou *primNomPossede*, où la partie [*names*] est vide (c.-à-d. aucun nom n'est donné), nous mettons une variable fraîche dans la position correspondante.
2. Nous décomposons les «and»-chaînes des notions dans les propositions quantifiées. La règle suivante s'applique aux occurrences de *propositionTete* :

   `for` $(notionQuantifiee)_O$
   `and` $(notionQuantifiee$ { `and` $notionQuantifiee$ $\})_T$ $(proposition)_S$
   $\Rightarrow$ `for` O `for` T S
3. Les articles `a`, `an`, `the` et le préfixe `we have` sont enlevés partout. La négation `it is wrong that` est remplacée avec simple `not`. La proposition constante `contrary` est remplacée par `not thesis`. Les verbes `be`, `are`, `do`, `have`, `exist` sont tous convertis dans le singulier de la troisième personne. Les mots-quantificateurs `all`, `each`, `any` sont remplacés par `every`. Dans les occurrences de *termes* et *notions*, les `and`'s sont remplacés par les virgules.

Dénotons par **N** tout non-terminal *primNomClasse*, *nomClasse*, *primRelationClasse*, *relationClasse*, *primNomPossede*, *nomPossede*, *notion* ou $notionQuantifiee$[2]. Pour toute occurrence O de **N**, notons $\mathbf{n}(O)$ la liste de variables dans la partie *names* dans la primitive correspondante. Après que la règle (1.4.1(1)) est passée, la liste de noms est non-vide pour toute occurrence de **N**.

1.4.2 Dégager les notions quantifiées I

Les règles suivantes enlèvent les notions quantifiées des occurrences de *terme* dans les sujets des propositions simples et dans les quantificateurs. Simplement dit, toute notion quantifiée O qui y intervient est remplacée par sa liste de noms $\mathbf{n}(O)$ et un quantificateur sur O est mis par-dessus la proposition.

[2] ainsi, **N** est un «méta-non-terminal»

Considérons deux occurrences S et O. Appelons O une *occurrence propre* dans S si O intervient proprement dans S. Appelons O une *occurrence native* dans S si O est propre dans S et n'appartient à aucune occurrence de *attributDroit* dans S.

Par exemple, dans la proposition ci-dessous, les occurrences X, Y et U de *notionQuantifiee* sont natives, tandis que l'occurrence V n'est pas native.

$$\texttt{(every member M of (some committee C)}_Y\texttt{)}_X\ \texttt{declares}$$
$$\texttt{(some opinion O such that (every member N of C)}_V\ \texttt{supports O)}_U$$

Deux règles de transformation ci-dessous s'appliquent aux occurrences de *propositionSimple* et *propositionTete*, respectivement. Dans les prémisses, O est la première (de gauche) occurrence native de *notionQuantifiee* dans S. Dans les conclusions, O est remplacée dans S par la liste de noms $\mathbf{n}(O)$:

1. $(termes\lceil(notionQuantifiee)_O\rfloor)_S\ (predicatDoes\ \{\,\texttt{and}\ predicatDoes\,\})_T$
   $\Rightarrow\quad \texttt{for}\ O\ \ S\lceil O \to \mathbf{n}(O)\rfloor\ \ T$
2. $\texttt{for}\ (notionQuantifiee\lceil(notionQuantifiee)_O\rfloor)_S\ (proposition)_T$
   $\Rightarrow\quad \texttt{for}\ O\ \texttt{for}\ S\lceil O \to \mathbf{n}(O)\rfloor\ \ T$

Pour la proposition dans l'exemple ci-dessus, ces règles produisent les transformations suivantes (nous ajoutons des parenthèses pour faciliter la lecture) :

```
every member M of some committee C declares some opinion O
          such that every member N of C supports O
                              ⇓ (1.4.2(1))
for (every member M of some committee C) M declares some opinion O
          such that every member N of C supports O
                              ⇓ (1.4.2(1))
for (every member M of some committee C) M declares some opinion O
        such that for (every member N of C) N supports O
                              ⇓ (1.4.2(2))
   for (some committee C) for (every member M of C) M declares
 some opinion O such that for (every member N of C) N supports O
```

1.4.3 Unifier les attributs

Maintenant nous transformons les attributs des notions vers une forme unifiée. Les règles suivantes s'appliquent aux occurrences de *nomClasse*, *nomRelation* et *nomPossede*.

1. $(\{\,leftAttribute\,\})_L\ (\mathbf{N})_O\ (predicatIs\ \{\,\texttt{and}\ predicatIs\,\})_A$
   $\Rightarrow\quad L\ O\ \texttt{such that}\ \mathbf{n}(O)\ \texttt{is}\ A$
2. $(\{\,leftAttribute\,\})_L\ (\mathbf{N})_O\ \texttt{that}\ (predicatDoes\ \{\,\texttt{and}\ predicatDoes\,\})_V$
   $\Rightarrow\quad L\ O\ \texttt{such that}\ \mathbf{n}(O)\ V$
3. $(\{\,leftAttribute\,\})_L\ (leftAttribute)_A\ (\mathbf{N})_O\ [\,\texttt{such that}\ (proposition)_S\,]$
   $\Rightarrow\quad L\ O\ \texttt{such that}\ \mathbf{n}(O)\ \texttt{is}\ A\ [\,\texttt{and}\ S\,]$

Après application de ces règles toutes les unités-prédicats sont envoyées aux propositions simples. Considérons la chaîne de transformations suivante :

```
                prime natural number X that divides N
                              ⇓ (1.4.3(2))
            prime natural number X such that X divides N
                              ⇓ (1.4.3(3))
        prime number X such that X is natural and X divides N
                              ⇓ (1.4.3(3))
    number X such that X is prime and X is natural and X divides N
```

1.4.4 Fendre les prédicats composés

Il est temps d'éliminer les conjonctions dan les propositions simples. Les règles suivantes s'appliquent aux occurrences de *propositionSimple* :

1. $(termes)_O$ $(predicatDoes)_P$ `and` $(predicatDoes$ { `and` $predicatDoes$ }$)_T$
   ⇒ O P `and` O T
2. $(termes)_O$ `is` $(predicatIs)_P$ `and` $(predicatIs$ { `and` $predicatIs$ }$)_T$
   ⇒ O `is` P `and` O `is` T
3. $(termes)_O$ `is` $(predicatIsA)_P$ `and` $(predicatIsA$ { `and` $predicatIsA$ }$)_T$
   ⇒ O `is` P `and` O `is` T

Notez que dans les règles ci-dessus, tous les termes dans $(termes)_O$ sont primordiaux.

1.4.5 Éliminer les «there is»-propositions

Par la suite, $\bar{O}$ dénote une occurrence O de **N** (sauf *primRelationClasse* et *relationClasse*) avec la partie *noms* enlevée, c.-à-d. avec la liste de noms $\mathbf{n}(O)$ remise à la liste nulle.

En cette étape, nous transformons les propositions primaires existentielles en propositions quantifiées, avec les quantificateurs non-bornés. Les règles suivantes s'appliquent aux occurrences de *propositionThereIs* :

1. `there exists` $(nomClasse)_O$ [`,` $(notions)_T$]
   ⇒ `for some` $\mathbf{n}(O)$ (`n`(O) `is` $\bar{O}$ [`and there exists` T])
2. `there exists no` $(nomClasse)_O$
   ⇒ `for every` $\mathbf{n}(O)$ (`n`(O) `is not` $\bar{O}$)
3. `there exists` $(relationClasse)_O$ [`such that` $(proposition)_S$] [`,` $(notions)_T$]
   ⇒ `for some` $\mathbf{n}(O)$ (O [`and` S] [`and there exists` T])
4. `there exists no` $(relationClasse)_O$ [`such that` $(proposition)_S$]
   ⇒ `for every` $\mathbf{n}(O)$ (`not` (O [`and` S]))

Illustrons ces règles par les transformations ci-dessous :

```
there exists number E and prime divisors G,H of E
                    ⇓ (1.4.1(3),1.4.3(3))
there exists number E, divisors G,H of E such that G,H is prime
                    ⇓ (1.4.5(1))
   for some E (E is a number and there exists
   divisors G,H of E such that G,H is prime)
                    ⇓ (1.4.5(1))
   for some E (E is a number and for some G,H
  (G,H is divisors of E such that G,H is prime))
```

1.4.6 Dégager les notions quantifiées II

Les règles ci-dessous enlèvent les notions quantifiées du reste des occurrences de *terme*. Ainsi, toutes les unités-termes deviennent les termes primordiaux.

Comme les règles (1.4.2(1-2)), les transformations suivantes s'appliquent aux occurrences de *propositionSimple* et *propositionTete*, respectivement. Dans les prémisses, O est la première (de gauche) occurrence native de $notionQuantifiee$ dans S. Dans les conclusions, O est remplacée dans S par $\mathbf{n}(O)$:

1. $(termes)_T$ $(predicatDoes\lceil(notionQuantifiee)_O\rfloor)_S$
   ⇒ `for` O T $S\lceil O \to \mathbf{n}(O)\rfloor$
2. `for` $(notionQuantifiee\lceil(notionQuantifiee)_O\rfloor)_S$ $(proposition)_T$
   ⇒ `for` O `for` $S\lceil O \to \mathbf{n}(O)\rfloor$ T

L'exemple ci-dessus peut ainsi continué comme suit :

`for (some committee C) for (every member M of C) M declares some opinion O such that for (every member N of C) N supports O`
$\Downarrow$ (1.4.6(1))
`for (some committee C) for (every member M of C) for (some opinion O such that for (every member N of C) N supports O) M declares O`

Remarque. Nous appliquons les règles (1.4.5(1-4)) avant les règles (1.4.6(1-2)), car autrement la proposition `there exists a subset of every set` serait transformée en `for every set X there exists a subset Y of X`, tandis que le sens correct est `for some Y for every set X (Y is a subset of X)`.

1.4.7 Transformer les quantificateurs

Nous transformons ici les quantificateurs bornés de ForTheL en quantificateurs non-bornés au-dessus des implications et conjonctions. Les règles suivantes s'appliquent aux occurrences de *propositionTete* (1.4.7(1-6)) et *symbProposition* (1.4.7(7-8)) :

1. `for [(] every` $(nomClasse)_O$ `[)] (proposition)`$_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` $\mathbf{n}(O)$ `is` $\bar{O}$ `then` S`)`
2. `for [(] some` $(nomClasse)_O$ `[)] (proposition)`$_S$
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`$\mathbf{n}(O)$ `is` $\bar{O}$ `and` S`)`
3. `for [(] no` $(nomClasse)_O$ `[)] (proposition)`$_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` $\mathbf{n}(O)$ `is` $\bar{O}$ `then (not` S`))`
4. `for [(] every` $(relationClasse)_O$ `[such that` $(proposition)_T$ `] [)] (proposition)`$_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` O `[and` T `] then` S`)`
5. `for [(] some` $(relationClasse)_O$ `[such that` $(proposition)_T$ `] [)] (proposition)`$_S$
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`O `[and` T `] and` S`)`
6. `for [(] no` $(relationClasse)_O$ `[such that` $(proposition)_T$ `] [)] (proposition)`$_S$
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(if` O `[and` T `] then (not` S`))`
7. `forall` $(relationClasse)_O$ $(symbProposition)_S$
   $\Rightarrow$ `forall` $\mathbf{n}(O)$ `(`O `=>` S`)`
8. `exists` $(relationClasse)_O$ $(symbProposition)_S$
   $\Rightarrow$ `exists` $\mathbf{n}(O)$ `(`O `/\` S`)`

1.4.8 Éliminer les «has»-prédicats

Pour toute occurrence O de *primNomPossede* et toute occurrence N de *terme*, $O\lceil N\rfloor$ notera la primitive originale de nom-classe ou nom-objet avec N remis sur la première place d'argument. Par exemple, si O est `subset` et N est `S`, alors $O\lceil N\rfloor$ est `subset of S`.

Les règles suivantes s'appliquent aux occurrences de *propositionSimple* :

1. $(termes)_N$ `is (with | of | having)` $(hasPredicate)_S$
   $\Rightarrow$ N `has` S
2. $(terme)_N$ `,` $(terme$ `{ , }` $terme)_T$ `has` $(hasPredicate)_S$
   $\Rightarrow$ N `has` S `and` T `has` S
3. $(terme)_N$ `has` $(primNomPossede)_O$ `[such that` $(proposition)_S$ `]`
   `[and (`$nomPossede$ `{and` $nomPossede$ `})`$_T$ `]`
   $\Rightarrow$ `for some` $\mathbf{n}(O)$ `(`$\mathbf{n}(O)$ `is` $\bar{O}\lceil N\rfloor$ `[and` S `] [and` N `has` T `])`
4. $(terme)_N$ `has no` $(primNomPossede)_O$ `[such that` $(proposition)_S$ `]`
   $\Rightarrow$ `for every` $\mathbf{n}(O)$ `(not (`$\mathbf{n}(O)$ `is` $\bar{O}\lceil N\rfloor$ `[and` S `]))`

Ces règles ressemblent à celles d'élimination de «there is». Illustrons-les par la chaîne de transformations suivante :

`M has a wife W and a salary not enough for W`
$\Downarrow$ (1.4.1(1), 1.4.1(3),1.4.3(1))

```
M has wife W and salary S such that S is not enough for W
```
$\Downarrow$ (1.4.8(3))
```
for some W (W is wife of M and
M has salary S such that S is not enough for W)
```
$\Downarrow$ (1.4.8(3))
```
for some W (W is wife of M and
for some S (S is salary of M and S is not enough for W))
```

Ici, `wife of _` est un nom-classe qui forme une notion, et `salary of _` est un nom-objet qui forme une fonction. Ainsi, dans la proposition finale nous avons les occurrences de *predicatIsA* des deux genres.

1.4.9 Traiter les «is a»-prédicats

Considérons une occurrence de *nomClasse*, de la forme

$$((primNomClasse)_O \texttt{ such that } (proposition)_S)$$

Après toutes les transformations précédentes, ce non-terminal n'intervient que dans les «is a»-prédicats. Rappelons que $\mathbf{n}(O)$ peut contenir un nom au maximum, et les règles ci-dessus maintiennent cette condition. Soit N une occurrence de *termes*. Si $\mathbf{n}(O)$ est vide, l'expression $S\lceil\mathbf{n}(O) \to N\rfloor$ est juste S. Si $\mathbf{n}(O)$ contient un nom v, l'expression $S\lceil\mathbf{n}(O) \to N\rfloor$ est le résultat de substitution de N dans toute occurrence de v dans S.

Les règles suivantes s'appliquent aux occurrences de *propositionSimple* :

1. $(terme)_N$ `,` $(terme\ \{$ `,` $terme\ \})_T$ `is` $([$`not`$])_C$ $(nomClasse)_O$
   $\Rightarrow$ N `is` C O `and` T `is` C O
   (si O ne contient pas dans l'attribut des m-prédicats appliqués à $\mathbf{n}(O)$)
2. $(termes)_N$ `is` $([$`not`$])_C$ $(primNomClasse)_O$ $[$`such that` $(proposition)_S\,]$
   $\Rightarrow$ `(`C `(`N `is` $\bar{O}$ $[$`and` $S\lceil\mathbf{n}(O) \to N\rfloor\,]$`))`
3. $(terme)_N$ `,` $(terme\ \{$ `,` $terme\ \})_T$ `is` $(primNomClasse)_O$
   $\Rightarrow$ N `is` O `and` T `is` O
4. $(termes)_N$ `is` $([$`not`$])_C$ $(definiteTerm)_O$
   $\Rightarrow$ N `is` C `equal to` O

Pour voir pourquoi la réserve dans la première règle est nécessaire, considérons deux chaînes de transformations. Dans le premier exemple, un «is a»-prédicat ne contient pas de m-primitives dans l'attribut. C'est pourquoi il s'applique à chaque sujet indépendamment :

```
A,B are not natural numbers
```
$\Downarrow$ (1.4.1(1),1.4.1(3),1.4.3(3))
```
A,B is not number X such that X is natural
```
$\Downarrow$ (1.4.9(1))
```
A is not number X such that X is natural
and B is not number X such that X is natural
```
$\Downarrow$ (1.4.9(2))
```
(not (A is number and A is natural))
and (not (B is number and B is natural))
```

Dans le deuxième exemple, un «is a»-prédicat emploie des m-primitives :

```
A,B are not equal numbers
```
$\Downarrow$ (1.4.1(1),1.4.1(3),1.4.3(3))
```
A,B is not number X such that X is equal
```
$\Downarrow$ (1.4.9(2))
```
not (A,B is number and A,B is equal)
```
$\Downarrow$ (1.4.9(3))
```
not (A is number and B is number and A,B is equal)
```

Remarquez aussi que l'occurrence N dans la règle (1.4.9(2)) contient plus qu'un terme seulement si S contient des m-prédicats appliqués à $\mathbf{n}(O)$. Dans ce cas, l'unité notion originale ne pouvait pas avoir de nom du tout (la première remarque dans la section 1.3.4). Par conséquent, $\mathbf{n}(O)$ a été introduit par la règle (1.4.1(1)). Par conséquent, $\mathbf{n}(O)$ n'intervient dans S que comme sujet dans une proposition simple (grâce aux règles (1.4.3(1-3))). Ainsi, l'expression $S\lceil\mathbf{n}(O) \rightarrow N\rfloor$ est bien formée.

1.4.10 Éliminer les m-prédicats

Soit S une occurrence de m-primitive et N une occurrence de *terme*. Nous dénotons par $S\lceil N\rfloor$ la primitive originale de verbe ou adjectif avec N remis sur la première place d'argument. Par exemple, si S est `parallel` et N est `X`, alors $S\lceil N\rfloor$ est `parallel to X`.

Les règles suivantes appliquent aux occurrences de *propositionSimple* :

1. $(terme)_N$ [`does`] [`not`] [`pairwise`] $primVerbeM$ $\Rightarrow$ erreur de syntaxe
2. $(terme)_N$ `is` [`not`] [`pairwise`] $primAdjectifM$ $\Rightarrow$ erreur de syntaxe
3. $(termes)_N$ [`does`] `not` ([`pairwise`])$_P$ $(primVerbeM)_S$
   $\Rightarrow$ `not (`N P S`)`
4. $(termes)_N$ `is not` ([`pairwise`])$_P$ $(primAdjectifM)_S$
   $\Rightarrow$ `not (`N `is` P S`)`
5. $(terme)_{N_1}$ `,` ... `,` $(terme)_{N_n}$ [`does`] `pairwise` $(primVerbeM)_S$
   $\Rightarrow$ `(`N_1 $S\lceil N_2\rfloor$`) and ... (`N_i $S\lceil N_{i+j}\rfloor$`) ... and (`N_{n-1} $S\lceil N_n\rfloor$`)`
6. $(terme)_{N_1}$ `,` ... `,` $(terme)_{N_n}$ [`does`] $(primVerbeM)_S$
   $\Rightarrow$ `(`N_1 $S\lceil N_2\rfloor$`) and ... (`N_i $S\lceil N_{i+1}\rfloor$`) ... and (`N_{n-1} $S\lceil N_n\rfloor$`)`
7. $(terme)_{N_1}$ `,` ... `,` $(terme)_{N_n}$ `is pairwise` $(primAdjectifM)_S$
   $\Rightarrow$ `(`N_1 `is` $S\lceil N_2\rfloor$`) and ... (`N_i `is` $S\lceil N_{i+j}\rfloor$`) ... and (`N_{n-1} `is` $S\lceil N_n\rfloor$`)`
8. $(terme)_{N_1}$ `,` ... `,` $(terme)_{N_n}$ `is` $(primAdjectifM)_S$
   $\Rightarrow$ `(`N_1 `is` $S\lceil N_2\rfloor$`) and ... (`N_i `is` $S\lceil N_{i+1}\rfloor$`) ... and (`N_{n-1} `is` $S\lceil N_n\rfloor$`)`

1.4.11 Traiter les sujets multiples

Enfin, nous transformons le reste des occurrences de *termes* ou *symbTermes*. Les règles ci-dessous s'appliquent aux occurrences de *propositionSimple* (1.4.11(1-2)) et *primRelation* (1.4.11(3)) :

1. $(terme)_N$ [`,` $(termes)_T$] [`does`] ([`not`])$_C$ $(primVerbe)_S$
   $\Rightarrow$ `(`C `(`N S`)) [and` T C S `]`
2. $(terme)_N$ [`,` $(termes)_T$] `is` ([`not`])$_C$ $(primAdjectif)_S$
   $\Rightarrow$ `(`C `(`N `is` S`)) [and` T `is` C S `]`
3. $(primRelation\lceil(symbTerme)_N$ `,` $(symbTermes)_T\rfloor)_S$
   $\Rightarrow$ $S\lceil N\rfloor$ `/\` $S\lceil T\rfloor$

1.4.12 Résoudre les synonymes

L'unité S que nous avons obtenue est effectivement une formule du premier ordre dont les atomes sont de six formes possibles :

$terme$ [`does`] $primVerbe$	$terme$ `is` $primAdjectif$	`contradiction`
$terme$ `is` $primNomClasse$	$primRelation$	`thesis`

Tous les termes dans cette formule sont composés des nom-objets et opérateurs symboliques et donc sont primordiaux. En convertissant les adjectifs et verbes dans les symboles des prédicats, le mot `contradiction` dans $\perp$, et le mot `thesis` dans $\mathfrak{T}$, nous allons obtenir une formule bien formé dans le langage décrit au début de la section 2.2.1.

La règle suivante s'applique aux primitives dans S qui ont été introduit comme les synonymes. Nous remplaçons toute telle primitive par une instance de l'unité-cible et appliquons

à nouveau toute la procédure de traduction pour la proposition obtenue. Cette récursion ne tombe jamais dans une descente infinie, car les déclarations de synonymes sont linéairement ordonnées dans le texte.

Par exemple, les déclarations de synonymes :

```
Let a relation on D stand for a relation with the domain equal to D.
Let U ** V stand for the intersection of U with V.
```

produisent la chaîne de transformations suivante :

```
X is relation on A ** B ** C
⇓ (1.4.12)
X is a relation with the domain equal to
the intersection of A with the intersection of B with C
⇓ (1.4.1–1.4.12)
X is relation and domain of X is equal to
intersection of A with intersection of B with C
```

Une fois tous les synonymes éliminés, la traduction est finie. Nous obtenons une formule bien formée du premier ordre. L'exemple d'une chaîne intégrale de transformations pour une proposition de ForTheL est donné à la figure 1.1.

1.4.13 Image d'une proposition spéciale

Au lieu d'intégrer les règles suivantes dans la séquence générale de transformations présentée ci-dessus, nous transformons une occurrence donnée de *defStatement* ou *sigStatement* en une proposition ordinaire et nous appliquons la procédure générale de traduction.

Pour une occurrence de *defStatement*, l'image-formule est calculée selon les équations suivantes :

$$
\begin{aligned}
|\,(teteNotion)_H \texttt{ is } (notion)_B\,| &= \forall v\,|\,v \texttt{ is } H \texttt{ iff } v \texttt{ is } B\,| \\
|\,(teteFonction)_H \texttt{ is } (termePrimord)_B\,| &= \forall v\,|\,v \texttt{ is equal to } H \texttt{ iff } v \texttt{ is } B\,| \\
|\,(teteFonction)_H \texttt{ is } (nomClasse)_B\,| &= \forall v\,|\,v \texttt{ is equal to } H \texttt{ iff } v \texttt{ is } B\,| \\
|\,(tetePredicat)_H \texttt{ iff } (proposition)_B\,| &= |\,H \texttt{ iff } B\,|
\end{aligned}
$$

L'image-formule de *sigStatement* est calculée comme suit :

$$
\begin{aligned}
|\,(teteNotion)_H \texttt{ is } (notion)_B\,| &= \forall v\,|\,\texttt{if } v \texttt{ is } H \texttt{ then } v \texttt{ is } B\,| \\
|\,(teteNotion)_H \texttt{ is notion}\,| &= \forall v\,|\,\texttt{if } v \texttt{ is } H \texttt{ then } \top\,| \\
|\,(teteFonction)_H \texttt{ is } (notion)_B\,| &= \forall v\,|\,\texttt{if } v \texttt{ is equal to } H \texttt{ then } v \texttt{ is } B\,| \\
|\,(teteFonction)_H \texttt{ is } (\texttt{term}\,|\,\texttt{constant})\,| &= \forall v\,|\,\texttt{if } v \texttt{ is equal to } H \texttt{ then } \top\,| \\
|\,(tetePredicat)_H \texttt{ implies } (proposition)_B\,| &= |\,\texttt{if } H \texttt{ then } B\,| \\
|\,(tetePredicat)_H \texttt{ is } (\texttt{atom})_B\,| &= |\,\texttt{if } H \texttt{ then } \top\,|
\end{aligned}
$$

Dans les équations ci-dessus, la variable v est soit $\mathbf{n}(H)$, soit une variable fraîche, si $\mathbf{n}(H)$ est vide. Le symbole $\top$ dénote la vérité.

1.5 Texte ForTheL

Nous commençons avec un exemple de texte en ForTheL à la figure 1.2 (les numéros des lignes ne sont pas compris). Par la suite, nous nous référons à ce texte comme au «Texte sur l'ensemble vide».

1.5.1 Sections de haut niveau

Un texte ForTheL est une séquence d'instructions pour l'analyseur syntaxique (groupes de tokens et déclarations de synonymes) et de *sections haut-niveau*. Il y a quatre *genres* de

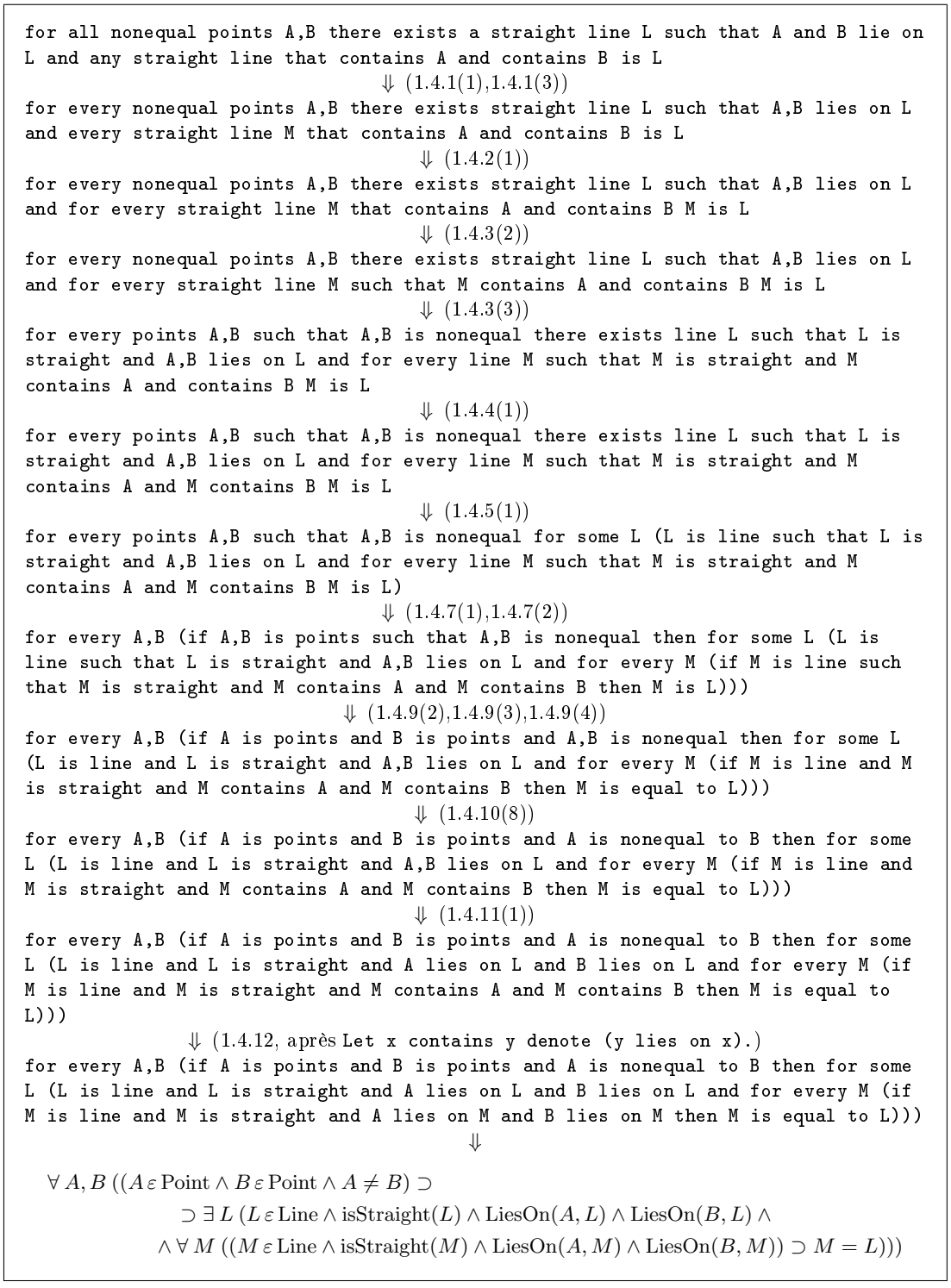


FIG. 1.1: Chaîne de traduction intégrale

```
 1:    [set/sets] [element/elements]
 2:    [belongs/belong] [subset/subsets]

 3:    Signature SetSort.
 4:      A set is a notion.

 5:    Signature ElmSort.
 6:      Let S be a set.
 7:      An element of S is a notion.

 8:    Let x belongs to y stand for (x is an element of y).
 9:    Let x is in y stand for (x belongs to y).

10:    Definition DefSubset.
11:      Let S be a set.
12:      A subset of S is a set T such that all elements of T are in S.

13:    Definition DefEmpty.
14:      Let S be a set.
15:      S is empty iff S has no elements.

16:    Axiom ExEmpty.
17:      There exists an empty set.

18:    Theorem.
19:      Let S be a set.
20:      S is a subset of every set iff S is empty.
21:    Proof.
22:      If S is empty then S is a subset of every set.
23:      Indeed if S is empty then any element of S belongs to any set T.
24:      Assume that S is a subset of every set.
25:      Let us show that S is empty.
26:        Let z belong to S.
27:        Take an empty set E.
28:        z is an element of E (by DefSubset).
29:        We have a contradiction (by DefEmpty).
30:      end.
31:    qed.
```

FIG. 1.2: Texte sur l'ensemble vide

sections haut-niveau : *axiomes*, *définitions*, *extensions de signature* et *théorèmes*. Dans le Texte sur l'ensemble vide il y a six sections haut-niveau : extension de signature pour la notion d'ensemble (lignes 3–4), extension de signature pour la notion d'élément d'ensemble (lignes 5–7), définition de la notion de sous-ensemble (lignes 10–12), définition de l'ensemble vide (lignes 13–15), axiome (lignes 16–17), théorème avec une démonstration (lignes 18–31).

Toute section haut-niveau a un *en-tête* qui détermine le genre de la section et peut contenir une *marque* à citer dans les *références* (comme dans les lignes 28 et 29). L'en-tête est suivi par le contenu de la section, que nous appelons son *corps*.

$$
\begin{aligned}
\mathit{texte} &\rightarrow \{\ \mathit{hautNiveau} \mid \mathit{groupeTokens} \mid \mathit{synonyme}\ \} \\
\mathit{hautNiveau} &\rightarrow \mathit{axiome} \mid \mathit{definition} \mid \mathit{signature} \mid \mathit{theoreme} \\
\mathit{axiome} &\rightarrow \mathit{axmEntete}\ \{\ \mathit{assume}\ \}\ \mathit{axmAffirm} \\
\mathit{axmEntete} &\rightarrow \texttt{Axiom}\ [\ \mathit{marque}\ ]\ . \\
\mathit{definition} &\rightarrow \mathit{defEntete}\ \{\ \mathit{assume}\ \}\ \mathit{defAffirm} \\
\mathit{defEntete} &\rightarrow \texttt{Definition}\ [\ \mathit{marque}\ ]\ . \\
\mathit{signature} &\rightarrow \mathit{sigEntete}\ \{\ \mathit{assume}\ \}\ \mathit{sigAffirm} \\
\mathit{sigEntete} &\rightarrow \texttt{Signature}\ [\ \mathit{marque}\ ]\ . \\
\mathit{proposition} &\rightarrow \mathit{thmEntete}\ \{\ \mathit{assume}\ \}\ \mathit{affirm} \\
\mathit{thmEntete} &\rightarrow (\ \texttt{Theorem} \mid \texttt{Lemma} \mid \texttt{Corollary} \mid \texttt{Proposition}\ )\ [\ \mathit{marque}\ ]\ . \\
\mathit{marque} &\rightarrow \mathit{mot}
\end{aligned}
$$

1.5.2 Phrases

Dans le corps de toute section composée nous voyons les *phrases* aussi de quatre *genres* : *assomptions* (lignes 6, 11, 14, 16, 19, 24, 26 dans le Texte sur l'ensemble vide), *sélections* (ligne 27), *affirmations* des formes variées (lignes 4, 7, 12, 15, 17, 20–31, 22–23, 23, 25–30, 28, 29), et *hypothèses de cas* (ne sont pas employées dans l'exemple). D'après les règles grammaticales ci-dessus, le corps de toute section haut-niveau est formé par zéro ou plus assomptions suivies par une affirmation de la forme appropriée.

Les assomptions, affirmations et hypothèses de cas sont construites des unités propositions. Les sélections sont construites des unités notions, elles expriment l'existence des objets dans les classes correspondantes.

$$
\begin{aligned}
\mathit{assume} &\rightarrow \mathit{asmPrefixe}\ \mathit{proposition}\ . \\
\mathit{asmPrefixe} &\rightarrow \texttt{let} \mid [\ \texttt{let us} \mid \texttt{we can}\ ]\ (\ \texttt{assume} \mid \texttt{suppose}\ )\ [\ \texttt{that}\ ] \\
\mathit{axmAffirm} &\rightarrow \mathit{proposition}\ . \\
\mathit{defAffirm} &\rightarrow \mathit{defProposition}\ . \\
\mathit{sigAffirm} &\rightarrow \mathit{sigProposition}\ . \\
\mathit{affirm} &\rightarrow \mathit{affPrefixe}\ \mathit{proposition}\ [\ \mathit{ref}\ ]\ .\ [\ \mathit{prfEntete}\ \mathit{preuve}\ \mathit{qed}\ ] \\
&\quad \mid \mathit{affPrefixe}\ \mathit{proposition}\ [\ \mathit{ref}\ ]\ .\ \texttt{indeed}\ \mathit{preuveCourte} \\
&\quad \mid \mathit{prfPrefixe}\ \mathit{proposition}\ [\ \mathit{ref}\ ]\ .\ \mathit{preuve}\ \mathit{qed}
\end{aligned}
$$

$$
\begin{aligned}
affPrefixe &\rightarrow [\,\texttt{then} \mid \texttt{therefore} \mid \texttt{hence}\,] \\
prfPrefixe &\rightarrow [\,\texttt{let us} \mid \texttt{we can}\,]\ (\,\texttt{prove} \mid \texttt{show}\,)\ [\,\texttt{by}\ method\,]\ [\,\texttt{that}\,] \\
select &\rightarrow selPrefixe\ notions\ [\,ref\,]\ \texttt{.}\ [\,prfEntete\ preuve\ qed\,] \\
&\mid\ selPrefixe\ notions\ [\,ref\,]\ \texttt{.}\ \texttt{indeed}\ preuveCourte \\
selPrefixe &\rightarrow [\,\texttt{then} \mid \texttt{therefore} \mid \texttt{hence}\,]\ [\,\texttt{let us} \mid \texttt{we can}\,]\ (\,\texttt{take} \mid \texttt{choose}\,) \\
cas &\rightarrow \texttt{case}\ proposition\ \texttt{.}\ preuve\ qed \\
prfEntete &\rightarrow \texttt{proof}\ [\,\texttt{by}\ methode\,]\ \texttt{.} \\
methode &\rightarrow \texttt{contradiction} \mid \texttt{case analysis} \mid \texttt{induction}\ [\,\texttt{on}\ plainTerm\,] \\
ref &\rightarrow \texttt{(}\ \texttt{by}\ marque\ \{\ \texttt{,}\ marque\,\}\ \texttt{)} \\
qed &\rightarrow \texttt{end.} \mid \texttt{qed.} \mid \texttt{obvious.} \mid \texttt{trivial.}
\end{aligned}
$$

Notez que les *preuve*'s sont la partie structurelle, le *corps*, d'une section affirmation ou sélection. Ainsi, les affirmations sur les lignes 20, 22 et 25 occupent chacune plusieurs lignes. De même, la démonstration d'un cas de preuve constitue le *corps* de l'hypothèse. C'est pourquoi nous allons souvent appeler cette section dans son intégralité le *cas de preuve*, l'identifiant ainsi avec la phrase-hypothèse.

1.5.3 Démonstrations

Dans un texte correct, tout cas de preuve, sélection ou affirmation (non-compris les affirmations dans les axiomes, définitions et extensions de signature) doit être fondé — doit s'ensuivre de ses prédécesseurs logiques dans le texte. L'auteur peut faciliter la déduction soit en citant les sections haut-niveau nécessaires dans les références, soit avec une *démonstration* qui fera un prédécesseur logique supplémentaire. Dans le Texte sur l'ensemble vide il y a trois démonstrations : une pour l'affirmation sur la ligne 20 (lignes 21–31), une pour l'affirmation sur la ligne 22 (ligne 23), et une pour l'affirmation sur ligne 25 (lignes 26–30).

Comme nous avons dit, les démonstrations ne sont pas des sections. Une démonstration est une liste de phrases (qui peuvent avoir leurs propres démonstrations) qui constitue le corps de l'affirmation, sélection ou hypothèse de cas. Les démonstrations courtes (ligne 23) sont des listes à un seul élément, une affirmation. La démonstration principale dans le Texte sur l'ensemble vide est composée de trois phrases : une affirmation, une assomption et une autre affirmation.

$$
\begin{aligned}
preuve &\rightarrow [\,\{\,prvCorps\,\}\ prvFin\,] \\
prvCorps &\rightarrow affirm \mid select \mid assume \\
prvFin &\rightarrow affirm \mid select \mid cas\,\{\,cas\,\} \\
preuveCourte &\rightarrow affirm
\end{aligned}
$$

L'auteur peut mentionner explicitement la *méthode de démonstration* qu'il applique. À ce moment, les démonstrations par contradiction, par analyse de cas et par induction générale sont admises. Les mentions «`by contradiction`» et «`by case analysis`» sont facultatives, car le vérificateur est capable d'établir l'application de ces méthodes par la forme de la démonstration. Par contre, la mention «`by induction`» est nécessaire : elle dit au vérificateur de construire une *hypothèse d'induction* appropriée et de l'insérer dans la démonstration. Nous allons expliquer la vérification des démonstrations par induction dans la section 1.6.2 d'une manière informelle et après dans la section 2.3.2 d'une façon précise.

1.5.4 Précédence logique. Déclaration des variables

Les occurrences de sections dans un texte sont liées par les relations de *subordination* et *précédence*. Dans les définitions suivantes nous disons «section» pour dire «occurrence d'une section dans le texte en considération».

Les *bornes* des sections sont placées conformément aux règles grammaticales ci-dessus. Par exemple, toute section haut-niveau commence avec le premier caractère de son en-tête et finit avec la fin de son affirmation. De même, une phrase avec la démonstration finit avec la fin de la démonstration et pas avec le point final de la phrase.

Une section $\mathbb{A}$ est appelée *sous-section* d'une section $\mathbb{B}$ si $\mathbb{A}$ se trouve dans les bornes de $\mathbb{B}$ et $\mathbb{B}$ n'est pas $\mathbb{A}$. Ainsi, une affirmation avec la démonstration est une super-section pour toute section intervenant dans la démonstration. Nous appelons $\mathbb{A}$ une *sous-section immédiate* de $\mathbb{B}$ si $\mathbb{B}$ est la moindre super-section de $\mathbb{A}$ (i.e. $\mathbb{A}$ est un élément du corps de $\mathbb{B}$).

Une section $\mathbb{A}$ est un *prédécesseur textuel* d'une section $\mathbb{B}$ si $\mathbb{B}$ commence après la fin de $\mathbb{A}$ dans le texte. Une section $\mathbb{A}$ est un *prédécesseur logique* d'une section $\mathbb{B}$ si $\mathbb{A}$ précède $\mathbb{B}$ textuellement et toute super-section de $\mathbb{A}$ est aussi une super-section de $\mathbb{B}$. Par défaut, «prédécesseur» signifie «prédécesseur logique».

Dans le Texte sur l'ensemble vide, l'assomption sur la ligne 23 a les prédécesseurs logiques suivants : les deux extensions de signature, les deux définitions, l'axiome `ExEmpty`, l'assomption sur la ligne 19 et l'affirmation sur la ligne 22. Les affirmations sur les lignes 20 et 23 ne sont pas les prédécesseurs de l'assomption en question, car la première est en une super-section et la deuxième est une sous-section d'un prédécesseur logique.

Nous utilisons la précédence logique pour déterminer l'ensemble des variables déclarées par une assomption ou une sélection. Les autres genres de section ne peuvent pas déclarer de variables.

Une assomption *décrit* une variable v si la proposition de l'assomption décrit v (voir la section 1.3.7). Une sélection *décrit* une variable v si v est un des noms de quelque notion O dans la chaîne, i.e. $v \in \mathbf{n}(O)$.

Une assomption ou sélection $\mathbb{A}$ *déclare* une variable v si $\mathbb{A}$ décrit v et aucun des prédécesseurs logiques de $\mathbb{A}$ ne décrit v. Si une variable est déclarée par un prédécesseur logique de $\mathbb{A}$, elle est dite être *connue* dans $\mathbb{A}$. Ainsi, une variable connue ne peut jamais être déclarée.

Dans un texte bien formé, toute variable libre dans une phrase (voir la section suivante pour la définition) doit être soit connue soit déclarée dans cette phrase. D'ailleurs, toute variable décrite dans une phrase-sélection doit être inconnue (et donc déclarée) dans cette phrase.

1.5.5 Image-formule des sections

Toute section A dans un texte ForTheL bien formé peut être traduit dans le langage du premier ordre. La formule-traduction dénotée $|\mathbb{A}|$, est appelée *l'image-formule* de $\mathbb{A}$.

– Assomptions : $|\, asmPrefixe\ (proposition)_S\ .\ | = |S|$

– Affirmations :

$$|\,(proposition)_S\ .\ | = |\,(defProposition)_S\ .\ | = |\,(sigProposition)_S\ .\ | = |S|$$
$$|\,affPrefixe\ (proposition)_S\ [\,ref\,]\ .\ [\,prfEntete\ preuve\ qed\,]\,| = |S|$$
$$|\,affPrefixe\ (proposition)_S\ [\,ref\,]\ .\ \texttt{indeed}\ preuveCourte\,| = |S|$$
$$|\,prfPrefixe\ (proposition)_S\ [\,ref\,]\ .\ preuve\ qed\,| = |S|$$

– Sélections :

$$|\,selPrefixe\ (notions)_N\ [\,ref\,]\ .\ [\,prfEntete\ preuve\ qed\,]\,| = \overline{|\,\texttt{there exist}\ N\,|}$$
$$|\,selPrefixe\ (notions)_N\ [\,ref\,]\ .\ \texttt{indeed}\ preuveCourte\,| = \overline{|\,\texttt{there exist}\ N\,|}$$

où, pour toute variable v décrite par la sélection (c.-à-d. pour tous les noms de toutes les notions dans N), le quantificateur existentiel sur v est enlevé de la formule avec la barre,

libérant v. Considérons les exemples suivants :

$$|\ \texttt{Take a set S and an element of S.}\ | = (\mathrm{aSet}(S) \wedge \exists x\,(\mathrm{aElement}(x, S)))$$
$$|\ \texttt{Take a set S and an element x of S.}\ | = (\mathrm{aSet}(S) \wedge \mathrm{aElement}(x, S))$$
$$|\ \texttt{Take a set and an element x of S.}\ | = \exists z\,(\mathrm{aSet}(z) \wedge \mathrm{aElement}(x, S))$$

– Cas de preuve : $|\ \texttt{case}\ (proposition)_S\ \texttt{.}\ preuve\ qed\ | = |S| \supset \texttt{thesis}$
– Sections haut-niveau :

$$|\ axmEntete\ (\{\ assume\ \}\ axmAffirm)_\Delta\ | = |\Delta|$$
$$|\ defEntete\ (\{\ assume\ \}\ defAffirm)_\Delta\ | = |\Delta|$$
$$|\ sigEntete\ (\{\ assume\ \}\ sigAffirm)_\Delta\ | = |\Delta|$$
$$|\ thmEntete\ (\{\ assume\ \}\ affirm)_\Delta\ | = |\Delta|$$

où $|\Delta|$, l'image-formule de la séquence de phrases Δ, est calculée selon les règles suivantes :

$\|\ \mathbb{A}\ \Delta\ \| = \forall \vec{x}\,(\|\mathbb{A}\| \supset \|\Delta\|)$	où	$\mathbb{A}$ est une assomption
		$\vec{x}$ est l'ensemble des variables déclarées dans $\mathbb{A}$
$\|\ \mathbb{A}\ \Delta\ \| = \exists \vec{x}\,(\|\mathbb{A}\| \wedge \|\Delta\|)$	où	$\mathbb{A}$ est une affirmation, sélection, ou cas de preuve
		$\vec{x}$ est l'ensemble des variables déclarées dans $\mathbb{A}$
		($\vec{x} = \varnothing$ si $\mathbb{A}$ n'est pas une sélection)
$\|\ \epsilon\ \| = \top$		l'image de la séquence nulle est la vérité

Notez que la présence d'une démonstration sous une affirmation, sélection, ou cas de preuve n'influence pas l'image-formule de la section. L'image exprime le sens voulu par l'auteur. Il appartient au vérificateur de contrôler que la démonstration fournie soutient ce sens.

Nous définissons l'ensemble des variables libres d'une section $\mathbb{A}$ comme ensemble des variables libres de son image-formule : $\mathcal{FV}(\mathbb{A}) = \mathcal{FV}(|\mathbb{A}|)$. Nous dénotons par $\mathcal{DV}(\mathbb{A})$ l'ensemble des variables déclarées dans $\mathbb{A}$. Si $\mathbb{A}$ n'est pas une assomption ou sélection, $\mathcal{DV}(\mathbb{A})$ est vide. Pour une séquence de sections Δ, les ensembles $\mathcal{FV}(\Delta)$ et $\mathcal{DV}(\Delta)$ sont définis comme unions des ensembles correspondants des sections individuelles.

Soulignons qu'on doit considérer une phrase de ForTheL en vue de ses prédécesseurs logiques pour établir l'ensemble des variables déclarées. Par conséquent, l'image-formule d'une séquence de phrases aussi dépend des prédécesseurs logiques. Pour afficher explicitement cette dépendance, nous introduisons la notation ci-dessous :

Soit Γ une séquence de sections et F une formule. L'ensemble $\mathcal{DV}_\Gamma(F)$ de variables «définies» dans F en vue de Γ est la différence $\mathcal{FV}(F)\backslash\mathcal{FV}(\Gamma)$. De même, $\mathcal{DV}_\Gamma(\mathbb{A}) = \mathcal{FV}(|\mathbb{A}|)\backslash\mathcal{FV}(\Gamma)$.

L'image-formule d'une séquence Δ en vue de Γ, dénotée $|\Delta|_\Gamma$, est définie comme suit :

$\|\ \mathbb{A}\ \Delta\ |_\Gamma = \forall \vec{x}\,(\|\mathbb{A}\| \supset \|\Delta|_{\Gamma,\mathbb{A}})$	$\mathbb{A}$ est une assomption et $\vec{x} = \mathcal{DV}_\Gamma(\mathbb{A})$
$\|\ \mathbb{A}\ \Delta\ |_\Gamma = \exists \vec{x}\,(\|\mathbb{A}\| \wedge \|\Delta|_{\Gamma,\mathbb{A}})$	$\mathbb{A}$ est une affirmation, selection ou cas, et $\vec{x} = \mathcal{DV}_\Gamma(\mathbb{A})$
$\|\ \epsilon\ |_\Gamma = \top$	l'image de la séquence nulle est la vérité

Dans un texte ForTheL bien formé, les ensembles $\mathcal{DV}(\mathbb{A})$ et $\mathcal{DV}_\Gamma(\mathbb{A})$ sont les mêmes si Γ est l'ensemble des prédécesseurs logiques de $\mathbb{A}$. Par conséquent, l'image $|\Delta|$ est égal à $|\Delta|_\Gamma$.

1.6 Exemple de formalisation

La démonstration formelle du fait que $\sqrt{2}$ n'est pas un nombre rationnel est un problème classique formalisé dans beaucoup d'assistants mathématiques [78]. Nous avons choisi sa version généralisée pour démontrer pas à pas le processus de formalisation dans ForTheL et préparation de texte vérifiable. Nous montrons comment le problème initial est transformé en un texte clos et comment ce texte doit être raffiné pour compléter les lacunes dans la démonstration que le système de vérification ne peut pas franchir de lui-même.

1.6.1 S'enfoncer dans le problème

L'existence des nombres qui ne peuvent pas être représentés comme une fraction de deux entiers a été prouvée par les mathématiciens de l'école de Pythagore qui ont démontré que la diagonale d'un carré avec côté 1 n'est pas une telle fraction. Voici le théorème dans sa forme traditionnelle :

THÉORÈME. $\sqrt{2}$ n'est pas rationnel.

Démonstration. Supposons que $\sqrt{2}$ est rationnel. Alors il y a deux nombres co-premiers n et m tels que $\sqrt{2} = n/m$, et donc $2m^2 = n^2$. Comme n^2 est pair, n est pair aussi ; mettons $n = 2k$. Alors $2m^2 = 4k^2$, donc $m^2 = 2k^2$. Ainsi, m^2 est aussi pair, et m de même. Par cela, n et m ont un diviseur commun supérieur à 1 ce qui contredit notre hypothèse. □

La proposition peut être facilement généralisée comme suit :

THÉORÈME. La racine carrée d'un nombre premier n'est pas rationnel.

Démonstration. Soit p un nombre premier tel que $\sqrt{p}$ est rationnel. Alors il y a deux nombres co-premiers n et m tels que $\sqrt{p} = n/m$, et donc $pm^2 = n^2$. Comme p est premier, p divise n ; mettons $n = kp$. Alors $pm^2 = p^2k^2$, donc $m^2 = pk^2$. Ainsi, p divise m^2 et m. Par cela, n et m ont un diviseur commun supérieur à 1 et nous obtenons une contradiction. □

Le fait que $2|n^2$ implique $2|n$ peut être démontré directement : $(2k+1)^2 = 2(2k^2+2k)+1$. L'assertion généralisée ($p|n^2 \supset p|n$) n'est pas aussi évident. Alors, pour faire notre démonstration vérifiable, nous appelons au lemme ci-dessous (dénoté PDP pour «Prime Divise Produit») :

LEMME PDP. Soit m et n des nombres naturels et p un nombre premier. Si p divise mn, alors p divise m ou n.

Démonstration. Nous allons démontrer ce lemme par induction sur $(m+n+p)$.

Si $m \geqslant p$, alors $p|mn$ implique $p|(m-p)n$ et nous pouvons se servir de l'hypothèse d'induction ($p|m-p$ évidemment implique $p|m$). De la même manière nous traitons le cas où $n \geqslant p$.

Supposons que $m, n < p$ et $pk = mn$. Si k est 0 ou 1, le lemme est prouvé. Sinon, soit r un diviseur premier de k. Alors r divise mn, et donc $r|m$ ou $r|n$, par l'hypothèse d'induction et le fait que $r \leqslant k < p$.

Si r divise m alors $p(k/r) = (m/r)n$ et l'hypothèse d'induction est applicable. Évidemment, $p|(m/r)$ implique $p|m$. Le cas où $r|n$ peut être démontré de la même façon. □

Ce que nous avons n'est pas encore un texte clos, suffisant en lui-même. Nous devons le compléter avec un ensemble des faits préliminaires, y compris (dans l'ordre d'introduction) : les propriétés basiques des nombres naturels (addition, soustraction, ordonnancement), des fragments de l'arithmétique Euclidienne (divisibilité, quotient, nombres premiers et co-premiers), définitions du nombre rationnel et de la racine carrée. Tous ces *préliminaires* sont des faits bien connus et nous pouvons les accepter sans preuve.

1.6.2 ForTheL'isation

L'appareil mathématique employé dans notre exemple est bien simple. C'est pourquoi il nous suffit juste de rendre notre notation dans l'ensemble des caractères ASCII et éliminer les excès de syntaxe pour rédiger ce texte en ForTheL. Nous citons les deux résultats principaux et omettons les préliminaires (les exemples de textes complets sont donnés dans les annexes).

Le texte à la figure 1.3 (dans le reste de cette section nous l'appelons le Texte sur la racine carrée) est un texte ForTheL bien formé. Fourni les préliminaires appropriés, il est probablement correct, ce qui veut dire que toute assertion faite dans le texte peut être déduite de ses prédécesseurs logiques. (La définition formelle d'un texte correct sera donnée dans le chapitre suivant). Nous disons «probablement», car ce texte rend les démonstrations que nous avons rédigées ci-dessus en langue naturelle et nous croyons que ces démonstrations sont valides. Pourtant, nous ne pouvons pas être absolument sûrs dans aucune de ces deux affirmations, et vérification formelle est donc nécessaire.

```
Lemma PDP.  For all natural numbers n,m,p
  if p is prime and p | n * m then p | n or p | m.
Proof by induction on ((n + m) + p).
  Let n,m,p be natural numbers.
  Assume that p is prime and p divides n * m.

  Case p <= n.
    p divides (n - p) * m and n - p < n.
    Then p divides n - p or p divides m.
    If p divides n - p then p divides n.
  end.

  Case p <= m.
    p divides n * (m - p) and m - p < m.
    Then p divides n or p divides m - p.
    If p divides m - p then p divides m.
  end.
  Case n < p and m < p.
    Take a natural number k such that n * m = p * k.
    Case k = 0 \/ k = 1. Obvious.
    Case k != 0 /\ k != 1.
      Take a prime divisor r of k.
      r divides n * m and r <= k and k < p.
      Then r divides n or r divides m.
      Case r divides n.
        p divides (n / r) * m and (n / r) < n.
        Then p divides (n / r) or p divides m.
        If p divides (n / r) then p divides n.
      end.
      Case r divides m.
        p divides n * (m / r) and (m / r) < m.
        Then p divides n or p divides (m / r).
        If p divides (m / r) then p divides m.
      end.
    end.
  end.
qed.

Theorem Main.  Let p be a prime natural number.
  For no rational number q the square of q is p.
Proof by contradiction.
  Let q be a rational number such that q * q = p.
  Take relatively prime natural numbers n,m such that q * m = n.
  Then p * (m * m) = (n * n).
  Hence p divides n * n and p divides n.
  Choose a natural number k such that n = p * k.
  Then we have p * (m * m) = p * (k * n).
  The square of m is equal to (p * k) * k.
  Hence p divides m * m and p divides m.
  We have a contradiction.
qed.
```

FIG. 1.3: Texte sur la racine carrée

Expliquons comment un texte en langue naturelle est converti en ForTheL. Premièrement, nous ajoutons un marquage structurel. Les démonstrations, les sous-démonstrations, les cas de preuve doivent avoir des en-têtes et suffixes explicites. Remarquons que notre vérificateur ne supporte pas la démonstration par analogie. C'est pourquoi nous sommes obligés de répéter les fragments de raisonnement avec des modifications minuscules dans la démonstration du lemme PDP (comparez les cas «`p <= n`» et «`p <= m`» ; «`r divides n`» et «`r divides m`»).

Comme notre intention est de démontrer une propriété de nombres premiers, nous pouvons simplifier notre texte en éliminant les opérations que nous utilisons «en passant», telles que la racine carrée ou division sur les nombres rationnels. (Notons qu'un résultat bien proche peut être formulé et démontré sans utiliser du tout la notion de nombre rationnel.)

Nous enlevons certains rappels destinés à un lecteur humain, tels que «Comme p est premier». Ils ne sont pas nécessaires pour la vérification (quoi que l'on puisse les considérer comme suggestions des prémisses importantes) et ne sont pas inclus dans la syntaxe courante de ForTheL. Par ailleurs, nous allons peut-être ajouter des propositions supplémentaires qui peuvent sembler superflu mais qui font la vérification automatique possible (voir section 1.6.3).

Enfin, comme ForTheL formalise juste un petit fragment de langue anglaise (et même «anglaise mathématique»), il y a certaines expressions naturelles qui doivent être reformulées. Par exemple, nous défaisons les propositions où un verbe s'applique à plusieurs objets : nous disons «p divides m^2 and p divides m» au lieu de «p divides m^2 and m».

Pour conclure, donnons une explications informelle sur l'emploi de l'induction (voir la section 2.3.2 pour les définitions précises). Les démonstrations par induction n'ont aucune structure particulière dans ForTheL ; par exemple, il n'y a aucun marquage spécial pour la base et le pas d'induction. Pour rédiger une démonstration par induction, l'auteur doit explicitement mentionner la méthode de démonstration : «`proof by induction` [`on` *term*]». La proposition à prouver doit correspondre à une formule universelle de la forme $\forall \vec{v} F$ et les variables libres du terme d'induction (notons-le t) doivent être parmi $\vec{v}$. Si le terme d'induction est omis, alors la variable sous le quantificateur le plus externe est prise comme t. Ces variables doivent être déclarées au début de la démonstration de la même façon qu'elles sont employées dans la proposition-but. Si toutes ces conditions sont satisfaites, le vérificateur construit et met dans la démonstration l'hypothèse d'induction appropriée :

$$\forall\, \vec{u}\, (t[\vec{u}/\vec{v}] \prec t \ \supset\ F[\vec{u}/\vec{v}])$$

Ici, la relation binaire $\prec$ désigne un ordonnancement abstrait bien-fondé, comme nous l'expliquons plus bas. Si nous considérons les variables libres de t comme constantes (avec des valeurs arbitraires mais fixes) introduites quelque part plus haut dans la démonstration, et appelons t la *valeur courante* du terme d'induction, nous pouvons exprimer l'hypothèse d'induction comme suit : *l'assertion a lieu pour toute valeur du terme d'induction qui est inférieure à la valeur courante selon quelque ordonnancement bien-fondé fixe.*

Si nous arrivons à démontrer l'assertion pour la valeur courante du terme d'induction en présence de l'hypothèse d'induction, nous pouvons conclure que l'assertion a lieu pour toutes les valeurs de ce terme et donc la proposition initiale est ainsi démontrée. Autrement dit, la méthode de démonstration par induction dans ForTheL suit le *principe général d'induction.*

Comme une axiomatisation finie d'une relation bien-fondée n'est pas possible en logique du premier ordre, le vérificateur ne peut en aucune façon contrôler qu'un ordonnancement donné soit bien-fondé. C'est pourquoi ForTheL offre une relation symbolique spéciale `-<-` qui est utilisée dans les démonstrations par induction comme un ordonnancement bien-fondé *a priori.* C'est l'auteur qui fournit les axiomes définissant les propriétés particulières de la relation `-<-`, et le système acceptera sans preuves que ces axiomes sont compatibles avec la bonne fondation.

Dans notre exemple, l'axiome correspondant est :

```
Axiom IH.   For all natural numbers n,m (n < m => n -<- m).
```

et l'hypothèse d'induction pour la démonstration de PDP est comme suit :

```
For all natural numbers N,M,P if ((N + M) + P) -<- ((n + m) + p)
  then if P is prime and P | N * M then P | N or P | M.
```

1.6.3 Remplir les lacunes

Malgré notre confiance en la validité du Texte sur la racine carrée, le système SAD ne pourra pas le vérifier. Les capacités déductives du vérificateur, même avec l'aide d'un démonstrateur automatique puissant, ne sont pas suffisantes pour établir d'une façon automatique la validité de la première affirmation dans la démonstration du lemme PDP : «`p divides (n - p) * m`», la quatrième ligne de la démonstration. En fait, presque aucune affirmation dans le Texte ne sera vérifiée dans sa forme actuelle.

Généralement, il y a plusieurs moyens de faire une affirmation plus «évidente» pour le vérificateur. Le plus simple est de la fournir avec une liste de *références* : des marques de sections haut-niveau à utiliser dans la vérification. Le reste de faits haut-niveau (axiomes, définitions, lemmes) sera exclu de la recherche de preuve. Si votre problème inclut un vaste ensemble des préliminaires, ce filtrage explicite des prémisses peut être bien utile. Notez qu'une section mentionnée dans les références n'est pas obligée de contribuer actuellement dans la démonstration trouvée.

Cependant, la liste exacte des prémisses nécessaires peut être très difficile à établir (en effet, l'auteur doit prévoir la démonstration bien en détail) et parfois assez long, ce qui rendra le texte peu naturel. D'ailleurs, il se peut que même avec les prémisses bien filtrées, le démonstrateur n'arrive pas à trouver la démonstration. Le pas de démonstration peut se trouver trop large.

Une autre façon de soulager le vérificateur est d'insérer une assertion intermédiaire qui est plus facile à démontrer que l'affirmation en question (notons la $\mathbb{A}$) et qui peut raccourcir la déduction de $\mathbb{A}$. L'auteur peut rédiger une sous-démonstration pour $\mathbb{A}$ et y mettre pas une seule assertion mais tout un raisonnement supplémentaire. La règle générale ici est suivante : si les propositions ajoutées ne sont utiles que pour démonstration de $\mathbb{A}$, mettez-les dans une sous-démonstration ; s'ils peuvent aider à vérifier aussi d'autres sections dans la démonstration courante, mettez-les au-dessus de $\mathbb{A}$, pour qu'ils deviennent prédécesseurs pour le reste de la démonstration ; si ces propositions apparaissent plusieurs fois dans votre démonstration dans les instances différentes, alors généralisez-les et mettez-les dans un lemme haut-niveau.

Le niveau de détail nécessaire s'établit d'habitude par essais et erreurs, jusqu'à ce que l'affirmation problématique est vérifiée. Bien entendu, ce niveau dépend fortement de vérificateur et démonstrateur utilisés. Ainsi, une certaine connaissance des mécanismes du vérificateur peut bien aider dans la préparation d'un texte vérifiable. Dans les cas difficiles, l'auteur peut en être réduit à réviser la démonstration entière et l'ensemble des préliminaires.

Considérons le premier cas de preuve du lemme PDP dans son état final, vérifiable :

```
Case p <= n.
  Let us show that p divides (n - p) * m and n - p < n.
    n = p + (n - p) and n * m = p * ((n * m) / p).
    n * m = (p * m) + ((n - p) * m) (by AMDistr).
    p divides (n - p) * m (by DefDiv, DivMin).
  end.
  Then p divides n - p or p divides m (by IH).
  Indeed ((n - p) + m) + p < (n + m) + p (by MonAdd).
  If p divides n - p then p divides n (by DefDiv,DivSum).
end.
```

Comparons la section ci-dessus avec son prototype dans le Texte sur la racine carrée. Elle est composée des même trois affirmations qui sont maintenant munies de sous-démonstrations et références. Notez que les tâches purement algébriques se trouvent assez difficiles pour un démonstrateur générique du premier ordre (surtout en présence des axiomes d'associativité, commutativité et distributivité). Ainsi, il nous a fallu ajouter trois affirmations intermédiaires pour déduire $p|(n-p)m$ de $p|nm$.

L'autre sous-but, $n - p < n$, ne réclame aucun soutien de notre part. Par contre, le système n'arrive pas à déduire de lui-même $((n-p)+m)+p < (n+m)+p$ de $n - p < n$, et dons nous mettons cette inégalité explicitement dans une sous-démonstration courte, pour pouvoir appliquer l'hypothèse d'induction.

Des autre affirmations «difficiles» dans le Texte sont traitées de façon similaire. Au bout de travail, la démonstration du lemme PDP devient à peu près deux fois plus longue. Par contre,

la démonstration du théorème principal ne s'agrandit pas, car nous pouvons la rendre vérifiable juste en ajoutant les références nécessaires :

```
Theorem Main.  Let p be a prime natural number.
  For no rational number q the square of q is p.
Proof by contradiction.
  Let q be a rational number such that q * q = p.
  Take relatively prime natural numbers n,m such that q * m = n.
  Then p * (m * m) = (n * n) (by MulAsso,MulComm).
  Hence p divides n * n and p divides n (by DefDiv,PDP).
  Choose a natural number k such that n = p * k (by DefDiv).
  Then we have p * (m * m) = p * (k * n) (by MulAsso).
  The square of m is equal to (p * k) * k (by MulComm,MulCanc).
  Hence p divides m * m and p divides m (by MulAsso,DefDiv,PDP).
  We have a contradiction (by DefRelPr).
qed.
```

Dans son état final, le texte complet se compose de 205 lignes : 126 lignes des préliminaires et 79 lignes pour le lemme PDP et le théorème principal. Le texte contient 95 buts (y compris les lemmes dans les préliminaires) et ils est entièrement vérifié par SAD en tandem avec le démonstrateur SPASS [76]. Les statistiques sur les textes et les démonstrateurs différents sont données dans l'annexe E.

Chapitre 2

Texte correct

2.1 Introduction

Le texte mathématique (que nous cherchons à approximer avec le texte en ForTheL) est un objet complexe contenant des axiomes, des définitions, des théorèmes et des démonstrations de formes variées. Qu'est-ce que signifie que un tel objet est «correct» ?

La sémantique d'un texte peut être rendue par une seule proposition, l'image-formule de ce texte. Ensuite, on pourrait déclarer le texte correct si son image-formule est démontrable dans la logique de base. Cette approche est simple et théoriquement transparente mais absolument pas pratique. La notion précise d'un texte correct qu'on obtient de cette façon peut difficilement être considérée comme spécification formelle d'un vérificateur. C'est pourquoi nous développons une notion spécifique de *correction*[1] qui, quoi que moins immédiate, d'une part peut être formalisée à l'aide d'un calcul logique et d'autre part peut servir comme spécification.

Dans notre approche nous distinguons *correction logique* et *ontologique* d'un texte formel.

Un texte *logiquement correct* est celui qu'on pourrait appeler «fondé» : tout assertion dans le texte peut être déduite de ses prédécesseurs logiques.

Un texte *ontologiquement correct* est celui qu'on pourrait appeler «sensé». Rien ne provient de nulle part et tout est utilisé proprement : tout symbole non-logique qui intervient dans le texte est introduit quelque part plus haut (dans ForTheL : par une définition ou extension de signature) et employé en dedans du domaine établi par l'introduction (dans ForTheL : satisfait les assomptions de la définition ou extension de signature). Autrement dit, tous les termes et formules dans un texte ontologiquement correct sont bien définis.

Correction ontologique est étroitement liée à la correction des types dans les langage typés (surtout dans les systèmes faiblement typés tels que WTT [34]). Elle permet d'apercevoir des erreurs de formalisation qui seraient autrement difficiles à détecter. En effet, une violation accidentelle de correction ontologique implique presque toujours incorrection logique : des assertions fausses ou indémontrable. Et il est beaucoup plus difficile de «tracer» l'échec du démonstrateur vers une occurrence mal formée que de la découvrir dès le début.

Outre cela, pendant le contrôle ontologique nous obtenons l'information sur l'applicabilité des définitions, extensions de signature et autres faits préliminaires dans les positions particulières dans le texte considéré. Tant que les transformations ultérieures, pendant la vérification logique, préservent la correction ontologique et les autres propriétés locales nous pouvons appliquer des définitions et des lemmes sans répéter les vérifications.

Notre approche peut être regardée comme un moyen de traiter des relations et fonctions partielles dans un texte mathématique. Au lieu d'introduire des individus spéciaux et de nouvelles valeurs propositionnelles (comme dans la logique forte de Kleene [37]), la correction ontologique exige que tout terme et tout atome sont bien défini a priori. En utilisant des techniques déductives qui préservent des propriétés locales, nous pouvons assurer que le texte en question reste toujours bien défini. À notre avis, cela correspond bien à la pratique habituelle. Il convient de mentionner dans ce rapport l'article [79] où une approche pareille (quoi que réalisée autrement) est argumentée.

[1] ici et par la suite nous appelons «correction de texte» la propriété d'un texte d'être correct.

Pourtant, il y a une difficulté théorique. Comment définir la correction ontologique formellement, étant donné qu'elle réfère aux propriétés des occurrences particulières des formules atomiques et des termes à l'intérieur de propositions complexes ?

Considérons une formule de la forme $(\cdots \forall x\,(x \in \mathbb{R}^{+*} \supset (\cdots \frac{xy}{x} \cdots)) \cdots)$. Il est bien évident que la fraction est bien définie et on peut la réduire à y mais comment le justifier ? La tâche telle quelle paraît absurde : tant qu'un terme contient des variables bornées, nous n'en pouvons pas raisonner. Dans la façon traditionnelle, on doit décomposer la formule jusqu'au quantificateur universel sur x, appliquer une substitution ou skolemization, séparer $x \in \mathbb{R}^+$, et ce n'est qu'après cela qu'on peut réduire la fraction. Néanmoins, nous voudrions nous assurer qu'une formule est ontologiquement correcte avant de l'admettre dans une démonstration.

Bien que la proposition «x n'est pas égal à zéro» est sûrement insensée, on peut dire que «x n'est pas égal à zéro dans cet occurrence de $\frac{xy}{x}$». L'intuition suggère que en dehors de la notion habituelle de validité, une certaine *validité locale* d'une proposition peut être définie par rapport à une position dans une formule. Une proposition qui est généralement fausse ou dénuée de sens peut devenir localement valide en étant considérée dans un contexte approprié. Dans ce qui suit, nous appelons une telle proposition une *propriété locale* de la position en question.

Dans ce chapitre, nous introduisons et étudions des propriétés locales. Ensuite nous utilisons nos résultats pour décrire des transformations des formules «admissibles» : celles qui préservent des propriétés locales dans les formules transformées. Finalement, nous définissons la correction ontologique et logique d'un texte formel.

2.2 Fondement mathématique

2.2.1 Préliminaires

Nous considérons un langage du premier ordre sans sortes dont la signature consiste en trois ensembles de symboles disjoints — symboles de fonctions, de prédicats et de notions. Les symboles logique du langage sont l'égalité ($\approx$), l'appartenance à une classe (ε), les connecteurs propositionnels standard $\neg$, $\wedge$, $\vee$, $\supset$, $\equiv$ et les quantificateurs $\forall$ et $\exists$.

Les termes dans ce langage sont les termes ordinaires du premier ordre. Les formules atomiques sont de cinq formes possibles : $\top$, $\mathfrak{T}$, $s_1 \approx s_2$, $P(s_1, \ldots, s_n)$ et $s\,\varepsilon\,N(s_1, \ldots, s_n)$, où $\top$ dénote la vérité, $\mathfrak{T}$ dénote la variable propositionnelle `thesis`, $s, s_1, \ldots, s_n$ sont des termes, P est un symbole de prédicat et N est un symbole de notion. Rappelons que les notions désignent des classes (paramétrées) d'objets, et donc la formule atomique $s\,\varepsilon\,N(s_1, \ldots, s_n)$ exprime que le terme s appartient à la classe particulière $N(s_1, \ldots, s_n)$. Rappelons aussi que la variable `thesis` est «teneuse de place» pour la *thèse courante* — la proposition qui est démontrée dans cette partie du texte (voir les sections 1.3.5 et 2.3.2).

Comme l'ordre respectif des sous-formules est important pour les définitions suivantes, nous considérons $F \wedge G$ et $G \wedge F$ comme des formules différentes (de même pour $\vee$, $\equiv$ et $\approx$).

L'expression $F \sim G$ est l'abréviation de $(F \supset G) \wedge (G \supset F)$. Nous écrivons l'égalité niée $\neg(s_1 \approx s_2)$ comme $s_1 \not\approx s_2$ et la vérité niée $\neg\top$ comme $\bot$.

Le *complément* d'une formule H, dénoté $H^\neg$, est défini comme suit :

$$
\begin{aligned}
(F \equiv G)^\neg &= (F \sim G)^\neg & \qquad (F \wedge G)^\neg &= \neg F \vee \neg G \\
(F \supset G)^\neg &= F \wedge \neg G & \qquad (F \vee G)^\neg &= \neg F \wedge \neg G \\
(\forall x\, F)^\neg &= \exists x\, \neg F & \qquad (\neg F)^\neg &= F \\
(\exists x\, F)^\neg &= \forall x\, \neg F & \qquad A^\neg &= \neg A
\end{aligned}
$$

Nous considérons les substitutions comme des fonctions de variables aux termes. Pour toute substitution ϕ, si $x\phi$ est diffèrent de x, le terme $x\phi$ est appelé le *substitut* de x dans ϕ. Une substitution est *finie* si l'ensemble des ses substituts est fini. Nous écrivons des substitutions finies comme cortèges $[t_1/x_1, \ldots, t_n/x_n]$.

Une *position* est un mot dans l'alphabet $\{0, 1, \ldots\}$. Les positions sont notées par les lettres grecques π, τ, μ, ν et ω ; la lettre ϵ dénote la position nulle (le mot vide). Les positions désignent des sous-formules et des sous-termes particuliers dans une formule ou un terme.

L'ensemble des positions d'une formule atomique ou d'un terme E, noté $\Pi(E)$, est défini comme suit ($i.\Pi$ dénote $\{\, i.\tau \mid \tau \in \Pi \,\}$) :

$$\Pi(s_0 \,\varepsilon\, N(s_1, \ldots, s_n)) = \{\epsilon\} \cup \bigcup i.\Pi(s_i) \qquad \Pi(s \approx t) = \{\epsilon\} \cup 0.\Pi(s) \cup 1.\Pi(t)$$
$$\Pi(P(s_0, \ldots, s_n)) = \{\epsilon\} \cup \bigcup i.\Pi(s_i) \qquad \Pi(\top) = \{\epsilon\}$$
$$\Pi(f(s_0, \ldots, s_n)) = \{\epsilon\} \cup \bigcup i.\Pi(s_i) \qquad \Pi(\mathfrak{T}) = \{\epsilon\}$$

L'ensemble des positions d'une formule H, noté $\Pi(H)$, est l'union disjointe :

$$\Pi(F) = \Pi^+(F) \sqcup \Pi^-(F) \sqcup \Pi^\circ(F)$$

de l'ensemble des *positions positives* $\Pi^+(H)$, de l'ensemble des *positions négatives* $\Pi^-(H)$ et de l'ensemble des *positions neutres* $\Pi^\circ(H)$ (dans la suite, A dénote une formule atomique) :

$$\Pi^+(F \equiv G) = \{\epsilon\} \qquad \Pi^+(F \supset G) = \{\epsilon\} \cup 0.\Pi^-(F) \cup 1.\Pi^+(G)$$
$$\Pi^+(F \wedge G) = \{\epsilon\} \cup 0.\Pi^+(F) \cup 1.\Pi^+(G) \qquad \Pi^+(F \vee G) = \{\epsilon\} \cup 0.\Pi^+(F) \cup 1.\Pi^+(G)$$
$$\Pi^+(\forall x\, F) = \{\epsilon\} \cup 0.\Pi^+(F) \qquad \Pi^+(\exists x\, F) = \{\epsilon\} \cup 0.\Pi^+(F)$$
$$\Pi^+(\neg F) = \{\epsilon\} \cup 0.\Pi^-(F) \qquad \Pi^+(A) = \Pi(A)$$

$$\Pi^-(F \equiv G) = \varnothing \qquad \Pi^-(F \supset G) = 0.\Pi^+(F) \cup 1.\Pi^-(G)$$
$$\Pi^-(F \wedge G) = 0.\Pi^-(F) \cup 1.\Pi^-(G) \qquad \Pi^-(F \vee G) = 0.\Pi^-(F) \cup 1.\Pi^-(G)$$
$$\Pi^-(\forall x\, F) = 0.\Pi^-(F) \qquad \Pi^-(\exists x\, F) = 0.\Pi^-(F)$$
$$\Pi^-(\neg F) = 0.\Pi^+(F) \qquad \Pi^-(A) = \varnothing$$

$$\Pi^\circ(F \equiv G) = 0.\Pi(F) \cup 1.\Pi(G) \qquad \Pi^\circ(F \supset G) = 0.\Pi^\circ(F) \cup 1.\Pi^\circ(G)$$
$$\Pi^\circ(F \wedge G) = 0.\Pi^\circ(F) \cup 1.\Pi^\circ(G) \qquad \Pi^\circ(F \vee G) = 0.\Pi^\circ(F) \cup 1.\Pi^\circ(G)$$
$$\Pi^\circ(\forall x\, F) = 0.\Pi^\circ(F) \qquad \Pi^\circ(\exists x\, F) = 0.\Pi^\circ(F)$$
$$\Pi^\circ(\neg F) = 0.\Pi^\circ(F) \qquad \Pi^\circ(A) = \varnothing$$

Nous mettons $\Pi^+(t) = \Pi(t)$ et $\Pi^-(t) = \Pi^\circ(t) = \varnothing$ pour tout terme t.

Parmi des positions, nous distinguons celles de formules ($\Pi_{\mathbf{F}}$), celles de formules atomiques ($\Pi_{\mathbf{A}}$), et celles de termes ($\Pi_{\mathbf{t}}$). Certainement, $\Pi(F) = \Pi_{\mathbf{t}}(F) \cup \Pi_{\mathbf{F}}(F)$, $\Pi_{\mathbf{A}}(t) = \Pi_{\mathbf{F}}(t) = \varnothing$, $\Pi_{\mathbf{A}}(F) \subseteq \Pi_{\mathbf{F}}(F)$, $\Pi(t) = \Pi_{\mathbf{t}}(t)$. Nous divisons les ensembles $\Pi_{\mathbf{t}}$, $\Pi_{\mathbf{A}}$ et $\Pi_{\mathbf{F}}$ dans les parties positives, négatives et neutres, aussi.

Pour une formule H et une position $\pi \in \Pi(H)$, la position $\widehat{\pi}$ est le préfixe maximal π dans $\Pi_{\mathbf{F}}(H)$. Dans la suite, nous utilisons souvent cette conversion pour étendre nos définitions de $\Pi_{\mathbf{F}}$ sur Π.

Une formule ou un terme E couplé avec une position $\tau \in \Pi(E)$ forme une *occurrence*.

Une position τ est une *sous-position* de π si $\tau = \pi.i.\mu$ pour quelque $i \in \{0, 1, \ldots\}$ et une position μ. Appelons τ une *sous-position immédiate* si μ est nulle, c'est-à-dire que $\tau = \pi.i$. Toute position qui n'est pas une sous-position de π est appelée *externe* par rapport à π. Remarquez que toute position est externe par rapport à soi-même.

Disons que π est un *prédécesseur textuel* de τ si $\pi = \omega.i.\mu$ et $\tau = \omega.j.\eta$ et $i < j$. Si $\mu = \epsilon$, nous disons que π est un *prédécesseur logique* de τ.

Soit E une formule ou un terme et τ une position dans $\Pi(E)$. Nous notons $E|_\tau$ la sous-formule ou le sous-terme intervenant dans cette position. Ci-dessous et par la suite, $(*F)$ désigne $(\neg F)$, $(\forall x\, F)$ ou $(\exists x\, F)$; et $(F * G)$ désigne $(F \equiv G)$, $(F \supset G)$, $(F \wedge G)$ ou $(F \vee G)$:

$$F|_\epsilon = F \qquad (* F)|_{0.\tau} = F|_\tau$$
$$(s_0 \,\varepsilon\, N(s_1, \ldots, s_n))|_{i.\tau} = s_i|_\tau \qquad (F * G)|_{0.\tau} = F|_\tau$$
$$P(s_0, \ldots, s_n)|_{i.\tau} = s_i|_\tau \qquad (F * G)|_{1.\tau} = G|_\tau$$
$$f(s_0, \ldots, s_n)|_{i.\tau} = s_i|_\tau \qquad (s \approx t)|_{0.\tau} = s|_\tau$$
$$t|_\epsilon = t \qquad (s \approx t)|_{1.\tau} = t|_\tau$$

Soit E une formule ou un terme, τ une position dans $\Pi(E)$ et e une formule ou un terme. Nous notons $E[e]_\tau$ le résultat de *remplacement* de $E|_\tau$ par e. Bien sûr, e doit être un terme si $\tau \in \Pi_{\mathbf{t}}(E)$, et une formule autrement :

$$\begin{aligned}
E[e]_\epsilon &= e & (*\,F)[e]_{0.\tau} &= *\,F[e]_\tau \\
(s_0\,\varepsilon\,N(s_1,\ldots,s_n))[e]_{0.\tau} &= s_0[e]_\tau\,\varepsilon\,N(s_1,\ldots,s_n) & (F*G)[e]_{0.\tau} &= F[e]_\tau * G \\
(s_0\,\varepsilon\,N(s_1,\ldots,s_n))[e]_{i.\tau} &= s_0\,\varepsilon\,N(s_1,\ldots,s_i[p]_\tau,\ldots,s_n) & (F*G)[e]_{1.\tau} &= F * G[e]_\tau \\
P(s_0,\ldots,s_n)[e]_{i.\tau} &= P(s_0,\ldots,s_i[p]_\tau,\ldots,s_n) & (s\approx t)[e]_{0.\tau} &= s[e]_\tau \approx t \\
f(s_0,\ldots,s_n)[e]_{i.\tau} &= f(s_0,\ldots,s_i[p]_\tau,\ldots,s_n) & (s\approx t)[e]_{1.\tau} &= s \approx t[e]_\tau
\end{aligned}$$

Notez que les variables libres dans e peuvent devenir bornées dans $F[e]_\tau$.

2.2.2 Validité locale et propriétés locales

Soit F et U des formules et π une position dans $\Pi_{\mathbf{F}}(F)$. Nous définissons *l'image locale* de U dans F, π, notée $\langle U\rangle^F_\pi$, comme suit :

$$\begin{aligned}
\langle U\rangle^{F\equiv G}_{0.\pi} &= \langle U\rangle^F_\pi & \langle U\rangle^{F\equiv G}_{1.\pi} &= \langle U\rangle^G_\pi & \langle U\rangle^{\forall x F}_{0.\pi} &= \forall x\,\langle U\rangle^F_\pi \\
\langle U\rangle^{F\supset G}_{0.\pi} &= G \vee \langle U\rangle^F_\pi & \langle U\rangle^{F\supset G}_{1.\pi} &= F \supset \langle U\rangle^G_\pi & \langle U\rangle^{\exists x F}_{0.\pi} &= \forall x\,\langle U\rangle^F_\pi \\
\langle U\rangle^{F\wedge G}_{0.\pi} &= G \supset \langle U\rangle^F_\pi & \langle U\rangle^{F\wedge G}_{1.\pi} &= F \supset \langle U\rangle^G_\pi & \langle U\rangle^{\neg F}_{0.\pi} &= \langle U\rangle^F_\pi \\
\langle U\rangle^{F\vee G}_{0.\pi} &= G \vee \langle U\rangle^F_\pi & \langle U\rangle^{F\vee G}_{1.\pi} &= F \vee \langle U\rangle^G_\pi & \langle U\rangle^F_\varepsilon &= U
\end{aligned}$$

La formule $\langle U\rangle^F_\pi$ exprime la proposition «U est localement valide dans la position π de la formule F». Notez que cette formule ne dépend pas de la sous-formule $F|_\pi$. Notez aussi qu'un quantificateur existentiel dans F produit un quantificateur universel dans l'image locale. Ce n'est pas accidentel : une proposition sur une variable bornée doit être valide pour toute valeur possible de la variable indépendamment de quantificateur. C'est pour la même raison que la négation n'est pas prise en compte pendant la construction de l'image.

Pour toute position $\pi \in \Pi_{\mathbf{t}}(F)$, nous définissons $\langle U\rangle^F_\pi$ comme $\langle U\rangle^F_{\widehat{\pi}}$.

Exemple 2.2.1. Soit F la formule

$$\forall n\,(n\in\mathbb{N}\ \supset\ \forall x\,(x\approx \mathbf{fib}(n)\ \equiv\ (x\in\mathbb{N}\wedge \\ \wedge\ ((n\le 1\ \wedge\ x\approx 1)\ \vee\ x\approx(\mathbf{fib}(n-1)+\mathbf{fib}(n-2))))))$$

Cette formule représente une définition récursive. Nous voulons vérifier que les arguments $(n-1)$ et $(n-2)$ satisfont les conditions de la définition et qu'ils sont strictement inférieurs à n.

Considérons le deuxième terme. Dénotons par π sa position, 0.1.0.1.1.1.1.1.0. Nous devons démontrer l'image locale $\langle (n-2)\in\mathbb{N}\ \wedge\ (n-2)<n\rangle^F_\pi$. Cette formule est équivalente à

$$\forall n\,(n\in\mathbb{N}\ \supset\ \forall x\,(x\in\mathbb{N}\ \supset\ ((n\le 1\ \wedge\ x\approx 1)\ \vee\ ((n-2)\in\mathbb{N}\ \wedge\ (n-2)<n))))$$

Mais cette dernière formule est fausse pour $n=x=0$! Cela montre une faute dans notre définition : le cas $x=0$ falsifie le côté gauche de la disjonction $F|_{0.1.0.1.1}$, et nous devons considérer la partie droite avec $n=0$ pour évaluer la validité de la disjonction entière. Maintenant il est facile de construire une bonne définition F' :

$$\forall n\,(n\in\mathbb{N}\ \supset\ \forall x\,(x\approx \mathbf{fib}(n)\ \equiv\ (x\in\mathbb{N}\wedge \\ \wedge\ ((n\le 1\ \wedge\ x\approx 1)\ \vee\ (n\ge 2\ \wedge\ x\approx(\mathbf{fib}(n-1)+\mathbf{fib}(n-2)))))))$$

Évidemment, l'image locale $\langle (n-2)\in\mathbb{N}\ \wedge\ (n-2)<n\rangle^{F'}_{0.1.0.1.1.1.1.1.1.0}$ est la formule valide :

$$\forall n\,(n\in\mathbb{N}\ \supset\ \forall x\,(x\in\mathbb{N}\ \supset \\ \supset\ ((n\le 1\ \wedge\ x\approx 1)\ \vee\ (n\ge 2\ \supset\ ((n-2)\in\mathbb{N}\ \wedge\ (n-2)<n)))))$$

Lemme 2.2.2. *Pour toutes F, $\pi \in \Pi(F)$ et une formule close U, $U \vdash \langle U \rangle_\pi^F$.*

Démonstration. La formule $\langle U \rangle_\pi^F$ est équivalente à une disjonction universelle dont U est une partie. □

Lemme 2.2.3. (a) $\vdash \langle U \supset V \rangle_\pi^F \supset (\langle U \rangle_\pi^F \supset \langle V \rangle_\pi^F)$ (b) $\vdash \langle \forall x\, U \rangle_\pi^F \supset \langle U[t/x] \rangle_\pi^F$

Les deux assertions du lemme 2.2.3 peuvent être démontrées par une induction sur la longueur de π, de la même façon que dans la démonstration du théorème 2.2.7 plus bas.

Les lemmes ci-dessus montrent que pouvons raisonner sur des propriétés locales d'une manière cohérente. Ils sont suffisamment puissants pour démontrer quelque corollaires intéressants.

Corollaire 2.2.4. $\vdash \langle U \equiv V \rangle_\pi^F \supset (\langle U \rangle_\pi^F \equiv \langle V \rangle_\pi^F)$

Démonstration. D'après le lemme 2.2.2, $\vdash \langle (U \equiv V) \supset (U \supset V) \rangle_\pi^F$. Ainsi, par le lemme 2.2.3(a), $\vdash \langle (U \equiv V) \rangle_\pi^F \supset (\langle U \supset V \rangle_\pi^F)$. Encore par ce lemme, $\vdash \langle (U \equiv V) \rangle_\pi^F \supset (\langle U \rangle_\pi^F \supset \langle V \rangle_\pi^F)$. De la même manière, nous obtenons $\vdash \langle (U \equiv V) \rangle_\pi^F \supset (\langle V \rangle_\pi^F \supset \langle U \rangle_\pi^F)$. □

Corollaire 2.2.5. $\vdash \langle U \wedge V \rangle_\pi^F \equiv (\langle U \rangle_\pi^F \wedge \langle V \rangle_\pi^F)$

Démonstration. Pour démontrer la nécessité, nous prenons les tautologies propositionnelles $(U \wedge V) \supset U$ et $(U \wedge V) \supset V$. Pour la suffisance, prenons la tautologie $U \supset (V \supset (U \wedge V))$. Nous «enfonçons» une tautologie choisie dans la formule F d'après le lemme 2.2.2 et appliquons le *modus ponens* local, le lemme 2.2.3(a). □

Corollaire 2.2.6. *Pour toute «forme» $C[\ ]$ sans quantificateurs,*

$$\vdash \big(\langle U_1 \equiv V_1 \rangle_\pi^F \wedge \cdots \wedge \langle U_n \equiv V_n \rangle_\pi^F \wedge \langle t_1 \approx s_1 \rangle_\pi^F \wedge \cdots \wedge \langle t_m \approx s_m \rangle_\pi^F\big) \supset \\ \supset \langle C[U_1, \ldots, U_n, t_1, \ldots, t_m] \equiv C[V_1, \ldots, V_n, s_1, \ldots, s_m] \rangle_\pi^F$$

Ici, le terme «forme» désigne une formule avec des «trous» dans lesquelles des formules ou termes peuvent se mettre, ainsi complétant la forme jusqu'à une formule bien formée. Le corollaire peut être démontré de la même façon que les affirmations précédentes.

La propriété principale des images locales est donnée par le théorème suivant.

Théorème 2.2.7. *Pour toutes formules F, U, V*

$$\begin{aligned} \pi \in \Pi_{\mathbf{F}}(F) \quad &\Rightarrow \quad \vdash \langle U \equiv V \rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi) \\ \pi \in \Pi_{\mathbf{F}}^+(F) \quad &\Rightarrow \quad \vdash \langle U \supset V \rangle_\pi^F \supset (F[U]_\pi \supset F[V]_\pi) \\ \pi \in \Pi_{\mathbf{F}}^-(F) \quad &\Rightarrow \quad \vdash \langle V \supset U \rangle_\pi^F \supset (F[U]_\pi \supset F[V]_\pi) \end{aligned}$$

Démonstration. Nous allons démontrer la première assertion seulement, pour les deux autres le raisonnement est pareil. Nous procédons par induction sur la longueur de π. Dans le cas de base ($\pi = \epsilon$), l'assertion est évidente. Sinon, considérons quatre cas principaux. Dans chaque cas nous déduisons formellement la proposition cherchée de l'hypothèse d'induction par la définition de l'image locale.

Cas $F = (\neg G)$, $\pi = 0.\tau$ (pareil pour $F = (G \equiv H)$) :

$$\begin{aligned} \vdash \langle U \equiv V \rangle_\tau^G \supset (G[U]_\tau \equiv G[V]_\tau) \quad &\Rightarrow \\ \Rightarrow \quad \vdash \langle U \equiv V \rangle_\tau^G \supset (\neg G[U]_\tau \equiv \neg G[V]_\tau) \quad &\Rightarrow \\ \Rightarrow \quad \vdash \langle U \equiv V \rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi) & \end{aligned}$$

Cas $F = (G \supset H)$, $\pi = 0.\tau$ (pareil pour $F = (G \vee H)$) :

$$\begin{aligned} \vdash \langle U \equiv V \rangle_\tau^G \supset (G[U]_\tau \equiv G[V]_\tau) \quad &\Rightarrow \\ \Rightarrow \quad \vdash (H \vee \langle U \equiv V \rangle_\tau^G) \supset (H \vee (G[U]_\tau \equiv G[V]_\tau)) \quad &\Rightarrow \\ \Rightarrow \quad \vdash \langle U \equiv V \rangle_\pi^F \supset ((G[U]_\tau \supset H) \equiv (G[V]_\tau \supset H)) \quad &\Rightarrow \\ \Rightarrow \quad \vdash \langle U \equiv V \rangle_\pi^F \supset (F[U]_\pi \equiv F[V]_\pi) & \end{aligned}$$

Cas $F = (H \supset G)$, $\pi = 1.\tau$ (pareil pour $F = (G \wedge H)$) :

$$\begin{aligned}
\vdash \langle U \equiv V\rangle^G_\tau \supset (G[U]_\tau \equiv G[V]_\tau) \quad &\Rightarrow \\
\Rightarrow \quad \vdash (H \supset \langle U \equiv V\rangle^G_\tau) \supset (H \supset (G[U]_\tau \equiv G[V]_\tau)) \quad &\Rightarrow \\
\Rightarrow \quad \vdash \langle U \equiv V\rangle^F_\pi \supset ((H \supset G[U]_\tau) \equiv (H \supset G[V]_\tau)) \quad &\Rightarrow \\
\Rightarrow \quad \vdash \langle U \equiv V\rangle^F_\pi \supset (F[U]_\pi \equiv F[V]_\pi)&
\end{aligned}$$

Cas $F = (\exists x\, G)$, $\pi = 0.\tau$ (pareil pour $F = (\forall x\, G)$) :

$$\begin{aligned}
\vdash \langle U \equiv V\rangle^G_\tau \supset (G[U]_\tau \equiv G[V]_\tau) \quad &\Rightarrow \\
\Rightarrow \quad \vdash \forall x\, \langle U \equiv V\rangle^G_\tau \supset \forall x\, (G[U]_\tau \equiv G[V]_\tau) \quad &\Rightarrow \\
\Rightarrow \quad \vdash \langle U \equiv V\rangle^F_\pi \supset (\exists x\, G[U]_\tau \equiv \exists x\, G[V]_\tau) \quad &\Rightarrow \\
\Rightarrow \quad \vdash \langle U \equiv V\rangle^F_\pi \supset (F[U]_\pi \equiv F[V]_\pi)&
\end{aligned}$$

□

Ainsi, nous avons droit de remplacer des sous-formules non seulement par des formules équivalentes mais aussi par celles qui sont équivalentes localement. Notons que l'inverse du théorème 2.2.7 est vrai en logique propositionnelle : $\vdash_0 \langle U \equiv V\rangle^F_\pi \equiv (F[U]_\pi \equiv F[V]_\pi)$. Ce n'est pas le cas en logique du premier ordre où, par exemple, $(\exists x\, x \approx 0)$ est équivalent à $(\exists x\, x \approx 1)$.

Corollaire 2.2.8. *Soit F une formule, $\pi \in \Pi_{\mathbf{t}}(F)$ une position de terme et s, t des termes quelconques. Alors,*

$$\vdash \langle s \approx t\rangle^F_\pi \supset (F[s]_\pi \equiv F[t]_\pi)$$

Démonstration. D'après le théorème 2.2.7 et le corollaire 2.2.6. □

Corollaire 2.2.9. *Soit F une formule, $\pi \in \Pi_{\mathbf{F}}(F)$ une position de sous-formule, et U, V des formules quelconques. Alors,*

$$\begin{aligned}
\vdash \langle U\rangle^F_\pi \supset (F[V]_\pi \equiv F[U \wedge V]_\pi) \qquad & \vdash \langle U\rangle^F_\pi \supset (F[V]_\pi \equiv F[U \supset V]_\pi) \\
\vdash \langle V \supset U\rangle^F_\pi \supset (F[V]_\pi \equiv F[U \wedge V]_\pi) \qquad & \vdash \langle U \supset V\rangle^F_\pi \supset (F[V]_\pi \equiv F[U \vee V]_\pi)
\end{aligned}$$

Considérons une formule close H de la forme $\forall \vec{x}\, (C \supset (A \equiv D))$, où A est une formule atomique. Soit F une formule et $\pi \in \Pi_{\mathbf{A}}(F)$ une position d'atome telle que $F|_\pi = A\sigma$ pour quelque substitution σ. Si on peut démontrer $\langle C\sigma\rangle^F_\pi$, on obtient $\langle A\sigma \equiv D\sigma\rangle^F_\pi$ par les lemmes 2.2.2 et 2.2.3 (étant donné que H est parmi les prémisses). Alors nous pouvons remplacer $A\sigma$ par $D\sigma$ par le théorème 2.2.7.

Dans l'exemple 2.2.1, nous pouvons garantir qu'une telle application de définition est toujours possible (comme $\langle n - 1 \in \mathbb{N} \wedge n - 2 \in \mathbb{N}\rangle^F_\pi$ a lieu) et n'amène jamais à une descente infinie (comme $\langle n - 1 < n \wedge n - 2 < n\rangle^F_\pi$ a lieu).

Validité locale dirigée

L'image locale introduite ci-dessus a un défaut : elle n'est pas stable par rapport aux transformations dans les positions voisines.

Exemple 2.2.10. Comme $\langle A\rangle^{A\wedge A}_0$ est vrai, $(A \wedge A)$ est équivalent à $(\top \wedge A)$ d'après le théorème 2.2.7. Mais $\langle A\rangle^{A\wedge A}_1$ est également vrai, alors que $\langle A\rangle^{\top\wedge A}_1$ est faux.

Généralement, on peut construire une formule F dont deux sous-formules U est V assurent certaines propriétés locales l'une pour l'autre. En utilisant ces propriétés, on remplace U par une formule U' localement équivalente. Mais par cela on perd des propriétés locales de V.

Cela n'aurait pas d'importance si on n'utilisait des propriétés locales que pour des transformations «en une fois», par exemple, pour des simplifications. En effet, on doit vérifier que la simplification est possible juste avant l'appliquer.

Mais laissez nous rappeler (et aussi anticiper) que la correction ontologique de formules et de parties d'un texte formel est définie en termes de propriétés locales. Outre cela, la génération d'évidences, l'application de définitions et autres méthodes du vérificateur (qui seront décrites dans le chapitre suivant) compte sur des ensembles de propriétés locales qui sont, une fois construits, assignés aux occurrences particulières. Si une transformation quelconque d'une formule contenant une telle occurrence (par exemple, insertion de son instance dans un successeur logique plus bas dans le texte — ce qui a lieu pendant application d'une définition ou d'un lemme) falsifie des propriétés locales, on serait obligé d'abandonner tout ce qu'on a su sur cette occurrence et rétablir ses propriétés locales à nouveau.

Pour éviter cela, nous corrigeons la définition de l'image locale de telle façon que les formules dans les positions précédentes seules sont pries en compte. Ça correspond bien à l'exposé traditionnel où des déclarations de types, des limites et d'autres conditions préliminaires sont habituellement écrites avant des formules «signifiantes».

L'image locale dirigée d'une formule U dans une position $\pi \in \Pi_{\mathbf{F}}(F)$ d'une formule F, dénotée $\langle\!|U|\!\rangle^F_\pi$, est définie comme suit :

$$\begin{array}{lll}
\langle\!|U|\!\rangle^{F\equiv G}_{0.\pi} = \langle\!|U|\!\rangle^F_\pi & \langle\!|U|\!\rangle^{F\equiv G}_{1.\pi} = \langle\!|U|\!\rangle^G_\pi & \langle\!|U|\!\rangle^{\forall x F}_{0.\pi} = \forall x\, \langle\!|U|\!\rangle^F_\pi \\
\langle\!|U|\!\rangle^{F\supset G}_{0.\pi} = \langle\!|U|\!\rangle^F_\pi & \langle\!|U|\!\rangle^{F\supset G}_{1.\pi} = F \supset \langle\!|U|\!\rangle^G_\pi & \langle\!|U|\!\rangle^{\exists x F}_{0.\pi} = \forall x\, \langle\!|U|\!\rangle^F_\pi \\
\langle\!|U|\!\rangle^{F\wedge G}_{0.\pi} = \langle\!|U|\!\rangle^F_\pi & \langle\!|U|\!\rangle^{F\wedge G}_{1.\pi} = F \supset \langle\!|U|\!\rangle^G_\pi & \langle\!|U|\!\rangle^{\neg F}_{0.\pi} = \langle\!|U|\!\rangle^F_\pi \\
\langle\!|U|\!\rangle^{F\vee G}_{0.\pi} = \langle\!|U|\!\rangle^F_\pi & \langle\!|U|\!\rangle^{F\vee G}_{1.\pi} = F \vee \langle\!|U|\!\rangle^G_\pi & \langle\!|U|\!\rangle^F_\varepsilon = U
\end{array}$$

Pour toute position $\pi \in \Pi_{\mathbf{t}}(F)$, nous définissons $\langle\!|U|\!\rangle^F_\pi$ comme $\langle\!|U|\!\rangle^F_{\widehat{\pi}}$.

Premièrement, notez que toutes les assertions démontrées jusqu'ici pour les images locales restent valides pour celles dirigées. En quelque sorte, une image dirigée n'est qu'une réduction, avec certaines conditions et alternatives éliminées. C'est illustré par le lemme trivial qui suit.

Lemme 2.2.11. $\quad \vdash \langle\!|U|\!\rangle^F_\pi \supset \langle U\rangle^F_\pi$

Démontrons que les images dirigées sont préservées par des transformation équivalentes.

Théorème 2.2.12. *Pour toute formule F et positions $\pi, \tau \in \Pi_{\mathbf{F}}(F)$, telles que τ est externe par rapport à π, on a :*

$$\vdash \langle\!|U \equiv V|\!\rangle^F_\pi \supset \left(\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \equiv \langle\!|W|\!\rangle^{F[V]_\pi}_\tau\right)$$

Démonstration. Nous procédons de nouveau par induction sur la longueur de π. Il est facile de voir que si τ est un prédécesseur textuel de π ou π est une sous-position de τ, alors les formules $\langle\!|W|\!\rangle^{F[U]_\pi}_\tau$ et $\langle\!|W|\!\rangle^{F[V]_\pi}_\tau$ sont identiques. Alors on peut supposer que π est un prédécesseur textuel de τ. C'est-à-dire qu'il y a des positions ω, μ et η telles que $\pi = \omega.0.\mu$ et $\tau = \omega.1.\eta$. En suivant la méthode de démonstration du théorème 2.2.7, on peut réduire le problème à :

$$\vdash \langle\!|U \equiv V|\!\rangle^{G*H}_{0.\mu} \supset \left(\langle\!|W|\!\rangle^{(G*H)[U]_{0.\mu}}_{1.\eta} \equiv \langle\!|W|\!\rangle^{(G*H)[V]_{0.\mu}}_{1.\eta}\right)$$

où $(G * H) = F|_\omega$. Cette formule est équivalente à :

$$\vdash \langle\!|U \equiv V|\!\rangle^G_\mu \supset \left(\langle\!|W|\!\rangle^{G[U]_\mu * H}_{1.\eta} \equiv \langle\!|W|\!\rangle^{G[V]_\mu * H}_{1.\eta}\right)$$

donc aussi à (étant donné que $*$ n'est pas équivalence, ce qui fait un cas trivial) :

$$\vdash \langle\!|U \equiv V|\!\rangle^G_\mu \supset \left((G[U]_\mu \dagger \langle\!|W|\!\rangle^H_\eta) \equiv (G[V]_\mu \dagger \langle\!|W|\!\rangle^H_\eta)\right)$$

où $\dagger$ est $\vee$ si $*$ est $\vee$, et $\dagger$ est $\supset$ autrement. D'après le théorème 2.2.7 et le lemme 2.2.11, l'image $\langle\!|U \equiv V|\!\rangle^G_\mu$ implique $(G[U]_\mu \equiv G[V]_\mu)$. Notre assertion est alors démontrée. □

Corollaire 2.2.13. *Pour toute formule F et positions $\pi, \tau \in \Pi_{\mathbf{t}}(F)$, telles que τ est externe par rapport à π, on a :*

$$\vdash \langle\!|s \approx t|\!\rangle^F_\pi \supset \left(\langle\!|W|\!\rangle^{F[s]_\pi}_\tau \equiv \langle\!|W|\!\rangle^{F[t]_\pi}_\tau\right)$$

Par la suite, nous utiliserons exclusivement des images locales dirigées. Nous omettons le mot «dirigé» partout sauf les cas où nous comparons les deux notions. Des propriétés locales dans le texte suivant sont ainsi les propriétés locales dirigées.

Historique

Le raisonnement à l'intérieur d'une formule n'est pas une nouvelle idée. À notre connaissance, un tel concept a été introduit pour la première fois par L.G. Monk dans [49] et plus tard élaboré dans [10]. P.J. Robinson et J. Staples [63] ont proposé une méthode déductive (appelée «inférence par fenêtres») qui opérait sur des sous-formules dans le contexte des sous-formules environnantes. Ce système d'inférence à été généralisé et étendu par J. Grundy [30].

Un trait commun aux approches mentionnées est que le contexte local d'une occurrence est représenté par un ensemble de formules qui sont regardées comme prémisses locales de l'occurrence en question. Des règles spéciales sont introduites pour manipuler ce contexte local et, pire que ça, certaines transformations «fortes», e.g. remplacer de $A \vee B$ par $\neg A \supset B$, sont nécessaires. La notion d'image locale introduite ici paraît moins gênant. En particulier, les résultats de cette section restent valides en logique intuitionniste. Par contre, nous ne voyons aucun moyen direct d'adapter pour cette logique les contextes locaux à la Monk [49].

En outre, la définition d'image locale s'étend sur le langage modal : $\langle U \rangle_{0.\pi}^{\Box F} = \Box \langle U \rangle_{\pi}^{F}$ et $\langle U \rangle_{0.\pi}^{\Diamond F} = \Box \langle U \rangle_{\pi}^{F}$ (de même pour les images dirigées). Alors nos assertions peuvent être démontrées en logique modale **K**, donc dans toute logique modale normale.

2.2.3 Transformations de formules

Dans cette section nous introduisons des transformations de formules qui sont employées dans les procédures du vérificateur. Tout au début, nous expliquerons qu'est ce que nous entendons par «transformation de formule» et quelles propriétés une telle transformation doit posséder.

Une *transformation atomique* est un remplacement dans une formule d'une sous-formule ou d'un terme avec quelque autre formule ou terme. Formellement, une transformation atomique, pour toute occurrence $F[U]_{\pi}$, retourne $F[V]_{\pi}$. Une *transformation chaînée* est une série de transformations atomiques.

Toute transformation atomique doit satisfaire certaines contraintes pour nous être utile :

(i) La formule produite par une transformation doit être en quelque relation logique avec la formule initiale : être son équivalent ou une conséquence ou, au contraire, l'impliquer ; tous cela, possiblement, en vue de certaines prémisses.

(ii) La transformation doit préserver toutes les propriétés locales dans les positions externes par rapport à π dans F. Nous avons expliqué pourquoi cette condition est importante dans la section précédente, en introduisant les images locales dirigées.

(iii) Comme la nouvelle sous-formule V a dû provenir de quelque part (de prémisses, de quelque partie de F ou de U même), les propriétés locales de «l'originale» doivent être transmises, d'une façon ou d'une autre, à V dans $F[V]_{\pi}$. Les détails spécifiques dépendent de la transformation en question.

Et en passant aux transformations chaînées :

(iv) Toutes les transformations atomiques dans la chaîne doivent être consistantes l'une avec l'autre ; c'est-à-dire que si l'une produit une conséquence logique de la formule initiale, l'autre doit donner une conséquence aussi.

Transformations équivalentes

Une *transformation équivalente* produit un équivalent logique de la formule initiale. Un résultat générique a été démontré dans la section précédente par les théorèmes 2.2.7 et 2.2.12, notamment : le remplacement par une sous-formule localement équivalente produit un résultat équivalent et préserve les propriétés locales dirigées dans les positions externes. Ainsi, toute transformation qui consiste en un tel remplacement va satisfaire automatiquement les deux premières conditions ci-dessus.

Plus bas nous considérons deux exemples utiles de transformation équivalente et nous montrons comment ils suffisent à la condition (iii).

Contraction booléenne. Soit H une formule. La *contraction booléenne (atomique)* de H, dénotée $\complement H$, est définie comme suit :

$$\begin{array}{llll}
\complement(F \equiv \top) = F & \complement(F \supset \top) = \top & \complement(F \wedge \top) = F & \complement(F \vee \top) = \top \\
\complement(\top \equiv F) = F & \complement(\top \supset F) = F & \complement(\top \wedge F) = F & \complement(\top \vee F) = \top \\
\complement(F \equiv \bot) = \neg F & \complement(F \supset \bot) = \neg F & \complement(F \wedge \bot) = \bot & \complement(F \vee \bot) = F \\
\complement(\bot \equiv F) = \neg F & \complement(\bot \supset F) = \top & \complement(\bot \wedge F) = \bot & \complement(\bot \vee F) = F \\
\complement(\forall x\, \top) = \top & \complement(\exists x\, \top) = \top & \complement(\neg\top) = \bot & \complement H = H \\
\complement(\forall x\, \bot) = \bot & \complement(\exists x\, \bot) = \bot & \complement(\neg\bot) = \top & \text{(pour toute autre } H\text{)}
\end{array}$$

La *contraction booléenne (complète)* $\overline{\complement} H$ est obtenue par application de $\complement$ partout dans H :

$$\overline{\complement}(F * G) = \complement((\overline{\complement} F) * (\overline{\complement} G)) \qquad \overline{\complement}(* F) = \complement(* (\overline{\complement} F)) \qquad \overline{\complement} A = A$$

Évidemment, la contraction complète n'est qu'une chaîne de contractions atomiques appliquées aux sous-formules.

Lemme 2.2.14. *Pour toute formule H, $\vdash H \equiv \complement H$ et $\vdash H \equiv \complement^* H$.*

La démonstration est triviale. Ainsi, la contraction booléenne est en effet une transformation équivalente conforme aux conditions (i) et (ii). Satisfaction de la troisième condition est démontrée par le lemme suivant.

Lemme 2.2.15. *Soit H une formule et π une position dans $\Pi_{\mathbf{F}}(H)$. Soit F une formule telle que $\complement F$ n'est pas une constante booléenne. Si F n'est pas de la forme $G \equiv \bot$, $\bot \equiv G$, ou $G \supset \bot$, alors pour toute position $\tau \in \Pi(\complement F)$ il y a une position $i.\tau \in \Pi(F)$ telle que $F|_{i.\tau} = (\complement F)|_\tau$ et :*

$$\vdash \langle\!| U |\!\rangle_{\pi.i.\tau}^{H[F]_\pi} \supset \langle\!| U |\!\rangle_{\pi.\tau}^{H[\complement F]_\pi}$$

Sinon, si F est de la forme $G \equiv \bot$, $\bot \equiv G$, ou $G \supset \bot$, alors pour toute position $0.\tau \in \Pi(\complement F)$ il y a une position $i.\tau \in \Pi(F)$ telle que $F|_{i.\tau} = (\complement F)|_{0.\tau}$ et :

$$\vdash \langle\!| U |\!\rangle_{\pi.i.\tau}^{H[F]_\pi} \supset \langle\!| U |\!\rangle_{\pi.0.\tau}^{H[\complement F]_\pi}$$

Démonstration. Supposons $F = \top \wedge G$ (les autres cas sont à démontrer de la même façon). Dans ce cas $\complement F = G$ et nous mettons $i = 1$. Alors l'image locale $\langle\!| U |\!\rangle_{\pi.1.\tau}^{H[F]_\pi}$ est $\langle\!| \top \supset \langle\!| U |\!\rangle_\tau^G |\!\rangle_\pi^H$ ce qui est équivalent à $\langle\!| \langle\!| U |\!\rangle_\tau^G |\!\rangle_\pi^H$, et donc à $\langle\!| U |\!\rangle_{\pi.\tau}^{H[G]_\pi}$. □

Ainsi, toute occurrence dans une formule contractée hérite ses propriétés locales de quelque occurrence correspondante dans la formule initiale.

Il est facile de voir que dans la démonstration ci-dessus, nous pouvions supposer dès le début que $\pi = \epsilon$ et ignorer la formule H, car elle produit le même contexte dans les images locales dans les deux parties de l'implication affirmée. C'est ce que nous allons faire dans les démonstrations suivantes de ce genre.

Contraction en vue. Introduisons une transformation plus intéressante. Pour cela nous aurons besoin de deux autres variétés de positions. Les ensembles de positions $\Pi_\wedge$ et $\Pi_\vee$ servent à séparer les composantes des conjonctions et des disjonctions extérieures :

$$\begin{array}{ll}
\Pi_\wedge(F \equiv G) = \{\epsilon\} & \Pi_\wedge(F \supset G) = \{\epsilon\} \\
\Pi_\wedge(F \wedge G) = \{\epsilon\} \cup 0.\Pi_\wedge(F) \cup 1.\Pi_\wedge(G) & \Pi_\wedge(F \vee G) = \{\epsilon\} \\
\Pi_\wedge(\forall x\, F) = \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\wedge(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} & \Pi_\wedge(\neg F) = \{\epsilon\} \cup 0.\Pi_\vee(F) \\
\Pi_\wedge(\exists x\, F) = \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\wedge(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} & \Pi_\wedge(A) = \{\epsilon\}
\end{array}$$

$$\Pi_\vee(F \supset G) = \{\epsilon\} \cup 0.\Pi_\wedge(F) \cup 1.\Pi_\vee(G) \qquad \Pi_\vee(F \equiv G) = \{\epsilon\}$$
$$\Pi_\vee(F \vee G) = \{\epsilon\} \cup 0.\Pi_\vee(F) \cup 1.\Pi_\vee(G) \qquad \Pi_\vee(F \wedge G) = \{\epsilon\}$$
$$\Pi_\vee(\forall x\, F) = \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\vee(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} \qquad \Pi_\vee(\neg F) = \{\epsilon\} \cup 0.\Pi_\wedge(F)$$
$$\Pi_\vee(\exists x\, F) = \{\epsilon\} \cup 0.\{\,\tau \mid \tau \in \Pi_\vee(F) \wedge x \notin \mathcal{FV}(F|_\tau)\} \qquad \Pi_\vee(A) = \{\epsilon\}$$

Comme avant, nous divisons ces ensembles en parties positives, négatives et neutres :

$$\Pi^+_\wedge(F) = \Pi_\wedge(F) \cap \Pi^+(F) \qquad \Pi^-_\wedge(F) = \Pi_\wedge(F) \cap \Pi^-(F) \qquad \Pi^\circ_\wedge(F) = \Pi_\wedge(F) \cap \Pi^\circ(F)$$
$$\Pi^+_\vee(F) = \Pi_\vee(F) \cap \Pi^+(F) \qquad \Pi^-_\vee(F) = \Pi_\vee(F) \cap \Pi^-(F) \qquad \Pi^\circ_\vee(F) = \Pi_\vee(F) \cap \Pi^\circ(F)$$

Lemme 2.2.16. *Pour toute formule F, on a :*

$$\tau \in \Pi^+_\wedge(F) \;\Rightarrow\; \vdash F \supset F|_\tau \qquad \tau \in \Pi^-_\wedge(F) \;\Rightarrow\; \vdash F \supset \neg(F|_\tau)$$
$$\tau \in \Pi^+_\vee(F) \;\Rightarrow\; \vdash F|_\tau \supset F \qquad \tau \in \Pi^-_\vee(F) \;\Rightarrow\; \vdash \neg(F|_\tau) \supset F$$

Démonstration. Peut être facilement démontré par induction sur la longueur de τ. □

Soit F et H des formules. La *contraction (atomique) de F en vue de H*, dénotée $\mathsf{C}_H\, F$, est définie comme suit :

$$\mathsf{C}_H\, F = \top \qquad \text{si } F = H|_\pi \text{ pour quelque } \pi \in \Pi^+_\wedge(H)$$
$$\mathsf{C}_H\, F = \bot \qquad \text{si } F = H|_\pi \text{ pour quelque } \pi \in \Pi^-_\wedge(H)$$
$$\mathsf{C}_H\, F = F \qquad \text{autrement}$$

La *contraction complète* $\overline{\mathsf{C}}_H\, F$ est obtenue par application de C_H partout dans F :

$$\overline{\mathsf{C}}_H\, F = \mathsf{C}_H\, F \qquad \text{si } \mathsf{C}_H\, F \text{ est une constante booléenne}$$
$$\overline{\mathsf{C}}_H(F * G) = (\overline{\mathsf{C}}_H\, F) * (\overline{\mathsf{C}}_H\, G) \qquad \text{autrement}$$
$$\overline{\mathsf{C}}_H(*F) = *(\overline{\mathsf{C}}_H\, F)$$
$$\overline{\mathsf{C}}_H\, A = A$$

Nous n'effectuons pas d'unification en comparant F avec $H|_\pi$ mais nous pouvons renommer des variables bornées si nécessaire. Des variables bornées de F ne doivent pas être confondues avec des variables libres de H, i.e. $\overline{\mathsf{C}}_{P(x)}(\exists x\, P(x))$ est égal à $\exists x\, P(x)$ et pas à $\exists x\, \top$.

En vue de H, $\overline{\mathsf{C}}_H\, F$ est obtenu à partir de F par une série de remplacements localement équivalentes (le lemme 2.2.16). Le résultat est alors équivalent à la formule initiale :

Lemme 2.2.17. *Pour toutes formules F, H, nous avons $H \vdash F \equiv \mathsf{C}_H\, F$ et $H \vdash F \equiv \overline{\mathsf{C}}_H\, F$.*

Ainsi, la contraction en vue d'une formule est une transformation équivalente. D'ailleurs, comme elle n'introduit pas de nouvelles sous-formules (sauf des constantes booléennes), nous obtenons d'après le théorème 2.2.12 le lemme suivante :

Lemme 2.2.18. *Pour toute position $\pi \in \Pi(\overline{\mathsf{C}}_H\, F)$, on a $H \vdash \langle\!|U|\!\rangle^F_\pi \supset \langle\!|U|\!\rangle^{(\overline{\mathsf{C}}_H\, F)}_\pi$.*

La contraction en vue d'une formule peut être étendue d'une manière triviale à la contraction en vue d'un ensemble de formules. Elle se combine naturellement avec la contraction booléenne pour que cette dernière élimine les constantes booléennes introduites par la première.

Remarque. Notez bien que notre façon de définir la contraction atomique $\mathsf{C}_H\, F$ n'est ni la seule possible ni la meilleure. Il y a des méthodes beaucoup plus convenables pour établir qu'une formule F ou sa négation est impliquée par une formule H que la comparaison graphique de F avec des sous-formules de H. L'important est le fait que les propriétés de C_H et $\overline{\mathsf{C}}_H$ décrites par les lemmes 2.2.17 et 2.2.18 ne dépendent pas de la méthode avec laquelle on a établi l'implication. Tant que C_H n'est que le remplacement d'une formule par une constante booléenne équivalente (en vue de la formule H), nous avons une transformation équivalente parfaitement conforme aux conditions (i)–(iv).

Transformations non-équivalentes

Les *transformations non-équivalentes* sont plus complexes. Bien que le théorème 2.2.7 suffise pour satisfaire la condition (i), l'extension correspondante du théorème 2.2.12 est impossible : le remplacement par une sous-formule non-équivalente peut toujours falsifier des propriétés locales dirigées dans les positions externes.

Considérez une conjonction $A \wedge B$. D'après le théorème 2.2.7, nous pouvons remplacer A par $\top$ (comme A implique $\top$) et obtenir la formule $\top \wedge B$ qui est la conséquence de la conjonction initiale. Or, par cette transformation, l'occurrence de B perd une propriété locale, notamment A. De l'autre côté, nous pouvons sans problèmes remplacer A par une formule qui implique A, par exemple $A \wedge C$: la formule obtenue impliquera $A \wedge B$, et l'occurrence de B héritera évidemment toutes les propriétés locales de la formule initiale (avec quelques-unes de plus).

Alors nous sommes obligés de restreindre la notion de transformation non-équivalente. Soit H une formule, E une formule ou un terme, $s \in \{+, -\}$ une polarité et π une position dans $\Pi_{\mathbf{F}}(H)$ si E est une formule ou dans $\Pi_{\mathbf{t}}(H)$ autrement. Le *remplacement signé* de $H|_\pi$ par E, notée $H[E]^s_\pi$, est défini comme suit :

$$
\begin{array}{ccc}
(\forall x\, F)[E]^+_{0.\tau} = \forall x\, F[E]^+_\tau & (\exists x\, F)[E]^+_{0.\tau} = \exists x\, F[E]^+_\tau & (F \equiv G)[E]^+_\tau = F \equiv G \\
(F \supset G)[E]^+_{0.\tau} = F[E]^-_\tau \supset G & (F \vee G)[E]^+_{0.\tau} = F[E]^+_\tau \vee G & (F \wedge G)[E]^+_{0.\tau} = F[E]^+_\tau \wedge \top \\
(F \supset G)[E]^+_{1.\tau} = F \supset G[E]^+_\tau & (F \vee G)[E]^+_{1.\tau} = F \vee G[E]^+_\tau & (F \wedge G)[E]^+_{1.\tau} = F \wedge G[E]^+_\tau \\
(\neg F)[E]^+_{0.\tau} = \neg F[E]^-_\tau & F[E]^+_\varepsilon = E & A[E]^+_\tau = A[E]_\tau
\end{array}
$$

$$
\begin{array}{ccc}
(\forall x\, F)[E]^-_{0.\tau} = \forall x\, F[E]^-_\tau & (\exists x\, F)[E]^-_{0.\tau} = \exists x\, F[E]^-_\tau & (F \equiv G)[E]^-_\tau = F \equiv G \\
(F \supset G)[E]^-_{0.\tau} = F[E]^+_\tau \supset \bot & (F \vee G)[E]^-_{0.\tau} = F[E]^-_\tau \vee \bot & (F \wedge G)[E]^-_{0.\tau} = F[E]^-_\tau \wedge G \\
(F \supset G)[E]^-_{1.\tau} = F \supset G[E]^-_\tau & (F \vee G)[E]^-_{1.\tau} = F \vee G[E]^-_\tau & (F \wedge G)[E]^-_{1.\tau} = F \wedge G[E]^-_\tau \\
(\neg F)[E]^-_{0.\tau} = \neg F[E]^+_\tau & F[E]^-_\varepsilon = E & A[E]^-_\tau = A[E]_\tau
\end{array}
$$

Le remplacement signé efface (en remplaçant par des constantes booléennes) précisément les sous-formules qui perdent ces propriétés locales pendant la transformation correspondante.

Lemme 2.2.19. *Pour toute formule F, une formule (un terme) E et une position d'une formule (respectivement d'un terme) π, on a :*

$$\vdash F[E]_\pi \supset F[E]^+_\pi \qquad\qquad \vdash F[E]^-_\pi \supset F[E]_\pi$$

Démonstration. La formule $F[E]^+_\pi$ peut être obtenue de $F[E]_\pi$ par une série de remplacements par $\top$ dans des positions positives de F, ou par $\bot$ dans des positions négatives de F. Pour tout remplacement de cette sorte, le théorème 2.2.7 est applicable. L'autre assertion est démontrée de la même façon. □

Corollaire 2.2.20. *Pour toutes formules F, U, V,*

$$
\begin{array}{rcl}
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (F[U]_\pi \supset F[V]^+_\pi) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (F[U]_\pi \supset F[V]^+_\pi) \\
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (F[V]^-_\pi \supset F[U]_\pi) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (F[V]^-_\pi \supset F[U]_\pi)
\end{array}
$$

Montrons que la condition (ii) est satisfaite par cette transformation «oublieuse».

Théorème 2.2.21. *Soit F, U, V, W des formules, π une position dans $\Pi_{\mathbf{F}}(F)$ et τ une position dans $\Pi(F)$ externe par rapport à π. Si $(F[V]^+_\pi)|_\tau$ n'est pas une constante booléenne, nous avons :*

$$
\begin{array}{rcl}
\pi \in \Pi^+_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^+_\pi}_\tau) \\
\pi \in \Pi^-_{\mathbf{F}}(F) & \Rightarrow & \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^+_\pi}_\tau)
\end{array}
$$

De même, si $(F[V]^-_\pi)|_\tau$ n'est pas une constante booléenne :

$$\begin{aligned}\pi \in \Pi^+_{\mathbf{F}}(F) \quad &\Rightarrow \quad \vdash \langle\!|V \supset U|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^-_\pi}_\tau)\\ \pi \in \Pi^-_{\mathbf{F}}(F) \quad &\Rightarrow \quad \vdash \langle\!|U \supset V|\!\rangle^F_\pi \supset (\langle\!|W|\!\rangle^{F[U]_\pi}_\tau \supset \langle\!|W|\!\rangle^{F[V]^-_\pi}_\tau)\end{aligned}$$

Démonstration. Un remplacement signé dans une position π n'affecte les propriétés locales que dans des sous-formules au-dessous et à droite de π. Alors si τ est un prédécesseur textuel de π ou π est une sous-position de τ, l'assertion du théorème est triviale. En effet, une image locale dirigée dans la position τ ne dépend pas de sous-formules à droite de τ. Donc nous pouvons supposer que π est un prédécesseur textuel de τ, c'est-à-dire qu'il y a des positions ω, μ, η telles que $\pi = \omega.0.\mu$ et $\tau = \omega.1.\eta$. Nous pouvons aussi supposer que $\omega = \epsilon$, car cette partie de l'image locale est la même dans $\langle\!|W|\!\rangle^{F[U]_\pi}_\tau$ et dans $\langle\!|W|\!\rangle^{F[V]^s_\pi}_\tau$. Considérons l'implication :

$$\langle\!|W|\!\rangle^{F[U]_{0.\mu}}_{1.\eta} \supset \langle\!|W|\!\rangle^{F[V]^s_{0.\mu}}_{1.\eta}$$

Par définition, elle est égale à (mettons que $F = G * H$) :

$$\langle\!|W|\!\rangle^{G[U]_\mu * H}_{1.\eta} \supset \langle\!|W|\!\rangle^{G[V]^s_\mu * H'}_{1.\eta} \qquad (\dagger)$$

Maintenant, en dépendance du connecteur $*$ et la polarité s, H' est soit H soit une constante booléenne. Pourtant, si H' est une constante booléenne, alors $(F[V]^s_\pi)|_\tau = H'_\eta$ est une constante booléenne aussi. Donc nous pouvons supposer que $H' = H$. Et nous pouvons supposer aussi que $*$ n'est pas l'équivalence, car sinon π serait une position neutre.

Cas $F = G \wedge H$. Comme H' n'est pas une constante booléenne, la polarité s doit être $-$. Alors l'implication (†) est égale à :

$$(G[U]_\mu \supset \langle\!|W|\!\rangle^H_\eta) \supset (G[V]^-_\mu \supset \langle\!|W|\!\rangle^H_\eta)$$

Si π était positive dans F, alors μ est positive dans G. D'après le corollaire 2.2.20, $\langle\!|V \supset U|\!\rangle^F_\pi$ (qui est égale à $\langle\!|V \supset U|\!\rangle^G_\mu$) implique $(G[V]^-_\mu \supset G[U]_\mu)$ et l'assertion est démontrée. Le cas $\pi \in \Pi^-_{\mathbf{F}}(F)$ est similaire.

Cas $F = G \supset H$. La polarité s doit être $+$ et l'implication (†) est égale à :

$$(G[U]_\mu \supset \langle\!|W|\!\rangle^H_\eta) \supset (G[V]^-_\mu \supset \langle\!|W|\!\rangle^H_\eta)$$

Si π était positive dans F, alors μ est négative dans G. De nouveau, $\langle\!|U \supset V|\!\rangle^F_\pi$ (qui est égale à $\langle\!|U \supset V|\!\rangle^G_\mu$) implique $(G[V]^-_\mu \supset G[U]_\mu)$ et l'assertion est démontrée. Le cas $\pi \in \Pi^-_{\mathbf{F}}(F)$ est similaire.

Cas $F = G \vee H$. La polarité s doit être $+$ et l'implication (†) est égale à :

$$(G[U]_\mu \vee \langle\!|W|\!\rangle^H_\eta) \supset (G[V]^+_\mu \vee \langle\!|W|\!\rangle^H_\eta)$$

Si π était positive dans F, alors μ est positive dans G. De nouveau, $\langle\!|U \supset V|\!\rangle^F_\pi$ (qui est égale à $\langle\!|U \supset V|\!\rangle^G_\mu$) implique $(G[U]_\mu \supset G[V]^+_\mu)$ et l'assertion est démontrée. Le cas $\pi \in \Pi^-_{\mathbf{F}}(F)$ est similaire. □

En somme, le remplacement positif nous permet de générer des conséquences, le remplacement négatif des antécédents, et les deux transformations préservent des propriétés locales dans les positions externes (en éliminant les positions où elles n'y arrivent pas).

Substitution locale. Une variété particulièrement utile de transformations non-équivalentes est la substitution locale. Appelons une occurrence d'un quantificateur *éliminable* si c'est un quantificateur universel dans une position positive ou un quantificateur existentiel dans une position négative. (Nous ne définissons pas une notion duale pour des quantificateurs dans les formules-buts puisque nous n'appliquons pas de substitutions locales dans les buts.) Appelons

une variable *instançiable* dans une position donnée si elle est bornée dans cette position par un quantificateur éliminable.

Considérons une formule F, une position $\pi \in \Pi(F)$ et une variable x instançiable dans la position π. Soit μ la position du quantificateur éliminable sur x. Soit t un terme libre pour la substitution dans x dans la formule $F|_{\mu.0}$.

Le résultat de la *substitution locale (atomique)* de t dans x dans l'occurrence F, π, noté $F[[t/x]]^+_\pi$, est la formule $F[(F|_{\mu.0})[t/x]]^+_\mu$. C'est-à-dire que nous remplaçons la sous-formule quantifiée $F|_\mu$ par l'instance $(F|_{\mu.0})[t/x]$ en utilisant le remplacement positif. Notez que le résultat de la substitution locale est le même pour toute position π sous la position μ donnée.

Lemme 2.2.22. *Soit F, π, x, μ et t introduit comme ci-dessus. Soit η une position dans $\Pi(F|_{\mu.0})$. Soit τ une position dans $\Pi(F)$ telle que τ est externe par rapport à μ et $(F[[t/x]]^+_\pi)|_\tau$ n'est pas une constante booléenne. Alors nous obtenons :*

$$\vdash F \supset F[[t/x]]^+_\pi \tag{a}$$

$$\vdash \langle\!|W|\!\rangle^F_\tau \supset \langle\!|W|\!\rangle^{F[[t/x]]^+_\pi}_\tau \tag{b}$$

$$\vdash \langle\!|W|\!\rangle^F_{\mu.0.\eta} \supset \langle\!|W[t/x]|\!\rangle^{F[[t/x]]^+_\pi}_{\mu.\eta} \tag{c}$$

Démonstration. Notons la formule $F[[t/x]]^+_\pi$ par F', la sous-formule $F|_{\mu.0}$ par G et la même sous-formule après substitution, $G[t/x]$, par G'. Ainsi F' est égal à $F[G']^+_\mu$ par définition.

Démontrons l'assertion (a). Si la position μ est positive, $F|_\mu$ est de la forme $\forall x\, G$. Alors $F|_\mu$ implique G', et d'après le corollaire 2.2.20, F implique F'. Si μ est négative, $F|_\mu$ est de la forme $\exists x\, G$. Alors $F|_\mu$ est impliqué par G' et, encore d'après le corollaire 2.2.20, F implique F'. L'assertion (b) peut être démontrée par le même raisonnement et le théorème 2.2.21.

Quant à l'assertion (c), l'image dans l'antécédent $\langle\!|W|\!\rangle^F_{\mu.0.\eta}$ est égale à $\langle\!|\forall x \langle\!|W|\!\rangle^G_\eta|\!\rangle^F_\mu$. Le conséquent $\langle\!|W[t/x]|\!\rangle^{F'}_{\mu.\eta}$ est égal à $\langle\!|\langle\!|W[t/x]|\!\rangle^{F'|_\mu}_\eta|\!\rangle^{F'}_\mu$. Comme la substitution locale n'affecte que les sous-formules au-dessous et à droite de μ, et comme $F'|_\mu$ est G', la dernière image locale est égale à $\langle\!|\langle\!|W[t/x]|\!\rangle^{G'}_\eta|\!\rangle^F_\mu$. Alors, il est suffisant de démontrer que $\forall x \langle\!|W|\!\rangle^G_\eta$ implique $\langle\!|W[t/x]|\!\rangle^{G'}_\eta$. Mais la dernière formule est égale à $(\langle\!|W|\!\rangle^G_\eta)[t/x]$, ce qui prouve l'assertion. □

Maintenant, soit σ une substitution $[t_1/x_1, t_2/x_2, \ldots, t_n/x_n]$ telle que toutes les variables x_1, x_2, …, x_n sont instançiables dans une position π dans une formule F. Nous pouvons admettre sans perte de généralité que les variables x_1, x_2, …, x_n n'interviennent pas dans les termes-subsituts t_1, t_2, …, t_n, et que les positions correspondantes des quantificateurs éliminables μ_1, μ_2, …, μ_n sont en ordre des plus longues aux plus courtes, c'est-à-dire que toute μ_i est la sous-position de μ_{i+1}. Le résultat de la *substitution locale* $F[\sigma]^+_\pi$ est alors défini par l'équation :

$$F[\sigma]^+_\pi = (\ldots(F[[t_1/x_1]]^+_{\mu_1.0})[[t_2/x_2]]^+_{\mu_2.0}\ldots)[[t_n/x_n]]^+_{\mu_n.0}$$

La définition serait plus claire si nous appliquions les substitutions atomiques $[t_i/x_i]$ dans π au lieu de $\mu_i.0$ (immédiatement sous le quantificateur correspondant). Pourtant, nous ne pouvons pas le faire, car après la première substitution atomique la position π peut ne plus être une position valide. Par ailleurs, les positions $\mu_i.0$ ne sont pas affectées par des substitutions locales dans les positions inférieures.

La substitution locale est une série de substitutions atomiques qui, par le lemme 2.2.22, sont des bonnes transformations non-équivalentes atomiques. Ainsi, la substitution locale est aussi une transformation valide qui satisfait les conditions (i)–(iv) ci-dessus. En particulier, les propriétés locales d'une sous-formule-instance sont des instances des propriétés locales de la sous-formule initiale.

2.3 Notion formelle de correction

2.3.1 Correction d'une formule

Dans ce qui suit, une section ForTheL $\mathbb{A}$ est considérée comme un triplet :

$$(\mathrm{T}\, H\, [\Lambda])$$

où T est le genre de la section, H est l'image-formule de $\mathbb{A}$, et Λ est la séquence de sections dans le corps de $\mathbb{A}$ (omise si vide). Le genre de $\mathbb{A}$ est l'un de suivants : `defn` pour une définition, `sign` pour une extension de signature, `axiom` pour un axiome, `theorem` pour un théorème, `case` pour un cas de preuve, `assume` pour une assomption, `select` pour une sélection, `affirm` pour une affirmation à la fin d'un théorème ou dans une démonstration, `posit` pour une affirmation à la fin d'un axiome, une définition ou une extension de signature.

Une formule F est *logiquement correcte* en vue d'une séquence de sections Γ, dénoté $\Gamma \vdash F$, si F peut être déduite dans le calcul classique de prédicats du premier ordre à partir des images-formules des sections dans Γ. Par la suite, si une section est mentionnée où une formule est attendue, c'est l'image-formule de cette section qui est entendue.

Pour définir la correction ontologique, il nous faut une définition auxiliaire. Soit τ une position de terme ou d'atome dans F. Soit $\mathbb{D}$ une définition ou une extension de signature avec le corps $\mathbb{B}_1, \ldots, \mathbb{B}_n, \mathbb{A}$. Rappelons que les sections $\mathbb{B}_i$ sont des assomptions et $\mathbb{A}$ est une affirmation du genre approprié. Soit E le terme ou l'atome de tête dans $\mathbb{A}$ (voir la section 1.3.6). Alors $\mathbb{D}$ est *applicable* dans l'occurrence $F|_\tau$ en vue de Γ si $F|_\tau = E\sigma$ et $\Gamma \vdash \langle\!|(|\mathbb{B}_1| \wedge \cdots \wedge |\mathbb{B}_n|)\sigma|\!\rangle^F_\tau$.

Une formule F est *ontologiquement correcte* en vue de Γ, ce que l'on note $\Gamma \blacktriangleright F$, si pour toute occurrence $F|_\tau$ de la forme $f(s_1, \ldots, s_n)$, $P(s_1, \ldots, s_n)$, ou $s \,\varepsilon\, N(s_1, \ldots, s_n)$ il y a une définition ou une extension de signature applicable dans Γ.

2.3.2 Calcul de textes corrects

La correction d'un texte est déduite à partir de correction logique et ontologique de formules particulières selon le Calcul de textes corrects **CTC** dont les règles d'inférence sont données dans la Figure 2.1.

Dans ce calcul, nous inférons des séquents de la forme $\Gamma \triangleright_G \Delta$. Seuls les séquents sont admis où $\mathcal{DV}_\Gamma(G) = \varnothing$ (i.e. $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$), et ni Γ ni G ne contiennent d'occurrences de $\mathfrak{T}$.

Dans un tel séquent, Δ est une séquence de sections dont on vérifie la correction et Γ est une séquence de sections qui précèdent Δ logiquement. La formule G est la *thèse courante* : une formule que l'on déduit de Γ à l'aide de raisonnement supplémentaire dans Δ (notez la première règle d'inférence de **CTC**).

Remarquons que la vérification d'une séquence Δ consiste en contre-application de règles de **CTC**, réduisant le séquent $\Gamma \triangleright_G \Delta$ aux $\vdash$- et $\blacktriangleright$-prémisses qui sont vérifiées directement.

Un texte Γ est *correct* si on peut inférer le séquent $\triangleright_\top \Gamma$ dans **CTC**. Nous disons que Γ est *logiquement correct* si on peut inférer $\triangleright_\top \Gamma$ en présumant l'axiome supplémentaire $\Gamma \blacktriangleright F$ pour des Γ et F arbitraires. De la même façon, Γ est *ontologiquement correct* si on peut inférer $\triangleright_\top \Gamma$ en présumant l'axiome supplémentaire $\Gamma \vdash F$ pour des Γ et F arbitraires.

Expliquons la nouvelle notation et d'autres particularités qui apparaissent dans **CTC**.

La prémisse $(\Gamma \blacktriangleright F)^*$ reflète une réserve spéciale qui nous devons faire en vérifiant la correction ontologique d'une définition ou une extension de signature. Notamment, le terme ou l'atome de tête dans les propositions spéciales peuvent ne pas avoir une définition ou une extension de signature applicable dans le texte précèdent, justement parce que ce symbole est introduit par la section considérée. Ainsi, la prémisse $(\Gamma \blacktriangleright F)^*$ dit que F est ontologiquement correct dans toute position prescrite sauf celle du terme ou de l'atome de tête.

L'expression $\Theta_G(F)$ note la formule F où certaines occurrences de G sont remplacées par $\mathfrak{T}$. Il peut exister plus qu'une seule formule $\Theta_G(F)$ pour F et G donnés.

Les expressions $\mathrm{IT}^\prec_t(G)$ et $\mathrm{IH}^\prec_t(G)$ désignent respectivement la *thèse d'induction* et *l'hypothèse d'induction*. Elles sont définies par la règle suivante. Soit G une formule de la forme :

$$\forall \vec{x}_1 \, (H_1 \supset \forall \vec{x}_2 \, (H_2 \supset \ldots \forall \vec{x}_n \, (H_n \supset F) \ldots))$$

Soit t un terme et $\prec$ un symbole de relation binaire. Alors :

$$\mathrm{IH}^\prec_t(G) = \forall \vec{x}'_1 \, (H_1\sigma \supset \forall \vec{x}'_2 \, (H_2\sigma \supset \ldots \forall \vec{x}'_n \, (H_n\sigma \supset (t\sigma \prec t \supset F\sigma)) \ldots))$$

$$\mathrm{IT}^\prec_t(G) = \forall \vec{x}_1 \, (H_1 \supset \forall \vec{x}_2 \, (H_2 \supset \ldots \forall \vec{x}_n \, (H_n \supset (\mathrm{IH}^\prec_t(G) \supset F)) \ldots))$$

où σ est la substitution «renommante» $[\vec{x}'_1/\vec{x}_1, \vec{x}'_2/\vec{x}_2, \ldots, \vec{x}'_n/\vec{x}_n]$ et $\vec{x}'_1, \vec{x}'_2, \ldots, \vec{x}'_n$ sont des variables fraîches.

$$\frac{\Gamma \vdash G}{\Gamma \ \triangleright_G} \qquad \frac{(\Gamma \blacktriangleright F)^* \qquad \Gamma,\ (\texttt{posit}\ F) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{posit}\ F),\ \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \vec{x}\,(F \supset G') \supset G \qquad \Gamma,\ (\texttt{assume}\ F) \triangleright_{G'} \Delta}{\Gamma \ \triangleright_G \ (\texttt{assume}\ \Theta_G(F)),\ \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \mathcal{DV}_\Gamma(F) = \varnothing \qquad \Gamma \triangleright_F \Lambda \qquad \Gamma \vdash (F \wedge G') \supset G \qquad \Gamma,\ (\texttt{affirm}\ F\ [\Lambda]) \triangleright_{G'} \Delta}{\Gamma \ \triangleright_G \ (\texttt{affirm}\ \Theta_G(F)\ [\Lambda]),\ \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \triangleright_{\exists \vec{x}\, F} \Lambda \qquad \Gamma \vdash \exists \vec{x}\,(F \wedge G') \supset G \qquad \Gamma, (\texttt{select}\ F\ [\Lambda]) \triangleright_{G'} \Delta}{\Gamma \ \triangleright_G \ (\texttt{select}\ \Theta_G(F)\ [\Lambda]),\ \Delta}$$

$$\frac{\Gamma \blacktriangleright F \qquad \mathcal{DV}_\Gamma(F) = \varnothing \qquad \Gamma, (\texttt{assume}\ F) \triangleright_G \Lambda \qquad \Gamma,\ (\texttt{case}\ (F \supset G)\ [\Lambda]) \triangleright_{G \vee F} \Delta}{\Gamma \ \triangleright_G \ (\texttt{case}\ (F \supset \mathfrak{T})\ [\Lambda]),\ \Delta}$$

$$\frac{\Gamma \triangleright_{\mathrm{IT}_t^\prec(G)} \Delta}{\Gamma \ \triangleright_G \ \Delta} \qquad \frac{\mathcal{DV}_{\Gamma,\Lambda}(\mathrm{IH}_t^\prec(G)) = \varnothing \qquad \Gamma \triangleright_{\mathrm{IT}_t^\prec(G)} \Lambda,\ (\texttt{assume}\ \mathrm{IH}_t^\prec(G)),\ \Delta}{\Gamma \ \triangleright_G \ \Lambda, \Delta}$$

$$\frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{theorem}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{theorem}\ |\Lambda|\ [\Lambda]),\ \Delta} \qquad \frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{axiom}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{axiom}\ |\Lambda|\ [\Lambda]),\ \Delta}$$

$$\frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{defn}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{defn}\ |\Lambda|\ [\Lambda]),\ \Delta} \qquad \frac{\Gamma \triangleright_\top \Lambda \qquad \Gamma, (\texttt{sign}\ |\Lambda|\ [\Lambda]) \triangleright_\top \Delta}{\Gamma \ \triangleright_\top \ (\texttt{sign}\ |\Lambda|\ [\Lambda]),\ \Delta}$$

FIG. 2.1: Calcul de textes corrects **CTC**

La thèse d'induction $\mathrm{IT}_t^\prec(G)$ est équivalente à la thèse originale G à condition que $\prec$ dénote un ordonnancement bien-fondé. Notez que la bonne fondation de $\prec$ ne peut pas être donnée par une axiomatisation finie dans le langage du premier ordre et le calcul **CTC** l'accepte sans preuve. Autrement dit, la correction d'un texte est vérifiée en présumant le schéma d'axiomes d'induction générale : $\mathcal{I}nd = \forall(\mathrm{IT}_{\mathbf{t}}^\prec(\mathbf{G}) \supset \mathbf{G})$, où $\mathbf{t}$ et $\mathbf{G}$ tiennent la place respectivement pour un terme et pour une formule de la forme appropriée.

Notons que $\mathrm{IT}_t^\prec(G)$ est équivalent à G si $\prec$ est une relation toujours fausse. C'est pourquoi l'extension de la logique du premier ordre par le symbole $\prec$ et le schéma d'axiomes $\mathcal{I}nd$ est conservatrice.

La première règle de traitement d'induction dans **CTC** dit qu'en démontrant la thèse d'induction nous démontrons aussi la thèse initiale. En contre-application, cela veut dire que le vérificateur peut remplacer la thèse courante par la thèse d'induction au cours de la vérification d'une démonstration par induction.

La deuxième règle de traitement d'induction dit de plus que l'hypothèse d'induction peut être omise dans la démonstration écrite (en ForTheL) et elle y sera insérée par le vérificateur, mais pas avant que toute les variables libres de l'hypothèse soient déclarées.

Le traitement de cas de preuve est une autre règle dans **CTC** où un prédécesseur logique implicite est ajouté par le vérificateur (en contre-application). Notez que l'opération Θ n'est pas appliquée à l'hypothèse de cas dans la conclusion de la règle. En conséquent, une hypothèse de cas en ForTheL ne doit pas contenir de mot `thesis`, sinon elle ne sera pas vérifiée.

2.3.3 Réduction de thèse

Dans les règles de **CTC** pour des assomptions, sélections et affirmations, nous rencontrons des $\vdash$-prémisses qui établissent la correspondance entre la thèse courante G et la nouvelle thèse G'. Par «nouvelle» nous entendons que G' est utilisé comme la thèse pour la séquence suivante

de sections Δ. Une telle transformation de thèse reflète notre perception de développement de démonstration quand une proposition complexe est démontrée.

Par exemple, si nous démontrons une conjonction $F \wedge G$ et nous mettons dans le texte de la démonstration l'affirmation de F (que nous sommes capables à démontrer), la thèse $F \wedge G$ sera alors réduite à G. Si nous démontrons une proposition universelle sur des ensembles $\forall x\,(x\,\varepsilon\,\mathrm{Set} \supset F)$, nous pouvons alors commencer par déclarer x comme un ensemble dans une assomption, ce qui réduira la thèse à F.

Quand nous voyons une telle correspondance entre la thèse courante et la phrase considérée, nous appelons cette phrase *motivée*. Des affirmations, sélections, assomptions motivées permettent de réduire la thèse à une nouvelle formule qui est, on peut espérer, plus simple à démontrer.

Parfois, la correspondance est évidente par la forme même des formules : comme dans le cas où la thèse est une implication dont l'assomption est l'antécédent. Parfois, quelque pas de raisonnement sont nécessaires. Par exemple, si des variables S, T sont déclarées comme ensembles et la thèse courante est $S\,\varepsilon\,\mathrm{Subset}(T)$, l'assomption «`let x be an element of S`» est alors motivée et réduit la thèse à $x\,\varepsilon\,\mathrm{Element}(T)$.

Il n'y a rien de spécial dans des affirmations ou sélections non-motivées. Si nous rencontrons une telle phrase, nous ne changeons pas la thèse courante, voila tout. Au contraire, les assomptions dans une démonstration mathématique bien rédigée doivent être toutes motivées, «suggérées» par la thèse courant. Une assomption non-motivée est un rétrécissement injustifié de l'espace de recherche de preuve. Mais il est aussi possible que nos capacités de raisonnement soient trop faibles pour découvrir la justification.

Comment peut-on déduire $\Gamma \triangleright_G (\mathtt{assume}\, F), \Delta$, si F n'est dans aucune relation visible avec G ? Bien que le calcul **CTC** admet des solutions variées, dans notre système (voir la section 3.2.2), le choix de la nouvelle thèse est guidé par la forme de Δ. Si les images-formules des phrases dans Δ contiennent des occurrences de $\mathfrak{T}$ (par exemple, s'il y a des sections-cas), nous supposons alors que la démonstration de G continue sous une assomption non-motivée et nous laissons la thèse inchangée :

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \vec{x}\,(F \supset G) \supset G \qquad \Gamma,\, (\mathtt{assume}\ F) \triangleright_G \Delta}{\Gamma \ \triangleright_G \ (\mathtt{assume}\ \Theta_G(F)),\, \Delta}$$

La prémisse $\Gamma \vdash \forall \vec{x}\,(F \supset G) \supset G$ est non-triviale : elle est équivalente à la disjonction $G \vee (\exists \vec{x}\, F)$. Rappelons que les variables libres de G sont déclarées dans Γ et donc ne sont pas parmi $\vec{x}$.

Si $\mathfrak{T}$ n'intervient pas dans les images-formules dans Δ, nous supposons que le reste de la démonstration est une sorte de raisonnement indépendant qui doit être considéré en lui-même. Alors nous prenons comme nouvelle thèse l'image du reste de la démonstration Δ. L'inférence est la suivante :

$$\frac{\Gamma \blacktriangleright F \qquad \vec{x} = \mathcal{DV}_\Gamma(F) \qquad \Gamma \vdash \forall \vec{x}\,(F \supset |\Delta|_\Gamma) \supset G \qquad \Gamma, (\mathtt{assume}\ F) \triangleright_{|\Delta|_\Gamma} \Delta}{\Gamma \ \triangleright_G \ (\mathtt{assume}\ \Theta_G(F)),\, \Delta}$$

L'image $|\Delta|_\Gamma$ est calculée selon la définition dans la section 1.5.5. Notez que les nouvelles variables dans Δ, celles inconnues de Γ ou F, sont toutes bornées dans $|\Delta|_\Gamma$.

Toute assomption dans Δ est un antécédent dans la nouvelle thèse $|\Delta|_\Gamma$ et sera donc considérée comme motivée. Toute affirmation ou sélection dans Δ va réduire la thèse aussi. Ainsi à la fin de Δ la thèse finale sera tout simplement $\top$. La prémisse $\Gamma \vdash \forall \vec{x}\,(F \supset |\Delta|_\Gamma) \supset G$ conclut la démonstration en déduisant G de l'image-formule de la séquence $(\mathtt{assume}\ F),\, \Delta$.

Les règles de **CTC** n'exigent pas que la thèse courante soit ontologiquement correcte. Ce n'est qu'au début d'une démonstration, quand la thèse initiale est la proposition démontrée, que sa correction ontologique est assurée. Comme ce sont des thèses initiales qui nous intéressent (voir le théorème 2.3.2), nous ne demandons plus du calcul. Néanmoins, il est bien désirable de maintenir la thèse courante ontologiquement correcte pendant toutes les réductions effectuées par le vérificateur. Ainsi, nous n'arriverons jamais à la fin de démonstration avec un but dont le sens même est incertain.

Si après une phrase non-motivée nous ne changeons pas la thèse courante, elle reste ontologiquement correcte. En effet, $\Gamma \blacktriangleright G$ implique $\Gamma, \mathbb{A} \blacktriangleright G$. L'image-formule de Δ est aussi une thèse admissible, comme le lemme suivant le montre.

Lemme 2.3.1. *Soit* $\Delta = \mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_n$ *une séquence de phrases dont les images-formules ne contiennent pas d'occurrences de* $\mathfrak{T}$. *Soit* Γ *une séquence de sections telle que* $\Gamma \blacktriangleright \mathbb{A}_1$ *et* $\Gamma, \mathbb{A}_1 \blacktriangleright \mathbb{A}_2$ *et ainsi de suite, jusqu'à* $\Gamma, \mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{n-1} \blacktriangleright \mathbb{A}_n$. *Alors* $\Gamma \blacktriangleright |\Delta|_\Gamma$.

Démonstration. Notons la formule $|\Delta|_\Gamma$ par D. Toute image-locale $\langle\!|W|\!\rangle_\pi^D$ que l'on doit déduire de Γ pour démontrer $\Gamma \blacktriangleright D$ est de la forme $\forall \vec{x}_1(|\mathbb{A}_1| \supset \forall \vec{x}_2(|\mathbb{A}_2| \supset \ldots \forall \vec{x}_{i-1}(|\mathbb{A}_{i-1}| \supset \forall \vec{x}_i \langle\!|W|\!\rangle_\tau^{|\mathbb{A}_i|}) \ldots))$, où $\vec{x}_1 = \mathcal{DV}_\Gamma(\mathbb{A}_1)$ et $\vec{x}_2 = \mathcal{DV}_{\Gamma,\mathbb{A}_1}(\mathbb{A}_2)$ et ainsi de suite. Comme les ensembles $\vec{x}_1$, ..., $\vec{x}_i$ sont disjoints l'un de l'autre et aussi de $\mathcal{FV}(\Gamma)$, notre problème est équivalent à $\Gamma, |\mathbb{A}_1|, |\mathbb{A}_2|, \ldots, |\mathbb{A}_{i-1}| \vdash \langle\!|W|\!\rangle_\tau^{|\mathbb{A}_i|}$. Cela nous est donné par $\Gamma, \mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{i-1} \blacktriangleright \mathbb{A}_i$. □

Nous allons démontrer dans la section 3.2.2 que les autres règles de réduction implantées dans notre système sont toutes basées sur des transformations qui satisfont les quatre conditions de la section 2.2.3. Ainsi la correction ontologique de la thèse courante est assurée.

Finalement, le théorème suivant peut être vu comme l'affirmation de cohérence de **CTC** :

Théorème 2.3.2. *Soit* Γ *et* Δ *des séquences de sections et* G *une formule. Si le séquent* $\Gamma \triangleright_G \Delta$ *peut être déduit dans* **CTC** *alors* $\mathcal{I}nd, \Gamma \vdash G$.

Démonstration. Nous allons démontrer l'assertion par le nombre de pas dans l'inférence de $\Gamma \triangleright_G \Delta$. Considérons le dernier pas. S'il est fait d'après une règle avec $\triangleright_\top$ dans la conclusion, alors G est $\top$ et l'assertion est triviale. Sinon nous avons sept cas à considérer. Nous allons dénoter les cas par la forme de la conclusion de la règle d'inférence correspondante.

Cas $\Gamma \triangleright_G$. La prémisse de cette règle est $\Gamma \vdash G$, d'où l'assertion.

Cas $\Gamma \triangleright_G (\texttt{assume}\ \Theta_G(F)), \Delta$. Par les prémisses de la règle, nous avons :

$$\Gamma \vdash \forall \vec{x}\,(F \supset G') \supset G \qquad\qquad \Gamma, (\texttt{assume}\ F) \triangleright_{G'} \Delta$$

où $\vec{x} = \mathcal{DV}_\Gamma(F) = \mathcal{FV}(F) \backslash \mathcal{FV}(\Gamma)$. Par l'hypothèse d'induction, cela implique $\mathcal{I}nd, \Gamma, F \vdash G'$. Aussi $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$ et $\mathcal{FV}(G') \subseteq \mathcal{FV}(\Gamma) \cup \mathcal{FV}(F)$. Alors $\mathcal{FV}(F \supset G') \backslash \mathcal{FV}(\Gamma) = \vec{x}$. Ainsi $\mathcal{I}nd, \Gamma \vdash \forall \vec{x}\,(F \supset G')$ et nous obtenons l'assertion.

Cas $\Gamma \triangleright_G (\texttt{affirm}\ \Theta_G(F)\ [\Lambda]), \Delta$. Cela est un cas particulier de celui qui suit.

Cas $\Gamma \triangleright_G (\texttt{select}\ \Theta_G(F)\ [\Lambda]), \Delta$. Par les prémisses, nous avons :

$$\Gamma \triangleright_{\exists \vec{x}\, F} \Lambda \qquad \Gamma \vdash \exists \vec{x}\,(F \wedge G') \supset G \qquad \Gamma, (\texttt{select}\ F\ [\Lambda]) \triangleright_{G'} \Delta$$

où $\vec{x} = \mathcal{FV}(F) \backslash \mathcal{FV}(\Gamma)$. Par l'hypothèse d'induction, nous obtenons $\mathcal{I}nd, \Gamma, F \vdash G'$ et $\mathcal{I}nd, \Gamma \vdash \exists \vec{x}\, F$. Aussi $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$ et $\mathcal{FV}(G') \subseteq \mathcal{FV}(\Gamma) \cup \mathcal{FV}(F)$. Par conséquent nous obtenons $\mathcal{FV}(F \wedge G') \backslash \mathcal{FV}(\Gamma) = \vec{x}$ et $\mathcal{I}nd, \Gamma \vdash \forall \vec{x}\,(F \supset G')$. Cela implique $\mathcal{I}nd, \Gamma \vdash \exists \vec{x}\,(F \wedge G')$ et nous avons l'assertion.

Cas $\Gamma \triangleright_G (\texttt{case}\ (F \supset \mathfrak{T})\ [\Lambda]), \Delta$. Par les prémisses, nous avons :

$$\Gamma, (\texttt{assume}\ F) \triangleright_G \Lambda \qquad\qquad \Gamma, (\texttt{case}\ (F \supset G)\ [\Lambda]) \triangleright_{G \vee F} \Delta$$

Aussi, $\mathcal{DV}_\Gamma(F) = \varnothing$ d'où $\mathcal{FV}(F), \mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$. Par l'hypothèse d'induction, nous obtenons $\mathcal{I}nd, \Gamma, F \vdash G$ et $\mathcal{I}nd, \Gamma, (F \supset G) \vdash (G \vee F)$. Le premier séquent implique $\mathcal{I}nd, \Gamma \vdash (F \supset G)$. Le deuxième séquent implique $\mathcal{I}nd, \Gamma, (F \supset G) \vdash G$ et nous avons l'assertion.

Cas $\Gamma \triangleright_G \Delta$ *et* $\Gamma \triangleright_G \Delta, \Lambda$ (règles de traitement d'induction). Par la prémisse de la règle et par l'hypothèse d'induction, nous obtenons $\mathcal{I}nd, \Gamma \vdash \mathrm{IT}_t^{\prec}(G)$. Par définition de $\mathcal{I}nd$, nous obtenons l'assertion. □

2.4 Exemple de vérification

À la figure 2.2, nous donnons un texte bien-formé en ForTheL. Dans cette section, nous allons le vérifier conformément aux règles du calcul **CTC**.

Remarquez les cortèges de nombres dans les commentaires. Chaque cortège note une «position» de la section correspondante (nous mettons le mot entre guillemets, car nous n'avons

```
[number/numbers]

Signature Nat.                                          # 0
  A natural number is a notion.                         # 0.0
Signature Zer.                                          # 1
  0 is a natural number.                                # 1.0
Signature Suc.                                          # 2
  Let i be a natural number.                            # 2.0
  succ i is a natural number.                           # 2.1
Signature Add.                                          # 3
  Let i,j be natural numbers.                           # 3.0
  i + j is a natural number.                            # 3.1
Signature Ord.                                          # 4
  Let i,j be natural numbers.                           # 4.0
  i -<- j is an atom.                                   # 4.1

Axiom ZerSuc.                                           # 5
  For any natural number i if i != 0 then               # 5.0
  there exists a natural number j such that succ j = i.
Axiom AddZer.                                           # 6
  For any natural number i (i + 0 = i).                 # 6.0
Axiom AddSuc.                                           # 7
  For all natural numbers i,j (i + succ j = succ (i+j)). # 7.0
Axiom OrdSuc.                                           # 8
  For any natural number i (i -<- succ i).              # 8.0

Lemma ZerAdd.                                           # 9
  For any natural number i (0 + i = i).                 # 9.0
Proof by induction.
  Let i be a natural number.                            # 9.0.0

  Case i = 0.                                           # 9.0.1
  Obvious.

  Case i != 0.                                          # 9.0.2
    Take a natural number j such that succ j = i.       # 9.0.2.0
    We have j -<- i.                                    # 9.0.2.1
    Hence 0 + j = j.                                    # 9.0.2.2
    Then we have the thesis.                            # 9.0.2.3
  end.
qed.
```

FIG. 2.2: Texte sur addition zéro de gauche

```
sign Nat
   posit  ∀x (x ε NatNum ⊃ ⊤)
sign Zer
   posit  ∀x (x ≈ 0 ⊃ x ε NatNum)
sign Suc
   assume  i ε NatNum
   posit  ∀x (x ≈ succ i ⊃ x ε NatNum)
sign Add
   assume  i ε NatNum ∧ j ε NatNum
   posit  ∀x (x ≈ i + j ⊃ x ε NatNum)
sign Ord
   assume  i ε NatNum ∧ j ε NatNum
   posit  i ≺ j ⊃ ⊤

axiom ZerSuc
   posit  ∀i (i ε NatNum ⊃ i ≠ 0 ⊃ ∃j (j ε NatNum ∧ succ j ≈ i))
axiom AddZer
   posit  ∀i (i ε NatNum ⊃ i + 0 ≈ i)
axiom AddSuc
   posit  ∀i (i ε NatNum ⊃ ∀j (j ε NatNum ⊃ i + (succ j) ≈ succ (i + j)))

axiom OrdSuc
   posit  ∀i (i ε NatNum ⊃ i ≺ succ i)

prop ZerAdd
   affirm  ∀i (i ε NatNum ⊃ 0 + i ≈ i)
      assume  i ε NatNum
      case  i = 0 ⊃ 𝔗
      case  i ≠ 0 ⊃ 𝔗
         select  j ε NatNum ∧ (succ j) ≈ i
         affirm  j ≺ i
         affirm  0 + j ≈ j
         affirm  𝔗
```

FIG. 2.3: Texte sur addition zéro de gauche (traduit)

pas étendu la notion de position sur les textes). Par exemple, 9.0 est la position de l'affirmation principale dans le lemme `ZerAdd`, 9.0.0 est la position de l'assomption qui commence la démonstration et 9.0.1, 9.0.2 indiquent les sections-cas. La relation de précédence logique sur ces positions (comme défini au début de la section 2.2.1) correspond bien à la précédence de sections ForTheL (comme défini dans la section 1.5.4).

Reconsidérons ce texte avec des images-formules à la place des phrases en ForTheL (figure 2.3). Nous allons effectuer les pas d'inférence nécessaires pour démontrer la correction du texte. Pour entrer dans la page, nous écrivons des positions entre parenthèses à la place des sections-triplets. Nous procédons de bas en haut, à partir de la conclusion désirée vers les axiomes. Commençons par le haut niveau.

$$\dfrac{\dfrac{\dfrac{}{\triangleright_\top (0), \ldots, (9)}}{\triangleright_\top (0.0)} \quad \dfrac{\dfrac{}{(0) \triangleright_\top (1), \ldots, (9)}}{(0) \triangleright_\top (1.0)} \quad \dfrac{\dfrac{}{(0,1) \triangleright_\top (2), \ldots, (9)}}{(0), (1) \triangleright_\top (2.0), (2.1)} \quad \dfrac{}{(0), (1), (2) \triangleright_\top (3), \ldots, (9)}}{}$$

$$\dfrac{(0),(1),(2) \rhd_\top (3),\ldots,(9)}{(0),(1),(2) \rhd_\top (3.0),(3.1) \qquad \dfrac{(0),\ldots,(3) \rhd_\top (4),\ldots,(9)}{(0),\ldots,(3) \rhd_\top (4.0),(4.1) \qquad (0),\ldots,(4) \rhd_\top (5),\ldots,(9)}}$$

$$\dfrac{(0),\ldots,(4) \rhd_\top (5),\ldots,(9)}{(0),\ldots,(4) \rhd_\top (5.0) \qquad \dfrac{(0),\ldots,(5) \rhd_\top (6),\ldots,(9)}{(0),\ldots,(5) \rhd_\top (6.0) \qquad (0),\ldots,(6) \rhd_\top (7),(8),(9)}}$$

$$\dfrac{(0),\ldots,(6) \rhd_\top (7),(8),(9)}{(0),\ldots,(6) \rhd_\top (7.0) \qquad \dfrac{(0),\ldots,(7) \rhd_\top (8),(9)}{(0),\ldots,(7) \rhd_\top (8.0) \qquad \dfrac{(0),\ldots,(8) \rhd_\top (9)}{(0),\ldots,(8) \rhd_\top (9.0) \qquad \dfrac{(0),\ldots,(9) \rhd_\top}{(0),\ldots,(9) \vdash \top}}}}$$

Maintenant, inspectons les corps des sections haut-niveau. Notez que la thèse $\top$ n'a jamais besoin d'être réduite dans les règles pour des assomptions, sélections et affirmations. C'est-à-dire que la nouvelle thèse (notée G' dans les règles à la figure 2.1) restera $\top$.

$$\dfrac{\rhd_\top (0.0)}{\blacktriangleright \forall x\,(x \varepsilon \mathrm{NatNum} \supset \top) \qquad \dfrac{(0.0) \rhd_\top}{(0.0) \vdash \top}} \qquad \dfrac{(0) \rhd_\top (1.0)}{(0) \blacktriangleright 0 \varepsilon \mathrm{NatNum} \qquad \dfrac{(0),(1.0) \rhd_\top}{(0),(1.0) \vdash \top}}$$

$$\dfrac{(0),(1) \rhd_\top (2.0),(2.1)}{(0),(1) \blacktriangleright i \varepsilon \mathrm{NatNum} \qquad (0),(1), \vdash \forall i\,(i \varepsilon \mathrm{NatNum} \supset \top) \supset \top \qquad (0),(1),(2.0) \rhd_\top (2.1)}$$

$$\dfrac{(0),(1),(2.0) \rhd_\top (2.1)}{(0),(1),(2.0) \blacktriangleright (\mathrm{succ}\ i) \varepsilon \mathrm{NatNum} \qquad \dfrac{(0),(1),(2.0),(2.1) \rhd_\top}{(0),(1),(2.0),(2.1) \vdash \top}}$$

$$\dfrac{(0),(1),(2) \rhd_\top (3.0),(3.1)}{(0),(1),(2) \blacktriangleright i,j \varepsilon \mathrm{NatNum} \quad (0),(1),(2) \vdash \forall i\,(|(3.0)| \supset \top) \supset \top \quad (0),(1),(2),(3.0) \rhd_\top (3.1)}$$

$$\dfrac{(0),(1),(2),(3.0) \rhd_\top (3.1)}{(0),(1),(2),(3.0) \blacktriangleright (i+j) \varepsilon \mathrm{NatNum} \qquad \dfrac{(0),(1),(2),(3.0),(3.1) \rhd_\top}{(0),(1),(2),(3.0),(3.1) \vdash \top}}$$

$$\dfrac{(0),\ldots,(3) \rhd_\top (4.0),(4.1)}{(0),\ldots,(3) \blacktriangleright i,j \varepsilon \mathrm{NatNum} \quad (0),\ldots,(3) \vdash \forall i\,(|(4.0)| \supset \top) \supset \top \quad (0),\ldots,(3),(4.0) \rhd_\top (4.1)}$$

$$\dfrac{(0),\ldots,(3),(4.0) \rhd_\top (4.1)}{(0),\ldots,(3),(4.0) \blacktriangleright i \prec j \supset \top \qquad \dfrac{(0),\ldots,(3),(4.0),(4.1) \rhd_\top}{(0),\ldots,(3),(4.0),(4.1) \vdash \top}}$$

$$\dfrac{(0),\ldots,(i-1) \rhd_\top ((i).0)}{(0),\ldots,(i-1) \blacktriangleright |((i).0)| \qquad \dfrac{(0),\ldots,(i-1),((i).0) \rhd_\top}{(0),\ldots,(i-1),((i).0) \vdash \top}} \qquad (\text{for all } i \in \{5,6,7,8\})$$

$$\dfrac{\Gamma \rhd_\top (9.0)}{\Gamma \blacktriangleright |(9.0)| \qquad \dfrac{\Gamma \rhd_{|(9.0)|} (9.0.0),(9.0.1),(9.0.2)}{\Gamma \rhd_G (9.0.0),\mathbb{A},(9.0.1),(9.0.2)} \qquad \Gamma \vdash (|(9.0)| \wedge \top) \supset \top \qquad \dfrac{\Gamma,(9.0) \rhd_\top}{\Gamma,(9.0) \vdash \top}}$$

où :

$$\Gamma = (0), \ldots, (8)$$
$$\mathbb{A} = (\texttt{assume}\ H)$$
$$|(9.0)| = \forall i\,(i\,\varepsilon\,\mathrm{NatNum} \supset 0 + i \approx i)$$
$$G = \mathrm{IT}_i^{\prec}(|(9.0)|) = \forall i\,(i\,\varepsilon\,\mathrm{NatNum} \supset (H \supset 0 + i \approx i))$$
$$H = \mathrm{IH}_i^{\prec}(|(9.0)|) = \forall i'\,(i'\,\varepsilon\,\mathrm{NatNum} \supset (i' \prec i \supset 0 + i' \approx i'))$$

Remarquez le dernier fragment de l'inférence où nous appliquons la règle de traitement d'induction. Nous remplaçons la thèse initiale, la proposition de l'affirmation (9.0), par une formule plus faible : la thèse d'induction G. Aussi, nous insérons l'hypothèse d'induction H dans la démonstration. Notons que la variable i qui est libre dans H est déclarée dans (9.0.0) et donc est connue là où l'hypothèse est placée.

Dans l'inférence ci-dessous, notez comment deux assomptions réduisent la thèse courante.

$$\frac{\Gamma \rhd_G (9.0.0), \mathbb{A}, (9.0.1), (9.0.2)}{\Gamma \blacktriangleright i\,\varepsilon\,\mathrm{NatNum} \qquad \Gamma \vdash \forall i\,(i\,\varepsilon\,\mathrm{NatNum} \supset G') \supset G \qquad \Gamma, (9.0.0) \rhd_{G'} \mathbb{A}, (9.0.1), (9.0.2)}$$

$$\frac{\Gamma, (9.0.0) \rhd_{G'} \mathbb{A}, (9.0.1), (9.0.2)}{\Gamma, (9.0.0) \blacktriangleright H \qquad \Gamma, (9.0.0) \vdash (H \supset G'') \supset G' \qquad \Gamma, (9.0.0), \mathbb{A} \rhd_{G''} (9.0.1), (9.0.2)}$$

où :

$$G' = (H \supset 0 + i \approx i) \qquad\qquad G'' = (0 + i \approx i)$$

Le premier cas de preuve est trivial. Notez que $\mathfrak{T}$ dans l'image-formule de (9.0.1) est remplacé par la thèse actuelle dans $(9.0.1)'$:

$$\frac{\Gamma, (9.0.0), \mathbb{A} \rhd_{G''} (9.0.1), (9.0.2)}{\Gamma, (9.0.0), \mathbb{A} \blacktriangleright i \approx 0 \qquad \dfrac{\Gamma, (9.0.0), \mathbb{A}, \mathbb{C}_1 \rhd_{G''}}{\dfrac{\Gamma, (9.0.0), \mathbb{A}, \mathbb{C}_1 \vdash G''}{\Gamma, i\,\varepsilon\,\mathrm{NatNum}, i \approx 0 \vdash\ 0 + i \approx i}} \qquad \Gamma, (9.0.0), \mathbb{A}, (9.0.1)' \rhd_{G'''} (9.0.2)}$$

où :

$$\mathbb{C}_1 = (\texttt{assume}\ i \approx 0) \qquad (9.0.1)' = (\texttt{case}\ (i \approx 0 \supset G'')) \qquad G''' = G'' \vee i \approx 0$$

L'autre cas de preuve est plus longue mais aussi simple :

$$\frac{\Delta_0 \rhd_{G'''} (\tau)}{\Delta_0 \blacktriangleright i \not\approx 0 \qquad \Delta_0, \mathbb{C}_2 \rhd_{G'''} (\tau.0), (\tau.1), (\tau.2), (\tau.3) \qquad \dfrac{\Delta_0, (\tau)' \rhd_{G''' \vee i \not\approx 0}}{\dfrac{\Delta_0, (\tau)' \vdash G''' \vee i \not\approx 0}{\vdash G'' \vee i \approx 0 \vee i \not\approx 0}}}$$

$$\frac{\Delta_1 \rhd_{G'''} (\tau.0), (\tau.1), (\tau.2), (\tau.3)}{\Delta_1 \blacktriangleright F_1 \qquad \dfrac{\Delta_1 \rhd_{\exists j F_1}}{\Delta_1 \vdash \exists j F_1} \qquad \Delta_1 \vdash \exists j\,(F_1 \wedge G''') \supset G''' \qquad \Delta_1, (\tau.0) \rhd_{G'''} (\tau.1), (\tau.2), (\tau.3)}$$

$$\frac{\Delta_2 \rhd_{G'''} (\tau.1), (\tau.2), (\tau.3)}{\Delta_2 \blacktriangleright j \prec i \qquad \dfrac{\Delta_2 \rhd_{j \prec i}}{\Delta_2 \vdash j \prec i} \qquad \Delta_2 \vdash (j \prec i \wedge G''') \supset G''' \qquad \Delta_2, (\tau.1) \rhd_{G'''} (\tau.2), (\tau.3)}$$

$$\frac{\Delta_3 \rhd_{G'''} (\tau.2), (\tau.3)}{\Delta_3 \blacktriangleright 0 + j \approx j \qquad \dfrac{\Delta_3 \rhd_{0+j \approx j}}{\Delta_3 \vdash 0 + j \approx j} \qquad \Delta_3 \vdash (0 + j \approx j \wedge G''') \supset G''' \qquad \Delta_3, (\tau.2) \rhd_{G'''} (\tau.3)}$$

$$\dfrac{\dfrac{\Delta_4 \blacktriangleright G'''}{\Delta_4 \vdash 0 + i \approx i}\ \dfrac{\dfrac{\Delta_4 \rhd_{G'''} (\tau.3)}{\Delta_4 \vdash G'''}}{\Delta_4 \vdash 0 + i \approx i}\ \dfrac{}{\Delta_4 \vdash (G''' \wedge \top) \supset G'''}\ \dfrac{\Delta_4, (\texttt{affirm}\ G''') \rhd_\top}{\Delta_4, (\texttt{affirm}\ G''') \vdash \top}}{\Delta_4 \rhd_{G'''} (\tau.3)}$$

où :

$$\begin{aligned}
\tau &= 9.0.2 & \Delta_0 &= \Gamma, (9.0.0), \mathbb{A}, (9.0.1)' \\
\mathbb{C}_2 &= (\texttt{assume}\ (i \not\approx 0)) & \Delta_1 &= \Delta_0, \mathbb{C}_2 \\
\Lambda &= (\tau.0), (\tau.1), (\tau.2), (\tau.3) & \Delta_2 &= \Delta_1, (\tau.0) \\
(\tau)' &= (\texttt{case}\ (i \not\approx 0 \supset G''')\ [\Lambda]) & \Delta_3 &= \Delta_2, (\tau.1) \\
F_1 &= j \,\varepsilon\, \mathrm{NatNum} \wedge \mathrm{succ}\ j \approx i & \Delta_4 &= \Delta_3, (\tau.2)
\end{aligned}$$

Il faut faire ici plusieurs remarques. Premièrement, considérons la branche de droite dans le premier fragment d'inférence où le but $G'' \vee i \approx 0 \vee i \not\approx 0$ est démontré. D'après les règles du calcul **CTC**, toute section-cas affaiblit la thèse courante en y ajoutant son hypothèse sous la disjonction. À la fin de l'analyse de cas nous devons démontrer la formule $G \vee H_1 \vee \cdots \vee H_n$, où G est la thèse initiale et $H_1, \ldots, H_n$ sont des cas explorés. Ainsi, l'analyse n'est pas obligée d'être exhaustive. Il est bon quand-même d'écrire des cas de telle façon que la disjonction seule $H_1 \vee \cdots \vee H_n$ soit valide à la fin de démonstration.

Deuxièmement, une phrase-sélection est valide si on peut démontrer l'existence des objets cités, c.-à-d. montrer que les classes désignées par les notions listées sont habitées. Bien que dans notre exemple la sélection (9.0.2.0) ne change pas la thèse courante, cela peut se produire si la thèse courante est la proposition d'existence.

Troisièmement, remarquez que l'affirmation (9.0.2.2) est une conséquence directe de l'hypothèse d'induction H et l'affirmation précédente (9.0.2.1). Si l'assomption $\mathbb{A}$ (dont H est l'image-formule) n'était pas insérée dans la démonstration, l'affirmation (9.0.2.2) ne serait pas démontrée. Néanmoins, il est possible de rédiger une démonstration par induction où aucune phrase ne dépend de l'hypothèse d'induction. Par exemple, on pourrait omettre la démonstration entière de l'affirmation (9.0). Le système devrait alors démontrer la thèse d'induction G, ce qui n'est pas difficile et n'exige aucun autre raisonnement par induction.

Quatrièmement, considérons le dernier fragment d'inférence. L'image-formule de l'affirmation (9.0.2.3) est la formule atomique $\mathfrak{T}$, qui désigne la thèse courante G'''. Une fois l'affirmation (9.0.2.3) démontrée, nous n'avons plus d'obligations de preuve et donc la nouvelle thèse est $\top$.

Maintenant, en présumant que tous les $\vdash$-séquents dans notre dérivation sont valides, nous avons démontré la correction logique de notre texte. Pour compléter la vérification, nous devons aussi démontrer que les phrases dans le texte sont ontologiquement correctes ($\blacktriangleright$-séquents). Dans ce exemple, c'est une tâche bien simple. Considérons le séquent $\Delta_1 \blacktriangleright F_1$ qui est équivalent à :

$$(0), \ldots, (8), (9.0.0), \mathbb{A}, (9.0.1)', \mathbb{C}_2 \blacktriangleright j \,\varepsilon\, \mathrm{NatNum} \wedge \mathrm{succ}\ j \approx i$$

Il y a deux occurrences de symboles non-logique dans F_1 : le symbole de notion NatNum (sans arguments) et le symbole de fonction succ (un argument). Pour chacun, il y a une seule extension de signature à inspecter : (0) pour NatNum et (2) pour succ. La première occurrence est évidemment correcte comme la section (0) n'a pas de conditions à démontrer dans la position de $j \,\varepsilon\, \mathrm{NatNum}$. Pour valider l'occurrence succ j, if faut démontrer que :

$$(0), \ldots, (8), (9.0.0), \mathbb{A}, (9.0.1)', \mathbb{C}_2 \vdash \langle\!|j \,\varepsilon\, \mathrm{NatNum}|\!\rangle_{1.0}^{F_1}$$

D'après la définition de l'image locale dirigée, $\langle\!|j \,\varepsilon\, \mathrm{NatNum}|\!\rangle_{1.0}^{F_1} = j \,\varepsilon\, \mathrm{NatNum} \supset j \,\varepsilon\, \mathrm{NatNum}$. Ainsi, la formule F_1 est ontologiquement correcte.

Chapitre 3

Assistant de preuve SAD

3.1 Introduction

Dans ce chapitre nous décrivons l'implantation actuelle des concepts développés dans le texte précédent : le système SAD (acronyme de «System for Automated Deduction»). Les composantes principales du système sont présentées sur la figure 3.1.

L'analyseur syntaxique («parser») reçoit un texte rédigé en ForTheL (soit dans le langage du premier ordre directement), vérifie si le texte est bien formé, et le convertit dans les structures internes de SAD conformément aux règles établies dans les sections 1.4 et 1.5.5. La représentation interne d'un texte ForTheL est une liste des triplets comme introduit dans la section 2.3.2, avec l'information supplémentaire : marques, références, termes d'induction et caetera.

Le *vérificateur* (considéré en détail dans la section 3.2) traverse l'arbre du texte, phrase après phrase, et vérifie si elles sont correctes ontologiquement et logiquement.

Pendant le contrôle ontologique, certaines propriétés locales des termes considérés, appelées *évidences*, sont accumulées et attribuées aux occurrences de ces termes. Les évidences contribuent à la vérification de conditions de définitions et extensions de signature. Il sont aussi employées pour la simplification dans le *raisonneur*.

Ensuite, le vérificateur traite la section en question selon les règles du calcul **CTC** (en contre-application). Il lance des nouvelles sous-vérifications pour les démonstrations données dans le texte et appelle le *raisonneur* pour prouver les assertions sans démonstration et les thèses courantes au bout des démonstrations. La procédure *motivatrice* est appelée pour construire la nouvelle thèse courante en passage à la section suivante.

Le *raisonneur* (considéré en détail dans la section 3.3) s'occupe de tâches de preuve particulières. Ce module peut être vu comme un démonstrateur heuristique qui opère par un assortiment des techniques de transformation des tâches de preuve. Cet assortiment ne constitue pas de calcul logique complet. L'objectif du raisonneur, «le niveau haut», n'est pas la recherche de preuve — mais de simplifier la tâche pour le *démonstrateur* automatique, «le niveau bas».

Les capacités du raisonneur actuellement implantées sont les suivantes : décomposer des buts conjonctifs, les simplifier à l'aide de propriétés locales accumulées, filtrer des prémisses selon des suggestions dans le texte, appliquer des définitions (ce qui permet de les «cacher» au démonstrateur).

Le *démonstrateur* est une procédure combinatoire de recherche de preuve en logique classique du premier ordre, dont l'objectif est de compléter les démonstrations commencées par le raisonneur. Si le démonstrateur n'arrive pas à construire la preuve, le raisonneur peut continuer à transformer la tâche, essayer un autre chemin de transformation ou rejeter le texte.

Le démonstrateur est un programme indépendant et SAD sait collaborer avec les démonstrateurs automatiques bien connus : Otter [48], SPASS [76], Vampire [61], E [64]. Notons que cela nous donne une échelle (de plus) pour comparer des démonstrateurs : en les testant sur des problèmes relativement simples dans les contextes complexes et surabondants (plutôt que sur des problèmes difficiles avec des prémisses ajustées, ce qui est souvent le cas pour des problèmes dans la librairie TPTP [68]). Il faut mentionner dans ce rapport le projet MPTP [70] qui emploie des démonstrateurs automatiques pour des problèmes de la librairie de Mizar.

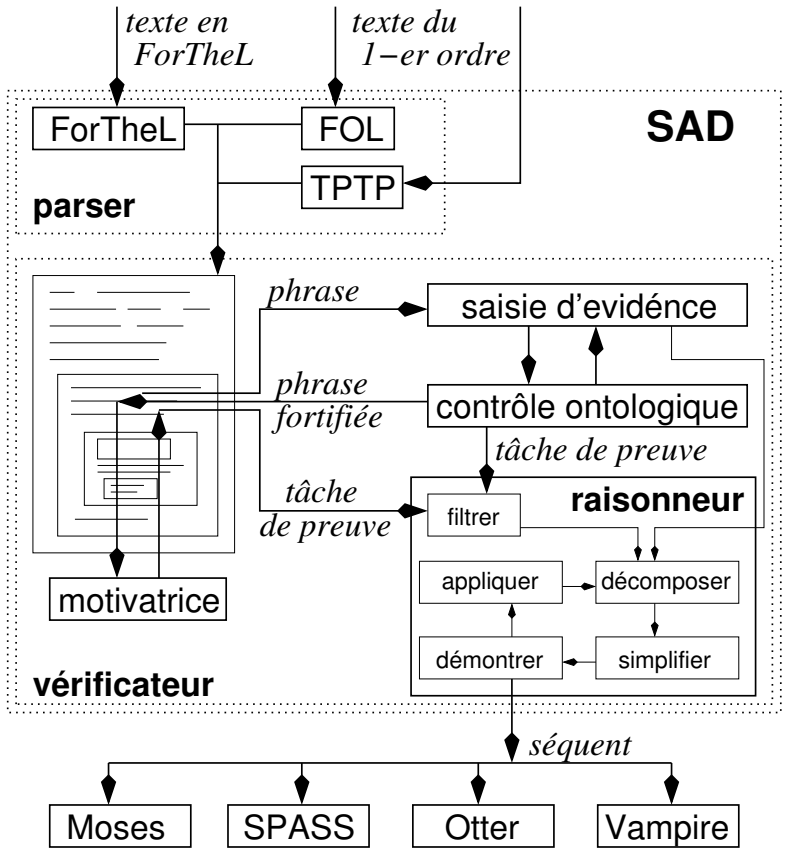


FIG. 3.1: Architecture de SAD

Le démonstrateur natif de SAD, appelé Moses, est basé sur un calcul original orienté but qui est décrit dans le chapitre 4.3. Moses effectue la recherche en profondeur d'abord avec retour-arrière (*depth-first traversal with backtracking*) et augmentation progressive de la profondeur limite. Les contraintes d'unification et génération de lemmes littéraux (*folding-up*) [40] sont employés pour améliorer l'efficacité de recherche. La notion originale de substitutions admissibles (section 4.3.1) permet de se passer de la skolémisation préliminaire et préserver la signature initiale de la tâche.

Remarquons que dans nos expériences avec les démonstrateurs mentionnés, les meilleurs résultats on été obtenus avec SPASS, qui est ainsi devenu notre démonstrateur principal. Les succès de SPASS sur des problèmes générés par SAD sont dûs en particulier à sa technique originale de traitement des littéraux qui représentent les «sortes» de termes.

Le système SAD est développé en Haskell, un langage fonctionnel pure [32]. Le démonstrateur Moses est implanté en C.

3.2 Vérificateur

Le vérificateur est une procédure récursive qui implante le calcul **CTC** «à l'envers» : à partir de conclusion vers des prémisses. Supposons que le vérificateur est en train de considérer le séquent $\Gamma \triangleright_G \mathbb{A}, \Delta$. Supposons aussi que $\mathbb{A} = (\mathrm{T}\ F\ [\Lambda])$ est une phrase.

Toutes les images-formules dans Γ sont ontologiquement correctes en vue de leurs prédécesseurs respectifs dans Γ, la thèse courante G est ontologiquement correcte en vue de Γ et $\mathcal{FV}(G) \subseteq \mathcal{FV}(\Gamma)$. Nous allons démontrer dans la section 3.2.2 que ces invariants sont maintenus par la motivatrice au cours de la vérification. Finalement, nous supposons que F ne contient pas d'occurrences de $\mathfrak{T}$ (ou qu'ils sont toutes remplacées par G).

Vérification ontologique. Tout au début, l'image-formule F passe au contrôle ontologique. La procédure de contrôle traverse la formule de gauche à droite en profondeur d'abord. Aucune occurrence de terme ou de sous-formule atomique n'est travaillée avant que tous ses prédécesseurs logiques ainsi que ses termes-arguments ne soient ontologiquement vérifiés et leurs *évidences* générées. La génération d'évidences est décrite dans la section 3.2.1.

Pour une occurrence donnée $F|_\tau$ de la forme $t\,\varepsilon\,N(\vec{s})$, $P(\vec{s})$ ou $f(\vec{s})$, toute définition et extension de signature $\mathbb{D}$ dans Γ est testée pour l'applicabilité, comme défini dans la section 2.3.1. Rappelons que cela consiste en démonstration de la conjonction (notons-la H) des conditions-assomptions dans $\mathbb{D}$ instanciées par les termes $\vec{s}$ dans la position de $F|_\tau$ en vue de Γ.

Les définitions et les extensions de signature sont considérées dans l'ordre des plus récentes aux plus anciennes. En fait, Γ est examiné en deux passages : «léger» et «lourd». Au passage «léger», les conditions instanciées H sont seulement simplifiées à l'aide des évidences générées pour les termes $\vec{s}$ dans la position de $F|_\tau$. Cette simplification est aussi décrite dans la section 3.2.1. Si nous obtenons $\top$ comme résultat, H est alors démontrée et $\mathbb{D}$ est applicable. Si aucune définition et extension de signature dans Γ ne réussit le test «léger», le contrôleur ontologique examine Γ à nouveau, cette fois en utilisant le raisonneur pour chaque tâche $\Gamma \vdash \langle\!|H|\!\rangle^F_\tau$.

Vérification logique. Ontologiquement contrôlée et fortifiée avec les évidences générées, la section $\mathbb{A}$ est ensuite travaillée selon son genre T. Si $\mathbb{A}$ est une affirmation, une sélection ou un cas de preuve, alors une sous-vérification est lancée pour la démonstration Λ. Si Λ est vide, le système procède immédiatement à la recherche de preuve pour la thèse initiale. Cette thèse initiale est prise conformément aux règles de **CTC** : la formule F si $\mathbb{A}$ est une affirmation ; la clôture existentielle de F si $\mathbb{A}$ est une sélection ; la thèse courante G si $\mathbb{A}$ est un cas de preuve (F devient une prémisse supplémentaire) ; la thèse d'induction $\mathrm{IT}^{\prec}_t(F)$ si Λ est une démonstration par induction sur t. Rappelons que les démonstrations par induction sont explicitement dénotées dans le texte ForTheL et le terme d'induction t peut y être mentionné aussi. S'il est omis, la variable sous le quantificateur universel le plus externe est prise comme t.

Une fois que Λ est entièrement vérifiée, la nouvelle thèse G' dans la vérification courante est construite par la *motivatrice* comme la section 3.2.2 le décrit.

Si $\mathbb{A}$ se trouve une assomption non-motivée, c'est-à-dire que la motivatrice n'arrive pas à la lier à la thèse courante G, alors la transition vers la nouvelle thèse est non-triviale et la tâche $\Gamma \vdash \forall\vec{x}\,(F \supset G') \supset G$, for $\vec{x} = \mathcal{DV}_\Gamma(F)$ (voir la section 2.3.3) appartient au raisonneur.

Autrement si $\mathbb{A}$ et les autres assomptions dans la vérification courante sont toutes motivées ; et nous sommes en train de vérifier une démonstration par induction ; et les variables libres de l'hypothèse d'induction sont enfin déclarées — si toutes ces conditions sont satisfaites, alors l'hypothèse d'induction est ajoutée à Γ.

Ensuite le vérificateur passe à sa tâche suivante : $\Gamma, \mathbb{A} \triangleright_{G'} \Delta$. Si Δ est vide et la thèse finale G' n'est pas la vérité, nous appelons le raisonneur pour démontrer G' et ainsi compléter la vérification courante.

Dans l'annexe D, nous étudions une session complète de vérification (du Texte sur l'ensemble vide de la section 1.5) dans le système SAD.

3.2.1 Génération d'évidences

Les évidences sont une variété particulière de propriétés locales (voir la section 2.2.2) que le système accumule et utilise au cours de la vérification. Une *évidence* pour une occurrence d'un terme t est un littéral L qui est localement valide et ontologiquement correct dans la position de t et contient t comme sous-terme.

Le but le plus important des évidences est de conserver l'information sur les «types» de termes. Cette information est habituellement exprimée par des formules atomiques $t\,\varepsilon\,N(\vec{s})$. Certaines propriétés simples, telles que être non-vide ou être premier, peuvent aussi nous servir.

Il peut exister beaucoup de tels littéraux. Toute procédure particulière de génération d'évidences établit sa propre sous-classe d'évidences «saisies» et nous ne pouvons pas en ce moment proposer une méthode qui serait clairement préférable. En général, l'accumulation d'évidences ne doit pas consommer beaucoup de ressources, car on effectue cette procédure pour toute oc-

currence de terme rencontrée. De l'autre côté, l'ensemble des évidences saisies ne doit pas être trop maigre pour être utile à la vérification.

Une fois générée, une évidence accompagne son occurrence de terme pendant toute la session de vérification. Nous appelons «fortifiées» les termes et formules qui ont été exposés à la génération d'évidences. Les transformations utilisées par le vérificateur et le raisonneur (réduction de thèse, application de définitions, etc) préservent la validité locale et la correction ontologique d'évidences de termes et formules fortifiés même quand ces derniers sont mis dans un nouveau contexte : déplacés plus bas dans le texte ou instancés ou substitués dans une autre formule. Néanmoins, dans certains cas il est utile de recalculer des évidences pour saisir des nouvelles évidences qui pouvait apparaître.

Génération d'évidences. Le générateur d'évidences implanté dans SAD reçoit du contrôleur ontologique une occurrence de terme $F|_\tau$ (notons ce terme t) avec l'ensemble de prémisses Γ. Le terme t est ontologiquement correct. Les propres sous-termes de t ainsi que les prédécesseurs logiques de t dans F et dans Γ sont ontologiquement corrects et fortifiés.

Les transformations décrites plus bas sont toutes effectuées localement, dans la position τ de la formule F. Nous pouvons supposer que aucune variable bornée dans F n'intervient dans $\mathcal{FV}(\Gamma)$. Alors, pour la simplicité, nous allons traiter comme constantes les variables connues dans τ : i.e. celles déclarées dans Γ ou bornée par un quantificateur au-dessus de τ. Ainsi tout prédécesseur de F dans Γ et tout prédécesseur de t dans F peut être vu (dans la position τ) comme une formule close. Autrement dit, dans «l'environnement local» $\Gamma \vdash \langle\!| \cdot |\!\rangle^F_\tau$, nous effaçons des quantificateurs universels externes dans l'image locale (en présumant que les collisions des variables n'apparaissent pas) et considérons des variables libres comme constantes.

L'ensemble d'évidences $\mathcal{E}^F_\tau(\Gamma)$ assigné au terme t est construit comme suit. Nous commençons par l'ensemble $\mathcal{E} = \{t \approx t\}$ et considérons l'ensemble S des évidences saisies pour les sous-termes de t dans F. Par définition d'une évidence, tout littéral $L \in S$ est localement valide et ontologiquement correct dans la position τ : $\Gamma \vdash \langle\!| L |\!\rangle^F_\tau$ et $\Gamma \blacktriangleright \langle\!| L |\!\rangle^F_\tau$. Tout littéral dans S qui contient t comme sous-terme est ajouté à $\mathcal{E}$.

Ensuite nous examinons les prédécesseurs logiques de τ dans $\langle\!| \cdot |\!\rangle^F_\tau$ et dans Γ (dans l'ordre de la fin au début du texte), en tirant des antécédents et des formules-prémisses un par un. Toute telle formule H est localement valide et ontologiquement correcte dans la position τ : $\Gamma \vdash \langle\!| H |\!\rangle^F_\tau$ et $\Gamma \blacktriangleright \langle\!| H |\!\rangle^F_\tau$.

Nous «résolvons» la formule H avec l'ensemble $\mathcal{E} \cup S$ comme suit. Nous comparons des sous-formules atomiques dans H avec des littéraux de $\mathcal{E} \cup S$, appliquons des substitutions locales correspondantes à H (à condition que ces substitutions soient permises) et contractons la formule obtenue en vue de $\mathcal{E} \cup S$. Les substitutions locales et la contraction en vue sont introduites dans la section 2.2.3. Si après un certain nombre de substitutions-contractions nous obtenons une formule $H'[L]_\pi$ telle que $\pi \in \Pi^+_\wedge(H')$ et t intervient dans L, nous ajoutons L à $\mathcal{E}$. (Bien sûr, ce n'est pas l'algorithme actuellement implanté, mais une approximation raisonnable.)

Démontrons que le littéral L est vraiment une évidence pour t. D'après les lemmes 2.2.22(a) et 2.2.17, on a $\Gamma \vdash \langle\!| H'[L]_\pi |\!\rangle^F_\tau$. Alors $\Gamma \vdash \langle\!| L |\!\rangle^F_\tau$ d'après le lemme 2.2.16. Quant à la correction ontologique, les propriétés locales de termes et d'atomes dans H sont préservées (les lemmes 2.2.15, 2.2.22(b,c)). Les termes introduits dans H par des substitutions locales proviennent de $\mathcal{E} \cup S$. Ainsi ils sont ontologiquement corrects eux aussi. Alors $\Gamma \blacktriangleright \langle\!| H'[L]_\pi |\!\rangle^F_\tau$. Comme $H'[L]_\pi$ est effectivement une conjonction dont L est une partie et comme $H'[L]_\pi$ est localement valide dans la position τ, nous obtenons $\Gamma \blacktriangleright \langle\!| L |\!\rangle^F_\tau$.

De plus, si le terme $F|_\tau$ est une variable x et la formule H est de la forme $x \approx s$, nous prenons les évidences saisies pour s, remplaçons s par x et ajoutons les littéraux obtenus à $\mathcal{E}$ (l'égalité $x \approx s$ est ajouté à $\mathcal{E}$ aussi). Il est clair que toute propriété locale de s est valide dans la position τ dans F et que x est localement égal à s dans τ. Une telle «succession» est surtout importante quand x est déclaré dans H, et il n'y a donc aucune information sur x sauf ce qui est connu sur s.

Quand Γ est traversé jusqu'au début, l'ensemble $\mathcal{E}^F_\tau(\Gamma) = \mathcal{E}$ est assigné à l'occurrence $F|_\tau$ en vue de Γ.

Exemple 3.2.1. Considérons l'image-formule F de la phrase (9.0) de l'exemple dans la section 2.4 (figure 2.2) :

$$F = |(9.0)| = \forall i\,(i\,\varepsilon\,\mathrm{NatNum} \supset 0 + i \approx i)$$

La liste des prédécesseurs logiques Γ de cette phrase consiste de huit sections haut-niveau : $\Gamma = (0), (1), \ldots, (8)$. Il y a cinq occurrences de termes dans F : $F|_{0.0.0} = i$, $F|_{0.1.0.0} = 0$, $F|_{0.1.0.1} = i$, $F|_{0.1.0} = (0 + i)$, et $F|_{0.1.1} = i$, dans l'ordre de passage du contrôleur ontologique.

L'ensemble d'évidences $\mathcal{E}^F_{0.0.0}(\Gamma)$ pour la première occurrence du terme i ne contient que l'évidence triviale $i \approx i$. En effet, il n'y a pas de propres sous-termes dans i pour former l'ensemble S. Ensuite, aucune formule dans le contexte local $\langle\!|\, \cdot \,|\!\rangle^F_{0.0.0} = \forall i\,(\cdot)$ ou dans Γ n'offre d'autres évidences pour i.

L'ensemble d'évidences $\mathcal{E}^F_{0.1.0.0}(\Gamma)$ pour le terme 0 contient (à part l'évidence triviale) le littéral $0\,\varepsilon\,\mathrm{NatNum}$. Cette évidence est produite par l'extension de signature (1) : $\forall x\,(x \approx 0 \supset x\,\varepsilon\,\mathrm{NatNum})$ se transforme dans $(0 \approx 0 \supset 0\,\varepsilon\,\mathrm{NatNum})$ par substitution locale, ce qui nous donne $0\,\varepsilon\,\mathrm{NatNum}$ après une contraction. L'évidence générée peut ensuite participer dans la résolution avec l'image-formule de la section (0), $\forall x\,(x\,\varepsilon\,\mathrm{NatNum} \supset \top)$. Pourtant, la formule obtenue $\top$ n'est pas une évidence pour 0.

L'ensemble d'évidences $\mathcal{E}^F_{0.1.0.1}(\Gamma)$ pour la deuxième occurrence du terme i est plus grand que pour la première. Premièrement, il contient le littéral $i\,\varepsilon\,\mathrm{NatNum}$, qui provient du contexte local de cette occurrence : $\langle\!|\, \cdot \,|\!\rangle^F_{0.1.0.1} = \forall i\,(i\,\varepsilon\,\mathrm{NatNum} \supset \cdot)$. Ce littéral peut être ensuite résolu presque avec toute image-formule dans Γ. En montant de (8) à (0), nous ajoutons à $\mathcal{E}$ les littéraux suivants : $i \prec \mathrm{succ}\, i$ (par la section (8) ; notez que la variable i dans $|(8)|$ est une variable bornée qui est instanciée à la variable localement libre i de l'évidence $i\,\varepsilon\,\mathrm{NatNum}$), $i + \mathrm{succ}\, i \approx \mathrm{succ}\,(i + i)$ (par (7), après deux résolutions), $i + 0 \approx i$ (par (6)), $(i + i)\,\varepsilon\,\mathrm{NatNum}$ (par (3)), $(\mathrm{succ}\, i)\,\varepsilon\,\mathrm{NatNum}$ (par (2)) et $(\mathrm{succ}\,(i + i))\,\varepsilon\,\mathrm{NatNum}$ (par (2) et l'évidence $(i + i)\,\varepsilon\,\mathrm{NatNum}$).

L'ensemble d'évidences $\mathcal{E}^F_{0.1.0}(\Gamma)$ pour le terme $(0 + i)$ est défini comme suit. L'ensemble initial de satellites $\mathcal{E} \cup S$ contient l'égalité $(0 + i) \approx (0 + i)$ et toutes les évidences saisies pour 0 et i (qui ne contiennent pas d'occurrences de $(0 + i)$ et donc ne sont pas des évidences pour ce terme). En résolvant cet ensemble avec les image-formules dans Γ nous obtenons trois évidences de plus : $0 + (\mathrm{succ}\, i) \approx \mathrm{succ}\,(0 + i)$ (d'après (7)), $(0 + i)\,\varepsilon\,\mathrm{NatNum}$ (d'après (3)) et $(\mathrm{succ}\,(0 + i))\,\varepsilon\,\mathrm{NatNum}$ (d'après (2) et l'évidence précédente).

L'ensemble d'évidences $\mathcal{E}^F_{0.1.1}(\Gamma)$ pour la troisième occurrence du terme i est le même que pour la deuxième : $\mathcal{E}^F_{0.1.1}(\Gamma) = \mathcal{E}^F_{0.1.0.1}(\Gamma)$.

La plupart des évidences saisies sont inutiles pour le raisonnement conséquent. À cet égard, la génération d'évidences ressemble à toute démonstration automatique qui produit des innombrables inférences en vain pour quelque pas importants. Par ailleurs, la génération d'évidences est une procédure qui s'arrête après un temps fini, car nous traversons le contexte une seule fois et à chaque étape le nombre de résolutions possibles est fini.

Le nombre d'évidences générées peut être exponentiellement supérieur à celui d'atomes dans Γ. Supposons que pour un terme i nous avons obtenu une évidence $i \prec j$. Ensuite, en rencontrant une prémisse de la forme $\forall x, y(x \prec y \supset x \prec f_1(y))$, nous générerons la nouvelle évidence $i \prec f_1(j)$. Une autre prémisse $\forall x, y(x \prec y \supset x \prec f_2(y))$ va doubler la taille de $\mathcal{E}$ en y ajoutant deux autres évidences : $i \prec f_2(j)$ et $i \prec f_2(f_1(j))$. Ainsi, en passant par une chaîne de n prémisses, jusqu'à $\forall x, y(x \prec y \supset x \prec f_n(y))$, nous obtiendrons 2^n évidences différentes. À présent, nous ne pouvons proposer aucun moyen raisonnable d'éviter une telle explosion.

Simplification par évidences. Nous pouvons simplifier une formule fortifiée F en contractant toute sous-formule atomique A en vue de l'ensemble d'évidences saisies pour les termes dans A. Par exemple, si la formule F dans l'exemple 3.2.1 était $\forall i\,(i\,\varepsilon\,\mathrm{NatNum} \supset 0\,\varepsilon\,\mathrm{NatNum} \supset 0 + i \approx i)$, nous pourrions éliminer l'antécédent $0\,\varepsilon\,\mathrm{NatNum}$, car il est une évidence for 0 dans sa position et donc est localement valide.

C'est la simplification par évidences que nous appliquons pour vérifier des conditions de définitions et d'extensions de signature en testant leur applicabilité. Rappelons que les évidences «voyagent» avec ses termes et donc se trouvent dans les conditions instanciées. Par exemple, l'extension de signature pour l'addition, marquée `Add` à la figure 2.2, sera trouvée applicable (au

passage «léger» du contrôle ontologique) dans la position de $0 + i$ dans la formule F ci-dessus. En effet, dans la condition instanciée $0 \,\varepsilon\, \mathrm{NatNum} \wedge i \,\varepsilon\, \mathrm{NatNum}$, les deux formules atomiques seront éliminées grâce aux évidences pour 0 et i.

Outre cela, c'est la simplification par évidences que nous appliquons en réalité pour contracter la formule instanciée H en vue de $\mathcal{E} \cup S$ pendant la génération d'évidences : comme la formule H est instanciée avec des termes de $\mathcal{E} \cup S$, elle devient fortifiée par leurs évidences.

3.2.2 Procédure «motivatrice»

La *motivatrice*, comme nous l'avons expliqué ci-dessus, est une procédure qui, ayant reçu la tâche courante de vérification $\Gamma \triangleright_G \mathbb{A}, \Delta$, compare la thèse courante G avec la phrase suivante $\mathbb{A}$ et suggère la nouvelle thèse G'. Dans ce qui suit, F dénote l'image-formule de $\mathbb{A}$ et $\vec{x} = \mathcal{DV}_\Gamma(\mathbb{A})$ l'ensemble (possiblement vide) de variables déclarées dans $\mathbb{A}$. Notons que les variables de $\vec{x}$ ne peuvent pas intervenir librement dans G, car toutes les variables libres de G doivent être déclarées dans Γ et donc ne peuvent pas être déclarées dans $\mathbb{A}$.

Les décisions de la motivatrice sont guidées par le genre de $\mathbb{A}$ (ou plutôt par la forme de la prémisse de «transition de thèse» dans les règles de **CTC**, qui est à son tour différente pour les différents genres de phrase).

Si $\mathbb{A}$ est une affirmation ou une sélection, la nouvelle thèse est juste la contraction de celle courante en vue de $|\mathbb{A}|$: $G' = \overline{\mathsf{C}}_F\, G$. La prémisse de «transition de thèse» $\Gamma \vdash \exists \vec{x}\, (F \wedge G') \supset G$ est visiblement valide. En effet, $\overline{\mathsf{C}}_F\, G$ est équivalent à G en vue de F (le lemme 2.2.17), et donc G' peut être remplacé dans $F \wedge G'$ par G (le théorème 2.2.7). Aussi, G' est ontologiquement correct en vue de $\Gamma, \mathbb{A}$.

Si $\mathbb{A}$ est une assomption, nous choisissons la nouvelle thèse selon que si $\mathbb{A}$ soit motivée ou non. Sommairement, nous considérons une assomption comme motivée si elle est impliquée par la négation de la thèse courante. Pour donner des définitions plus précises, il nous faut quelque notions supplémentaires.

Une variable bornée dans une formule H est dite *interne* si elle est introduite dans H soit par l'analyser syntaxique comme un nom frais pour une occurrence de notion (voir la règle de transformation (1.4.1(1)) dans la section 1.4), soit par une transformation quelconque dans le vérificateur, e.g. par application d'une définition. Notre intention est de saisir des variables dont les noms n'étaient pas écrits explicitement dans le texte vérifié.

Nous disons qu'on peut *libérer* une variable bornée v dans une formule H si des substitutions locales dans v sont permises et si la formule $\forall v\, H[[v/v]]^+_\pi$ est équivalente à H (π est toute position sous le quantificateur sur v). Pour cela, le quantificateur sur v ne doit pas se trouver sous des quantificateurs de la polarité opposée, la substitution locale ne doit pas introduire de constantes booléennes et la variable v ne doit pas intervenir librement dans H. D'après le lemme 2.2.22, la formule obtenue avec la variable v libérée, $H[[v/v]]^+_\pi$, hérite toutes les propriétés locales de la formule initiale.

Une *variation* d'une formule H est toute formule qui peut être obtenue à partir de H par une série de transformations suivantes :

- appliquer une définition d'une façon destructive (la section 3.3.3) à un atome ;
- renommer une variable interne en une variable de $\vec{x}$;
- libérer une variable de $\vec{x}$.

Une assomption $\mathbb{A}$ est *motivée* s'il y existe une variation G° de G telle que $\overline{\mathsf{C}}_{\neg G^\circ}\, F = \top$. Dans ce cas, la nouvelle thèse est la contraction : $G' = \overline{\mathsf{C}}_F\, G^\circ$.

Remarquez que la clôture universelle $\forall \vec{x}\, G^\circ$ est ontologiquement correcte et équivalente à G en vue de Γ. D'ailleurs, $\forall \vec{x}\, (F \supset G^\circ)$ est équivalent à $\forall \vec{x}\, G^\circ$. En effet, $(F \supset G^\circ) \Leftrightarrow (\neg G^\circ \supset \neg F) \Leftrightarrow (\neg G^\circ \supset \neg \top) \Leftrightarrow G^\circ$. Comme G' est équivalent à G° en vue de F, la prémisse de «transition de thèse» $\Gamma \vdash \forall \vec{x}\, (F \supset G') \supset G$ est valide. Aussi, G' est ontologiquement correct en vue de $\Gamma, \mathbb{A}$.

S'il y a plusieurs chaînes de variation de la thèse G qui font $\mathbb{A}$ assomption motivée, la motivatrice va choisir une parmi les plus courtes.

Dans le cas où $\mathbb{A}$ est une assomption non-motivée, i.e. nous n'avons trouvé aucune variation convenable, la nouvelle thèse G' est choisie comme décrit dans la section 2.3.3. Notamment, s'il

y a des occurrences de $\mathfrak{T}$ dans les image-formules de sections dans Δ, nous prenons $G' = G$. Autrement, $G' = |\Delta|_\Gamma$. Dans les deux cas, la prémisse $\Gamma \vdash \forall \vec{x}\,(F \supset G') \supset G$ est non-triviale et appartient au raisonneur.

Exemple 3.2.2. Soit G la $S \subseteq T$, où S et T sont déclarés dans Γ comme ensembles (ainsi que G est ontologiquement correct en vue de Γ). Soit $\mathbb{A}$ une assomption (`assume` $(x \in S)$) et la variable x déclarée in $\mathbb{A}$ (ainsi que x n'intervient pas librement dans Γ). Nous pouvons démontrer que $\mathbb{A}$ est motivée en construisant une variation de G :

1. Nous appliquons la définition de $\subseteq$ d'une façon destructive et obtenons $\forall z\,(z \in S \supset z \in T)$.
2. La variable z est introduite par la transformation précédente et donc est une variable interne. Nous la renommons alors en x et nous obtenons $\forall x\,(x \in S \supset x \in T)$.
3. Le quantificateur sur x se trouve en haut de la formule. Ainsi nous pouvons libérer x et obtenir la variation cherchée $G^\circ : x \in S \supset x \in T$.

La négation de G° est équivalente à la conjonction $x \in S \wedge x \notin T$ qui évidemment implique l'image-formule de $\mathbb{A}$. Ainsi $\mathbb{A}$ est motivée et la nouvelle thèse G' est la formule $x \in T$ qui est ontologiquement correct en vue de $\Gamma, \mathbb{A}$ et satisfait la prémisse $\forall x\,(x \in S \supset G') \supset S \subseteq T$.

Exemple 3.2.3. Supposons que dans l'exemple précédent, G est la formule $S \subseteq T \wedge P(S)$, où la relation P est définie sur 2^T. La partie gauche de la conjonction ainsi assure la correction ontologique de la partie droite. Maintenant, si nous pouvions suivre le chemin de l'exemple 3.2.2, nous obtiendrions la nouvelle thèse $x \in T \wedge P(S)$ qui est ontologiquement incorrecte en vue de $\Gamma, \mathbb{A}$, car ce que nous savons sur S dans la position de $P(S)$ n'est pas suffisant. En effet, nous savons seulement que S est un ensemble (d'après Γ), qu'il y a un élément x dans S (d'après $\mathbb{A}$) et que ce x appartient aussi à T (d'après la partie gauche de la nouvelle thèse). Tout cela n'implique pas que S est un sous-ensemble de T.

Or, nous ne pouvons pas suivre l'exemple précédent. Notamment, le troisième pas dans la variation ci-dessus nous est interdit : la variable x ne peut pas être libérée dans la formule $\forall x\,(x \in S \supset x \in T) \wedge P(S)$, car la substitution locale remplace $P(S)$ par $\top$. L'assomption $\mathbb{A}$ est ainsi non-motivée.

Et même si on relâche les règles et on construit la «variation» $(x \in S \supset x \in T) \wedge P(S)$ (qui n'est pas ontologiquement correcte mais est équivalente, sous $\forall x$, à la thèse initiale), l'assomption $\mathbb{A}$ est toujours non-motivée. En effet, la négation de cette «variation» est équivalente à la formule $(x \in S \wedge x \notin T) \vee \neg P(S)$ qui n'implique pas $x \in S$, l'image-formule de $\mathbb{A}$.

3.3 Raisonneur

Dans la version actuelle de SAD, la procédure du raisonneur est assez simple. Pour une tâche de preuve $\Gamma \vdash G$, nous commençons par «filtrer» le contexte Γ, comme la section 3.3.1 le décrit. Nous allons dénoter par Γ^* l'ensemble de prémisses transformé.

Ensuite nous créons une pile vide de sous-tâches de preuve. Nous décomposons le but G en un ensemble de sous-buts $\{G_1, \ldots, G_n\}$ dont la conjonction est équivalente à G (voir la section 3.3.2). Et nous plaçons les sous-tâches $(\Gamma^* \vdash G_1), \ldots, (\Gamma^* \vdash G_n)$ au sommet de la pile.

Tant qu'il y a de sous-tâches dans la pile, nous en saisissons une du sommet et simplifions sa formule-but à l'aide d'évidences, comme décrit dans la section 3.2.1. Si la formule ainsi obtenue est $\top$, la sous-tache est alors surnommée «triviale» et le raisonneur passe à la suivante. Autrement, le démonstrateur automatique est lancé pour déduire le but à partir de prémisses. Si le démonstrateur y arrive, le raisonneur se prend à la sous-tâche suivante. Si le démonstrateur échoue, le raisonneur essaie de faciliter le problème par l'application de définitions, comme décrit dans la section 3.3.3.

On applique des définitions aux occurrence choisies dans les prémisses ainsi que dans le but d'une sous-tache. Ensuite le nouveau but est décomposé et les sous-tâches obtenues sont mises sur la pile. Et la boucle principale du raisonneur se poursuit jusqu'à ce que la pile se vide ou le raisonneur se rend.

Le raisonneur reconnaît sa défaite soit quand il n'y a plus d'occurrences auxquelles une définition peut être appliquée, soit quand une sous-tâche reste non-démontrée après un nombre

fixé d'essais consécutifs (par défaut 7 pour les tâches de preuve ordinaires et 3 pour les tâches qui proviennent de la vérification ontologique).

La procédure du raisonneur n'utilise pas de retour-arrière : on décompose, simplifie et applique des définitions d'une façon progressive et on ne revient jamais aux états précédents de la tâche de preuve.

3.3.1 Filtrage de contexte

La procédure de filtrage parcourt la liste de prémisses Γ et «bloque» des sections considérées non-nécessaires pour la tache de preuve. Dans la version actuelle de SAD, cette procédure s'applique une fois, au début de la boucle du raisonneur, comme expliqué ci-dessus. Pourtant, il peut se trouver avantageux de changer le filtre à chaque itération. Dans ce qui suit nous expliquons comment les sections à bloquer sont choisies et qu'est-ce qu'est une section bloquée.

Avant tout, il faut savoir si l'auteur du texte a suggéré son propre filtre pour le but en question. Supposons que la tâche de preuve courante provient d'une $\vdash$-prémisse dans une règle de **CTC**. On considère alors la phrase d'où la thèse courante est venue, i.e. la phrase qui a initié la vérification courante. Si cette phrase (affirmation, sélection, ou cas de preuve) est muni d'une liste de *références*, on bloque toutes les sections haut-niveau sauf celles mentionnées.

S'il n'y pas de références explicites ou si la tâche de preuve courante provient de la vérification ontologique, nous bloquons toutes les sections-définitions. La motivation pour cela est expliquée dans la section 3.3.3.

Les prémisses bloquées ne sont pas enlevées de Γ (en effet, si on enlève toutes les définitions, le reste ne sera plus ontologiquement correct). Par contre, ces prémisses ne sont pas exposées aux transformations suivantes effectuées par le raisonneur et, le plus important, elles ne sont jamais envoyées au démonstrateur. Pour toute section bloquée $\mathbb{C}$, nous ajoutons à Γ, juste après $\mathbb{C}$, son substitut allégé où l'image-formule de $\mathbb{C}$ (notons-le H) est remplacée par une certaine simplification de H.

Les règles de génération de cette formule allégée peuvent être assez arbitraires, à condition qu'on obtienne une conséquence logique de H qui soit ontologiquement correcte en vue des prédécesseurs logiques de $\mathbb{C}$ et de la section $\mathbb{C}$ elle-même. Il est aussi désirable que certaines formes importantes, telles que démodulateurs potentiels et les «règles de sorte», soient préservées dans transformation. Par «règles de sorte» nous entendons des implications de la forme $(x_1 \,\varepsilon\, N_1 \wedge \cdots \wedge x_n \,\varepsilon\, N_n) \supset f(x_1, \ldots, x_n) \,\varepsilon\, N$ et $x \,\varepsilon\, N_1 \supset x \,\varepsilon\, N_2$. Ces implications permettent au démonstrateur de produire ses propres «évidences» pour des termes générés au cours de l'inférence. En absence de telles règles, le travail du démonstrateur serait trop facilité.

Dans notre implantation actuelle, les formules allégées sont générées par des séries de remplacements positifs (définis dans la section 2.2.3). Dans les images-formules de définitions, nous remplaçons l'équivalence principale $u \approx f(\vec{v}) \equiv F$ par $F[f(\vec{v})/u]$. De même, dans les images-formules d'extensions de signature, nous remplaçons l'implication principale $u \approx f(\vec{v}) \supset F$ par $F[f(\vec{v})/u]$. Ensuite nous remplaçons d'autres équivalences ($\equiv$) par les conjonctions d'implications ($\sim$), pour que toute occurrence ait une polarité non-neutre.

Appelons une formule atomique *quasi-plate* si tout argument est soit une variable, soit un terme clos. Dans les positions négatives, nous remplaçons par $\bot$ toute sous-formule atomique qui n'est pas quasi-plate. La skolémisation est prise en compte : une sous-formule n'est pas considérée quasi-plate si elle contient des variables à remplacer par un symbole de Skolem (sauf s'il s'agit d'une constante, ce qui donne un terme clos). Cela finit la transformation.

Il est facile à voir que cette procédure satisfait nos demandes. Il est pas moins facile, pourtant, de proposer quelque autre procédure aussi conforme. Pour en choisir les meilleures, on doit se guider sur des expériences.

3.3.2 Décomposition de but

Étant donné une formule-but G, nous la transformons en un ensemble de clauses. Ici nous entendons par *clauses* les listes disjonctives ordonnées de formules. Les règles ci-dessous s'appliquent aux clauses particulières. Après l'application d'une règle, la clause-prémisse est remplacée par les clauses-conclusions dans l'ensemble construit.

$$\frac{C_1,\ F \equiv G,\ C_2}{C_1,\ F \sim G,\ C_2} \qquad \frac{C_1,\ F \supset G,\ C_2}{C_1,\ \neg F,\ G,\ C_2} \qquad \frac{C_1,\ \forall u\, F,\ C_2}{C_1,\ F[z/u],\ C_2}$$

$$\frac{C_1,\ F \vee G,\ C_2}{C_1,\ F,\ G,\ C_2} \qquad \frac{C_1,\ F \wedge G,\ C_2}{C_1,\ F,\ C_2 \qquad C_1,\ \neg F,\ G,\ C_2} \qquad \frac{C_1,\ \neg H,\ C_2}{C_1,\ H^{\neg},\ C_2}$$

Ici, C_1, C_2 dénotent des sous-clauses, possiblement vides. La formule H dénote toute formule non-atomique. La variable z est fraîche par rapport à C_1, C_2, F et Γ. La dérivation commence par l'ensemble $\{G\}$ (où G est considéré comme clause à un élément) et s'arrête quand aucune règle ne peut être appliquée à aucune clause. Ensuite les clauses dans l'ensemble obtenu sont remplacées par les disjonctions correspondantes. L'ensemble final, dénoté par $\mathcal{S}plit(G)$, est l'ensemble de sous-buts.

Lemme 3.3.1. *Pour tout Γ et G, $\Gamma \vdash G$ si et seulement si $\Gamma \vdash \bigwedge \mathcal{S}plit(G)$.*

Démonstration. Il suffit de démontrer que chaque règle ci-dessus produit des clauses équivalentes à celle initiale. La conjonction $F \wedge (\neg F \vee G)$ est équivalente à $F \wedge G$. La règle d'élimination de quantificateur est une skolémisation habituelle dans le but. On peut aussi dire que le quantificateur sur u est levé en haut de la clause et du $\vdash$-séquent même. Nous remplaçons u par une variable fraîche pour éviter des collisions. Pour les autres règles l'affirmation est évidente. □

Lemme 3.3.2. *Toute sous-formule atomique dans $\mathcal{S}plit(G)$ intervient dans G et hérite toutes les propriétés locales de cette occurrence (modulo remplacement $[z/u]$ où nécessaire). En conséquent, si G est ontologiquement correct en vue de Γ, toute formule de $\mathcal{S}plit(G)$ l'est aussi.*

Démonstration. La première partie de l'assertion est évidente puisque la transformation n'introduit pas de nouvelles sous-formules atomiques dans $\mathcal{S}plit(G)$. Pour démontrer la deuxième partie, il suffit de voir que $\forall u\, F$ implique $F[z/u]$ et que $F \wedge G$ implique F et $\neg F \vee G$. D'après le lemme 2.2.21, nous pouvons remplacer toute composante d'une disjonction par une conséquence logique sans perdre des propriétés locales. □

3.3.3 Application de définitions

Les définitions, si on les traite comme des formules logiques ordinaires, sont assez difficile à travailler dans un démonstrateur automatique : elles sont disposées à gonfler l'espace de recherche et peuvent produire des inférences redondantes difficiles à détecter. De plus, il nous coûte des pas d'inférence additionnels à éliminer les conditions de définitions. Voila pourquoi nous préférons cacher les définitions au démonstrateur, comme décrit dans la section 3.3.1, et les appliquer dans le raisonneur. Le contrôle ontologique préliminaire ici est opportun. Le raisonneur ne peut pas savoir précisément quelles occurrences doivent être «dépliées». Comme l'application de définitions totale, jusqu'aux symboles non-définis, est peu pratique, une certaine stratégie d'applications doit être établie.

Application destructive et conservatrice. Soit F une formule (image-formule) dans une tâche de preuve du raisonneur. Soit $F|_\pi$ une occurrence de la forme $f(s_1, \ldots, s_n)$, $P(s_1, \ldots, s_n)$ ou $s\,\varepsilon\, N(s_1, \ldots, s_n)$. Soit $\Gamma_1, \mathbb{D}, \Gamma_2$ la séquence des prédécesseurs logiques de F, où $\mathbb{D}$ est une définition applicable dans $F|_\pi$, comme défini dans la section 2.3.1.

La section $\mathbb{D}$ est de la forme $(\texttt{defn}\ D\ [(\texttt{assume}\, H_1), \ldots, (\texttt{assume}\, H_m), (\texttt{posit}\, S)])$. L'image-formule D est $\forall \vec{x}_1 (H_1 \supset \ldots \forall \vec{x}_m (H_m \supset S) \ldots)$ et $\vec{x}_i$ est $\mathcal{DV}_{\Gamma_1, H_1, \ldots, H_{i-1}}(H_i)$. L'image-formule S est de la forme

$$\begin{array}{ll} P(x_1, \ldots, x_n) \equiv C & \text{si } F|_\pi \text{ est } P(s_1, \ldots, s_n) \\ \forall v\, (v\,\varepsilon\, N(x_1, \ldots, x_n) \equiv C) & \text{si } F|_\pi \text{ est } s\,\varepsilon\, N(s_1, \ldots, s_n) \\ \forall v\, (v \approx f(x_1, \ldots, x_n) \equiv C) & \text{si } F|_\pi \text{ est } f(s_1, \ldots, s_n) \end{array}$$

où $\{x_1, \ldots, x_n\} = \vec{x}_1 \sqcup \cdots \sqcup \vec{x}_m$ et $\mathcal{FV}(C) \subseteq \{v\} \sqcup \{x_1, \ldots, x_n\}$. Nous pouvons renommer les variables dans $\mathbb{D}$ ainsi que les variables libres de $F|_\pi$ (et de la formule atomique correspondante $F|_{\hat{\pi}}$ si $F|_\pi$ est un terme) n'appartiennent pas à l'ensemble $\{v, x_1, \ldots, x_n\}$.

Soit τ la position de S dans D et σ la substitution $[s_1/x_1, \ldots, s_n/x_n]$. L'instance locale $D[\sigma]^+_\tau$ est la formule $(H_1\sigma \supset \ldots (H_m\sigma \supset S\sigma)\ldots)$. Proprement dit, nous appliquons à chaque H_i juste une partie de σ, en ne remplaçant que des variables de $\vec{x}_1 \sqcup \cdots \sqcup \vec{x}_i$, mais nous pouvons simplifier la notation, étant donné que les variables de $\vec{x}_{i+1} \sqcup \cdots \sqcup \vec{x}_m$ n'interviennent pas librement dans H_i (sinon elles ne seraient pas déclarées plus bas). Par l'applicabilité de $\mathbb{D}$, la conjonction $H_1\sigma \wedge \cdots \wedge H_m\sigma$ est localement valide dans la position de $F|_\pi$. Alors d'après le lemme 2.2.22, la formule $S\sigma$ est localement valide et ontologiquement correcte dans $F|_\pi$ en vue de $\Gamma_1, \mathbb{D}, \Gamma_2$.

Le résultat *d'application destructive* de la définition $\mathbb{D}$ dans $F|_\pi$ est la formule :

$F[C\sigma]_\pi$	si $F|_\pi$ est $P(s_1, \ldots, s_n)$
$F[C[s/v]\sigma]_\pi$	si $F|_\pi$ est $s \,\varepsilon\, N(s_1, \ldots, s_n)$
$F[C[s/v]\sigma]_{\widehat{\pi}}$	si $F|_\pi$ est $f(s_1, \ldots, s_n)$ et $F|_{\widehat{\pi}}$ est $s \approx f(s_1, \ldots, s_n)$ ou $f(s_1, \ldots, s_n) \approx s$
$F[t\sigma]_\pi$	si $F|_\pi$ est $f(s_1, \ldots, s_n)$, C est $v \approx t$ et $v \notin \mathcal{FV}(t)$

Cette formule est équivalente à F d'après le théorème 2.2.7 et est ontologiquement correcte en vue de $\Gamma_1, \mathbb{D}, \Gamma_2$. Nous appelons cette transformation destructive comme l'occurrence initiale est effacée de la formule. Remarquez qu'on ne peut appliquer qu'une certaine sous-classe de définitions de fonction.

Supposons que F soit une prémisse dans la tâche de preuve. Le résultat *d'application conservatrice* de la définition $\mathbb{D}$ dans $F|_\pi$ est la formule :

$F[C\sigma \wedge F|_\pi]_\pi$	si $F|_\pi$ est $P(s_1, \ldots, s_n)$ et π est positif
$F[C\sigma \vee F|_\pi]_\pi$	si $F|_\pi$ est $P(s_1, \ldots, s_n)$ et π est négatif
$F[C[s/v]\sigma \wedge F|_\pi]_\pi$	si $F|_\pi$ est $s \,\varepsilon\, N(s_1, \ldots, s_n)$ et π est positif
$F[C[s/v]\sigma \vee F|_\pi]_\pi$	si $F|_\pi$ est $s \,\varepsilon\, N(s_1, \ldots, s_n)$ et π est négatif
$F[C[s/v]\sigma \wedge F|_\pi]_{\widehat{\pi}}$	si $F|_\pi$ est $f(s_1, \ldots, s_n)$, π est positif et $F|_{\widehat{\pi}}$ est $s \approx f(s_1, \ldots, s_n)$ ou $f(s_1, \ldots, s_n) \approx s$
$F[C[s/v]\sigma \vee F|_\pi]_{\widehat{\pi}}$	si $F|_\pi$ est $f(s_1, \ldots, s_n)$, π est négatif et $F|_{\widehat{\pi}}$ est $s \approx f(s_1, \ldots, s_n)$ ou $f(s_1, \ldots, s_n) \approx s$
$F[C[F|_\pi/v]\sigma \wedge F|_{\widehat{\pi}}]_{\widehat{\pi}}$	si $F|_\pi$ est $f(s_1, \ldots, s_n)$ et π est positif
$F[C[F|_\pi/v]\sigma \supset F|_{\widehat{\pi}}]_{\widehat{\pi}}$	si $F|_\pi$ est $f(s_1, \ldots, s_n)$ et π est négatif

Si F est la formule-but, les polarités de positions dans les conditions ci-dessus doivent être inversées. Autrement dit, la définition s'applique à la négation de F.

Considérons les six premiers cas de la définition ci-dessus. Il sont pareils aux trois premiers cas d'application destructive, sauf qu'on n'enlève pas l'atome original mais on le garde à côté de la formule insérée dans une conjonction ou une disjonction, selon la position. La formule obtenue est équivalente à F et est ontologiquement correcte en vue de $\Gamma_1, \mathbb{D}, \Gamma_2$.

Quant aux deux derniers cas, remarquez que la formule $C[f(x_1, \ldots, x_n)/v]$ est impliquée par la formule S (rappelons, $S = \forall v\,(v \approx f(x_1, \ldots, x_n) \equiv C)$ et $v \notin \{x_1, \ldots, x_n\}$). Ainsi, la formule instanciée $C[F|_\pi/v]\sigma$ (où $F|_\pi$ est $f(s_1, \ldots, s_n)$) est localement valide dans l'occurrence $F|_{\widehat{\pi}}$. Alors nous pouvons la mettre à côté de l'atome original $F|_{\widehat{\pi}}$ dans une conjonction ou une implication d'après le corollaire 2.2.9. La formule obtenue est équivalente à F et est ontologiquement correcte en vue de $\Gamma_1, \mathbb{D}, \Gamma_2$.

Nous appelons cette transformation conservatrice puisque l'occurrence initiale reste dans la formule et peut participer dans la recherche de preuve. Notez qu'on ne peut pas utiliser l'application conservatrice dans des positions neutres. Il faut alors convertir d'abord les équivalences ($\equiv$) en implications doubles ($\sim$).

Stratégies de l'application de définitions. Il est peu probable qu'on puisse donner une solution optimale pour le problème de sélection de bonnes occurrences pour l'application de

définitions. Toute méthode raisonnable sera efficace pour certains textes et échouera pour les autres. En absence d'une stratégie qui pourrait s'adapter à un texte donné, l'auteur s'est penché à adapter ses textes aux stratégies présentes. En conséquent, ces stratégies doivent être suffisamment simples et intuitives pour faciliter une telle adaptation.

Au cours du développement du système SAD, nous avons considéré plusieurs stratégies de l'application de définitions, chacune avec ses avantages et inconvénients. Notez bien que ces «avantages» et «inconvénients» proviennent de nos expériences avec une certaine implantation et certains textes formels.

- «Peler». Cette stratégie agit suivant la hiérarchie de définitions. Disons qu'une définition dépend d'un symbole de signature si ce symbole intervient dans la partie droite de la définition. À toute itération, le raisonneur applique des définitions aux symboles les plus «jeunes», dont aucune définition applicable à proximité du but ne dépend. Cela découvre une autre couche de symboles, qui deviennent ainsi les plus jeunes, et auxquels on applique des définitions à l'itération suivante.

  Cette stratégie admet l'application destructive et n'exige pas de retour-arrière. Elle est claire et naturelle. La difficulté avec cette méthode est la suivante. Quand nous nous sommes enfoncés dans une démonstration longue en ForTheL, il est très probable que notre contexte contient des symboles de toutes les couches de la hiérarchie de définitions. On est alors obligé d'attendre pendant plusieurs itérations du raisonneur avant qu'une seule définition nécessaire, disons, celle de sous-ensemble (un symbole des plus «vieux»), ne soit appliquée.
- «Peser». Cette stratégie choisit des occurrences «prometteuses» à l'aide d'évaluations heuristiques qui prennent en compte le niveau du symbole dans la hiérarchie de définitions, la présence des occurrences complémentaires (pour la résolution), la proximité du but et ainsi de suite. L'application est nécessairement conservatrice pour préserver des occurrences qui ont été faussement choisies. L'inconvénient de cette méthode est que son comportement est difficile à prévoir sans connaissance détaillée de l'implantation. Alors, il n'est pas clair comment adapter un texte pour cette stratégie quand elle s'égare.
- «Battue». Cette stratégie prescrit au raisonneur d'appliquer les définitions partout où c'est possible dans la limite de la section haut-niveau courante. L'application doit être conservatrice. Les formules insérées apportent de nouveaux symboles définis auxquels le raisonneur appliquera des définitions à l'itération suivante. L'avantage de cette méthode est qu'elle est simple et prévisible. D'autre part, elle peut amener vite aux formules assez volumineuses et donc alourdir la tâche du démonstrateur au lieu de l'alléger.

Dans la version actuelle du système, c'est la troisième stratégie qui est implantée. Sa performance est acceptable pour les textes formels à notre disposition. Toutefois, notre choix de stratégie peut changer un jour, incité (comme peut-être tout dans notre travail) par la nécessité et la curiosité.

Chapitre 4

Démonstration orientée but

4.1 Introduction

Dans ce chapitre nous étudions l'approche de la recherche automatique de preuve associée au projet «Evidence Algorithm» [14, 15, 16, 47]. Dans les travaux précédents de notre équipe sur ce sujet, la description de la procédure déductive de SAD a été basée sur les calculs séquentiels qui avaient les traits caractéristiques suivants :

- la déduction part du séquent en question et régresse vers les axiomes ;
- la déduction est orientée but, c'est-à-dire que le choix du pas d'inférence suivant est borné par la forme de la formule-conclusion du séquent ;
- les règles d'inférence ne changent pas les formules-hypothèses initiales ; mais elles peuvent ajouter de nouvelles hypothèses atomiques ;
- les substitutions des variables libres sont définies par les obligations d'unification que l'on accumule pendant la déduction ;
- au lieu de skolémiser les formules initiales, on impose la contrainte spéciale d'admissibilité sur la substitution sommaire des variables libres.

Tout cela nous permet d'expliquer notre approche d'une façon naturelle dans le cadre des calculs de tableaux [11]. Ces calculs ont comme objet l'arbre (tableau) de formules. Un pas d'inférence séparé ajoute de nouvelles feuilles au bout d'une des branches dans cet arbre. L'arbre de départ consiste en une seule branche qui contient l'ensemble de formules dont on cherche à démontrer l'incohérence. La démonstration est accomplie si chacune des branches de l'arbre obtenu contient deux littéraux complémentaires. En somme, les calculs séquentiels dans le style de «Evidence Algorithm» peuvent être facilement reformulés en tant que calculs de tableaux :

- le séquent initial $F_1, \ldots, F_n \to G$ devient la branche initiale $F_1, \ldots, F_n, \neg G$;
- un règle d'inférence séquentielle qui génère un ou plusieurs séquents avec les conclusions $G_1, \ldots, G_s$ devient un règle qui ajoute les feuilles $G_1^\neg, \ldots, G_s^\neg$ à une des branches dans le tableau (ici, $G_i^\neg$ est la formule complémentaire à G_i) ;
- les hypothèses atomiques qui ont été accumulées dans un séquent se trouvent dans la branche au-dessus du noeud qui correspond à ce séquent dans le tableau.

4.2 Préliminaires

Nous travaillons toujours en logique classique du premier ordre avec les opérations d'implication ($\supset$), de disjonction ($\vee$), de conjonction ($\wedge$), de négation ($\neg$) et les quantifications universelle ($\forall$) et existentielle ($\exists$). Nous n'utilisons pas le connecteur d'équivalence dans ce chapitre.

Dans la section 4.3 la logique est par défaut sans égalité. L'égalité est traitée dans la section 4.4.

Les notions de substitution, de terme-substitut, de position dans une formule, ainsi que la notation et les opérations correspondantes sont utilisées selon les définitions de la section 2.2.1. Notez que toutes les positions sont, en absence d'équivalence, soit positives soit négatives.

Une *contrainte* est une conjonction (peut-être vide) de *contraintes atomiques* $s = t$ ou $s \succ t$ ou $s \succeq t$. Les lettres γ et δ dénotent les contraintes. Le symbole $\top$ dénote la contrainte vide. Une contrainte composée $(a = b \wedge b \succ c)$ peut être écrite sous forme abrégée $(a = b \succ c)$. Une contrainte sur les cortèges de termes $(\vec{s} = \vec{t})$ désigne la conjonction $(s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$.

Une substitution σ *résout* une contrainte atomique $s = t$ si les termes $s\sigma$ et $t\sigma$ sont graphiquement identiques. Elle est une solution d'une contrainte atomique $s \succ t$ (respectivement, $s \succeq t$) si $s\sigma > t\sigma$ (respectivement, $s\sigma \geqslant t\sigma$) dans quelque ordonnancement de réduction $>$ qui est total sur les termes clos. On dit que σ est une solution d'une contrainte générale γ si elle résout toutes les contraintes atomiques dans γ. La contrainte est dite *satisfaisable* si elle possède une solution.

Nous présentons plusieurs calculs de tableaux dans ce chapitre. Il est donc désirable d'adopter une façon unifiée de présentation pour eux. Un *tableau* est un arbre fini $\mathbb{T}$ dont les noeuds (sauf la racine) sont des paires $F \cdot \gamma$, où F est une formule et γ est une contrainte. La racine d'un tableau contient l'ensemble des formules initiales dont on cherche à démontrer l'incohérence.

Une branche qui contient la formule atomique $\bot$ (la contradiction) est *close*. Un tableau est *clos* si toutes ses branches sont closes et l'ensemble commun de ses contraintes est satisfaisable. Certaines calculs dans ce chapitre exigent d'ailleurs que la substitution résolvante doit être «admissible» (pour une certaine notion d'admissibilité).

L'inférence commence avec un seul noeud de racine (aussi dit *le noeud initial*). Un pas d'inférence étend une des branches en ajoutant de nouvelles feuilles ou de sous-arbres en dessous d'elle. Les règles d'inférence sont décrites comme suit :

$$\frac{\Gamma \parallel \Delta}{\mathbb{T}_1 \quad \cdots \quad \mathbb{T}_n}$$

où Γ est l'ensemble des formules initiales (le noeud de racine), Δ la branche que l'on étend (avec les contraintes omises) et $\mathbb{T}_1, \ldots, \mathbb{T}_n$ des sous-arbres ajoutés. L'ordre des formules dans Γ et Δ est significatif. Si la contrainte auprès d'une formule dans $\mathbb{T}_i$ n'est pas spécifiée, cette contrainte est considérée comme vide, donc vraie.

Comme exemple, considérons les tableaux classiques du premier ordre **Tb** (figure 4.1). Les symboles avec barre $\bar{u}, \bar{\mathbf{g}}$ sont «frais», c'est-à-dire nouveaux par rapport au tableau courant. Ainsi, les γ-règles remplacent une variable bornée avec une variable fraîche et les δ-règles introduisent un nouveaux symbole de Skolem.

Exportation :

$$\frac{\Gamma, F, \Lambda \parallel \Delta}{F}$$

α-règles :

$$\frac{\Gamma \parallel \Delta, F \wedge G, \Lambda}{\begin{array}{c} F \\ G \end{array}} \qquad \frac{\Gamma \parallel \Delta, \neg(F \vee G), \Lambda}{\begin{array}{c} \neg F \\ \neg G \end{array}} \qquad \frac{\Gamma \parallel \Delta, \neg(F \supset G), \Lambda}{\begin{array}{c} F \\ \neg G \end{array}}$$

Double négation :

$$\frac{\Gamma \parallel \Delta, \neg\neg F, \Lambda}{F}$$

β-règles :

$$\frac{\Gamma \parallel \Delta, \neg(F \wedge G), \Lambda}{\neg F \qquad \neg G} \qquad \frac{\Gamma \parallel \Delta, F \vee G, \Lambda}{F \qquad G} \qquad \frac{\Gamma \parallel \Delta, F \supset G, \Lambda}{\neg F \qquad G}$$

γ-règles :

$$\frac{\Gamma \parallel \Delta, \forall\, v\, F, \Lambda}{F[\bar{u}/v]} \qquad \frac{\Gamma \parallel \Delta, \neg\exists\, v\, F, \Lambda}{\neg F[\bar{u}/v]}$$

δ-règles :

$$\frac{\Gamma \parallel \Delta, \exists\, v\, F, \Lambda}{F[\bar{\mathbf{g}}(\mathcal{FV}(F))/v]} \qquad \frac{\Gamma \parallel \Delta, \neg\,\forall\, v\, F, \Lambda}{\neg F[\bar{\mathbf{g}}(\mathcal{FV}(F))/v]}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta, \neg P(s_1, \ldots, s_n), \Lambda, P(t_1, \ldots, t_n)}{\bot \cdot (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)} \qquad \frac{\Gamma \parallel \Delta, P(s_1, \ldots, s_n), \Lambda, \neg P(t_1, \ldots, t_n)}{\bot \cdot (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)}$$

FIG. 4.1: Tableaux classiques **Tb**

Le calcul **Tb** est cohérent et complet en logique classique du premier ordre [11] :

Théorème 4.2.1. *L'ensemble fini de formules closes est contradictoire si, et seulement si, l'on peut en dériver un tableau clos dans* **Tb**.

4.3 Calcul de tableaux orienté but

4.3.1 Substitutions admissibles

La notion de substitutions admissibles pour les calculs séquentiels à été proposée par A. Lyaletski [43]. Elle permet d'assurer la cohérence d'un calcul basé sur les meta-variables et l'unification sans recours à la skolémisation, c.-à-d. avec préservation de la signature initiale.

Un ensemble de formules clos Γ est dite être libre de collisions (*sans-collision*) si aucune variable n'est pas bornée par plusieurs quantificateurs dans Γ (même dans les formules différentes). Pour un tel Γ dénotons par $\mathcal{V}_\Gamma$ l'ensemble des *variables indexées* $\{{}^k v | v$ intervient dans $\Gamma, k \in \mathbb{N}\}$. L'expression ${}^k F$ désignera la formule F où toute variable v, libre ou bornée, est remplacée par la variable indexée ${}^k v$. Notons que v, ${}^k v$ et ${}^{k+1} v$ sont trois variables différentes. Un Γ-*terme* est un terme dont les symboles fonctionnels proviennent de Γ et les variables, de $\mathcal{V}_\Gamma$.

Une variable ${}^k v \in \mathcal{V}_\Gamma$ est dite d'être *inconnue* dans Γ (${}^k v \in \mathcal{V}_\Gamma^+$) si Γ contient une formule de la forme $P[(\forall v F)^+]_\tau$ ou $P[(\exists v F)^-]_\tau$. Par ailleurs, ${}^k v \in \mathcal{V}_\Gamma$ est dite d'être *fixe* dans Γ si quelque formule dans Γ est de la forme $P[(\forall v F)^-]_\tau$ ou $P[(\exists v F)^+]_\tau$. Bien entendu, toute variable dans $\mathcal{V}_\Gamma$ est soit inconnue soit fixe : $\mathcal{V}_\Gamma^+ \cup \mathcal{V}_\Gamma^- = \mathcal{V}_\Gamma$, $\mathcal{V}_\Gamma^+ \cap \mathcal{V}_\Gamma^- = \varnothing$. Dans ce qui suit nous écrivons des variables fixes en caractères gras : ${}^k \mathbf{v}$.

Un ensemble sans-collision Γ génère une relation $\lhd_\Gamma$: $\mathcal{V}_\Gamma^+ \times \mathcal{V}_\Gamma^-$ comme suit : ${}^k u \lhd_\Gamma {}^m \mathbf{w}$ si, et seulement si, $k = m$ et le quantificateur sur w se trouve sous le quantificateur sur u dans Γ, c.-à-d. Γ contient une formule de la forme $(\dots \mathcal{Q}_1 u (\dots \mathcal{Q}_2 w (\dots) \dots) \dots)$. Notons que ${}^k u \lhd_\Gamma {}^k \mathbf{w}$ implique ${}^m u \lhd_\Gamma {}^m \mathbf{w}$.

Pour une substitution donnée σ, nous définissons une relation $\lll_\Gamma^\sigma$: $\mathcal{V}_\Gamma^- \times \mathcal{V}_\Gamma^+$ comme suit : ${}^m \mathbf{w} \lll_\Gamma^\sigma {}^k u$ si, et seulement si, ${}^m \mathbf{w}$ se trouve dans ${}^k u\sigma$.

Une substitution finie σ est *admissible* dans Γ si les conditions suivantes sont satisfaites :

1. pour toute variable fixe ${}^k \mathbf{w} \in \mathcal{V}_\Gamma^-$, ${}^k \mathbf{w}\sigma = {}^m \mathbf{w}$ (pour m quelconque) ;
2. pour toutes ${}^k \mathbf{w}, {}^m \mathbf{w} \in \mathcal{V}_\Gamma^-$, ${}^k u \in \mathcal{V}_\Gamma^+$, si ${}^k \mathbf{w}\sigma = {}^m \mathbf{w}\sigma$ et ${}^k u \lhd_\Gamma {}^k \mathbf{w}$, alors ${}^k u\sigma = {}^m u\sigma$;
3. la fermeture transitive de la composition $(\lhd_\Gamma \circ \lll_\Gamma^\sigma) : \mathcal{V}_\Gamma^- \times \mathcal{V}_\Gamma^-$ est irréflexive.

L'homologue suivant de la théorème d'Herbrand est démontré pour les substitution admissibles [44] :

Théorème 4.3.1. *Soit Γ un ensemble sans-collision de formules closes. Soit Γ' l'ensemble des formules de Γ avec les quantificateurs effacés (donc, toutes les variables dans Γ' sont libres). Alors Γ est satisfaisable si, et seulement si, pour tout nombre naturel n et toute substitution finie et admissible dans Γ, l'ensemble $\{{}^1\Gamma', \dots, {}^n\Gamma'\}\sigma$ est satisfaisable.*

Nous allons redémontrer ce théorème sous un autre aspect, par la connexion entre l'admissibilité et la skolémisation.

La *substitution skolémisante* θ_Γ est une substitution qui remplace chaque variable fixe ${}^k \mathbf{v} \in \mathcal{V}_\Gamma^-$ avec un symbole fonctionnel de Skolem $\mathbf{v}({}^k u_1, \dots, {}^k u_n)$ où ${}^k u_1, \dots, {}^k u_n$ est la suite maximale de variables inconnues telles que ${}^k u_i \lhd_\Gamma {}^k \mathbf{v}$ et le quantificateur sur u_{i+1} se trouve sous le quantificateur sur u_i dans Γ. Notons que θ_Γ est une substitution infinie.

Théorème 4.3.2. *Soit Γ un ensemble sans-collision de formules closes. Soit σ une substitution admissible dans Γ. Il y a une substitution π telle que pour tous Γ-termes s, t, $s\sigma = t\sigma$ implique $s\theta_\Gamma\pi = t\theta_\Gamma\pi$.*

Démonstration. Nous pouvons admettre sans perte de généralité que toute variable remplacé par σ n'intervient pas dans les termes-substituts de σ (une telle variable peut être simultanément renommée dans tous les substituts de σ).

Soit ϕ une substitution $(\theta_\Gamma)\sigma$, c.-à-d. une substitution qui remplace toute variable fixe ${}^i \mathbf{v}$ par le terme ${}^i \mathbf{v}\theta_\Gamma\sigma$. Notons que ϕ n'est pas équivalente à la composition de $\sigma \circ \theta_\Gamma$; en particulier

ϕ ne remplace pas des variables inconnues. Nous obtenons la substitution ϕ en appliquant σ aux termes-substituts de θ_Γ.

Mettons $\phi^1 = \phi$, $\phi^2 = \phi \circ \phi$, et $\phi^{i+1} = \phi \circ \phi^i$. Nous allons démontrer qu'il y existe un tel n que $\phi^{n+1} = \phi^n$. Soit W_i l'ensemble des variables fixes intervenant dans les termes-substituts de ϕ^i. Remarquez que W_i est fini et $W_{i+1} \subseteq W_i$ pour tous $i > 0$. Comme la fermeture transitive de $(\lhd_\Gamma \circ \lll^\sigma_\Gamma)$ est irréflexive, et donc est un ordonnancement partiel strict, on peut choisir dans W_i un élément maximal par rapport à cet ordonnancement (à condition que W_i ne soit pas vide). Désignons un tel élément par ${}^k\mathbf{w}$. S'il y avait une variable ${}^l\mathbf{v} \in W_i$ telle que ${}^l\mathbf{v}\phi$ contenait ${}^k\mathbf{w}$, on aurait ${}^k\mathbf{w}\,(\lhd_\Gamma \circ \lll^\sigma_\Gamma)\,{}^l\mathbf{v}$ ce qui contredit la supériorité de ${}^k\mathbf{w}$. En effet, si ${}^l\mathbf{v}\phi$ contient ${}^k\mathbf{w}$, alors il y a une variable inconnue ${}^lu \lhd_\Gamma {}^l\mathbf{v}$ telle que ${}^lu\sigma$ contient ${}^k\mathbf{w}$, d'où ${}^k\mathbf{w} \lll^\sigma_\Gamma {}^lu$.

Ainsi, si W_i n'est pas vide, alors W_{i+1} est un propre sous-ensemble de W_i. C'est pourquoi il y existe un tel n que $W_n = \varnothing$. Il est bien évident que $\phi \circ \phi^n = \phi^n$.

Soit $\psi = \phi^n$ et $\pi = \psi \circ \sigma$. Pour toute variable fixe ${}^k\mathbf{v}$, ${}^k\mathbf{v}\theta_\Gamma\sigma = {}^k\mathbf{v}\sigma\theta_\Gamma\sigma$ par définition de l'admissibilité. Alors nous obtenons ${}^k\mathbf{v}\theta_\Gamma\pi = {}^k\mathbf{v}\theta_\Gamma\sigma\psi = {}^k\mathbf{v}\sigma\theta_\Gamma\sigma\psi = {}^k\mathbf{v}\sigma\phi\psi = {}^k\mathbf{v}\sigma\psi$. Pour toute variable inconnue ${}^ku$, nous avons ${}^ku\theta_\Gamma\pi = {}^ku\sigma\psi$ aussi. Ainsi pour tout Γ-terme r, $r\theta_\Gamma\pi = r\sigma\psi$.

Ainsi pour tous Γ-termes s et t, $s\sigma = t\sigma \;\Rightarrow\; s\sigma\psi = t\sigma\psi \;\Rightarrow\; s\theta_\Gamma\pi = t\theta_\Gamma\pi$. □

Théorème 4.3.3. *Soit Γ un ensemble sans-collision de formules closes et π, une substitution finie. Il y a une substitution σ admissible dans Γ tel que pour tous Γ-termes s et t, $s\theta_\Gamma\pi = t\theta_\Gamma\pi$ implique $s\sigma = t\sigma$.*

Démonstration. Nous pouvons admettre que π ne remplace pas les variables fixes et ne les introduit non plus. Nous pouvons aussi admettre que toute variable inconnue remplacée par π n'intervient pas dans les termes-substituts de π.

Soit ψ la substitution $(\theta_\Gamma)\pi$, c.-à-d. la substitution qui remplace toute variable fixe ${}^i\mathbf{v}$ par le terme ${}^i\mathbf{v}\theta_\Gamma\pi$. La substitution ψ n'est pas la composition $\pi \circ \theta_\Gamma$, elle est obtenu par l'application de π aux termes-substituts de θ_Γ.

Nous introduisons une relation d'équivalence $\bowtie$ sur les variables fixes de $\mathcal{V}_\Gamma$ comme suit : ${}^k\mathbf{w} \bowtie {}^l\mathbf{v}$ si, et seulement si, ${}^k\mathbf{w}\psi = {}^l\mathbf{v}\psi$. Autrement dit, ${}^k\mathbf{w} \bowtie {}^l\mathbf{v}$ si w et v sont la même variable dans Γ et pour toutes ${}^ku \lhd_\Gamma {}^k\mathbf{w}$, ${}^ku\pi = {}^lu\pi$. Comme π est finie, l'ensemble quotient $\mathcal{V}^-_\Gamma / \bowtie$ contient un nombre fini de membres ayant plus qu'un élément.

La substitution σ est définie comme suit. Pour toute variable fixe ${}^l\mathbf{v}$, ${}^l\mathbf{v}\sigma = {}^k\mathbf{v}$, où k est le moindre indice dans le classe d'équivalence de ${}^l\mathbf{v}$. Pour toute variable inconnue ${}^lu$, ${}^lu\sigma$ est le terme ${}^lu\pi$ avec certains sous-termes remplacés par des variables fixes selon le règle suivant : tout sous-terme maximal r de ${}^lu\pi$ tel que $r = {}^m\mathbf{v}\psi$ (pour ${}^m\mathbf{v}$ quelconque) est remplacé dans ${}^lu\sigma$ par la variable fixe ${}^k\mathbf{v}$ où k est le moindre indice dans le classe d'équivalence de ${}^m\mathbf{v}$.

La substitution σ est admissible dans Γ. Deux premières conditions dans la définition de l'admissibilité sont évidemment satisfaites. Pour démontrer la troisième condition notons que ${}^ku\,(\lll^\sigma_\Gamma \circ \lhd_\Gamma)\,{}^lx$ implique que ${}^ku\pi$ est le propre sous-terme de ${}^lx\pi$. La fermeture transitive de $(\lll^\sigma_\Gamma \circ \lhd_\Gamma)$ est ainsi irréflexive, d'où la fermeture transitive de $(\lhd_\Gamma \circ \lll^\sigma_\Gamma)$ est irréflexive, elle aussi.

Soit k le moindre indice dans le classe d'équivalence d'une variable fixe ${}^l\mathbf{v}$. Nous obtenons ${}^l\mathbf{v}\theta_\Gamma\pi = \mathbf{v}({}^lu\pi, \dots) = \mathbf{v}({}^ku\pi, \dots) = {}^k\mathbf{v}\theta_\Gamma\pi = {}^k\mathbf{v}\psi = {}^l\mathbf{v}\sigma\psi$. Pour toute variable inconnue ${}^lu$, nous avons ${}^lu\theta_\Gamma\pi = {}^lu\pi = {}^lu\sigma\psi$ par construction de σ. Ainsi pour tout Γ-terme r, $r\theta_\Gamma\pi = r\sigma\psi$.

Considérons des Γ-termes s, t tels que $s\theta_\Gamma\pi = t\theta_\Gamma\pi$ et, donc, $s\sigma\psi = t\sigma\psi$. Supposons que $s\sigma \neq t\sigma$. Comme ψ ne remplace que des variables fixes, nous pouvons admettre qu'il y a un sous-terme r dans $s\sigma$ et une variable fixe ${}^k\mathbf{v}$ dans $t\sigma$ tels que $r\psi = {}^k\mathbf{v}\psi$ mais $r \neq {}^k\mathbf{v}$.

Par construction de σ, la variable ${}^k\mathbf{v}$ possède le moindre indice dans son classe d'équivalence. Si r est une variable fixe, alors elle a le moindre indice dans son classe d'équivalence, elle aussi. Comme $r \bowtie {}^k\mathbf{v}$, on obtient $r = {}^k\mathbf{v}$. Sinon, r est un terme avec un symbole de Skolem $\mathbf{v}$ en haut. Comme le terme s ne contient pas de symboles de Skolem, r a été introduit à $s\sigma$ dans un terme-substitut de σ. Ainsi, $r\psi$ intervient dans un substitut de π. Par ailleurs, $r\psi = {}^k\mathbf{v}\psi$ et, donc, r ne peut pas intervenir dans un terme-substitut de σ, par construction de σ. Nous obtenons la contradiction. Ainsi, $s\sigma = t\sigma$. □

4.3.2 Calcul de tableaux orienté but

Le calcul de base de notre démonstrateur automatique est un raffinement de **Tb** où le choix de pas d'extension d'une branche est borné par la forme de la formule dans la feuille de branche. Cette formule est considérée comme un *but*; si c'est une formule non-littérale, on applique $\alpha\beta\gamma\delta$-règles pour la décomposer; si c'est un littéral, alors on choisit un nouveau but.

Nous dirigeons l'inférence en marquant une sous-formule atomique dans chacun des noeuds générés. Pour cette raison, nous utiliserons les arbres de triplets $F \diamond \tau \cdot \gamma$, où F est une formule, γ est une contrainte, et τ est une position d'une sous-formule atomique dans F.

Au début d'une inférence nous prenons une des formules initiales, marquons un atome dans elle, et la mettons en dessous de la racine.

Nous prenons en compte ce marquage au cours de décomposition de but. Dans les α-règles, nous n'ajoutons qu'une seule de deux sous-formules : celle qui contient l'atome choisi. Dans les β-règles, une des branches générées hérite l'atome choisi du but courant et, dans l'autre, nous choisissons d'une façon aléatoire un nouveau atome marqué.

Dès que le but est un littéral L (avec l'atome A marqué) notre choix de nouveau but peut être raffiné aussi : parmi les formules initiales, nous choisissons une qui contient un atome unifiable avec A mais avec le signe d'occurrence opposé.

Comme nous travaillons avec des substitutions admissibles et l'admissibilité est définie par rapport à l'ensemble des formules initiales, nous pouvons ignorer les quantificateurs pendant l'inférence. Chaque fois que nous transmettons une formule initiale dans le tableau, nous indexons toutes les variables dans cette formule avec un indice frais.

Les règles d'inférence de calcul **GDT** sont données à la figure 4.2. Notons que l'ordre de formules dans Γ est significatif : le règle de début prend la dernière formule de Γ en tant que le premier but.

Un **GDT**-tableau $\mathbb{T}$ avec l'ensemble Γ dans la racine est *clos* si, et seulement si, toutes les branches dans $\mathbb{T}$ sont closes et l'ensemble des contraintes dans $\mathbb{T}$ peut être résolu par une substitution qui est admissible dans Γ.

Le calcul **GDT** est cohérent et complet en logique classique du premier ordre :

Théorème 4.3.4. *Soit $\Gamma = \Delta \cup \{G\}$ un ensemble sans-collision de formules closes. Supposons que Δ est satisfaisable. Alors Γ est contradictoire si, et seulement si, il y existe un* **GDT***-tableau clos dont (Δ, G) est la racine.*

Ce théorème est impliqué par la cohérence et complétude du calcul **GD** [71] dont **GDT** est simple reformulation en termes de tableaux. Dans la section 4.3.4 nous donnerons une démonstration indirecte du théorème 4.3.4.

Considérons un exemple de l'inférence dans **GDT**. Nous réfutons l'ensemble Γ suivant :

$$\forall\, pqx\,.\,(p \subseteq q \wedge x \in p) \supset x \in q$$
$$\forall\, ry\,.\,E(r) \supset \neg(y \in r)$$
$$\forall\, s\,.\,(\forall\, z\,.\,\neg(z \in s)) \supset E(s)$$
$$\exists\, e\,.\,E(e)$$
$$\neg\forall\, a\,.\,(\forall\, b\,.\,a \subseteq b) \supset E(a)$$

Notons que Γ est sans-collision, ainsi que les conditions des théorèmes 4.3.1 et 4.3.4 sont satisfaites et l'application des $\gamma\delta$-règles n'amène pas à une inférence incohérente.

La figure 4.3 présente une **GDT**-réfutation $\mathbb{T}$ de l'ensemble Γ. Les atomes sélectionnés sont encadres. Notez que toutes les branches dans $\mathbb{T}$ sont closes bien que nous n'avons pas une fois appliqué les règles de terminaison : toutes les contradictions $\bot$ ont été introduites par les pas de «sélection de but». Cependant, ce n'est pas le cas pour toutes **GDT**-inférences. Voici l'ensemble commun des contraintes dans $\mathbb{T}$:

$$^0\mathbf{a} = {}^1s \qquad {}^1\mathbf{z} = {}^2x \qquad {}^1s = {}^2p \qquad {}^2p = {}^3\mathbf{a}$$
$$^2q = {}^3b \qquad {}^2x = {}^4y \qquad {}^2q = {}^4r \qquad {}^4r = {}^5\mathbf{e}$$

Début :

$$\frac{\Gamma,\ F[P(\vec{s})]_\tau \parallel}{{}^0F[P(\vec{s})]_\tau \diamond \tau}$$

Double négation :

$$\frac{\Gamma \parallel \Delta,\ \neg\neg F \diamond 0.\tau}{F \diamond \tau}$$

α-règles :

$$\frac{\Gamma \parallel \Delta,\ F \wedge G \diamond 0.\tau}{F \diamond \tau} \qquad \frac{\Gamma \parallel \Delta,\ F \wedge G \diamond 1.\tau}{G \diamond \tau}$$

$$\frac{\Gamma \parallel \Delta,\ \neg(F \vee G) \diamond 0.0.\tau}{\neg F \diamond 0.\tau} \qquad \frac{\Gamma \parallel \Delta,\ \neg(F \vee G) \diamond 0.1.\tau}{\neg G \diamond 0.\tau}$$

$$\frac{\Gamma \parallel \Delta,\ \neg(F \supset G) \diamond 0.0.\tau}{F \diamond \tau} \qquad \frac{\Gamma \parallel \Delta,\ \neg(F \supset G) \diamond 0.1.\tau}{\neg G \diamond 0.\tau}$$

β-règles :

$$\frac{\Gamma \parallel \Delta,\ \neg(F \wedge G[Q(\vec{s})]_\mu) \diamond 0.0.\tau}{\neg F \diamond 0.\tau \qquad \neg G[Q(\vec{s})]_\mu \diamond 0.\mu} \qquad \frac{\Gamma \parallel \Delta,\ \neg(F[Q(\vec{s})]_\mu \wedge G) \diamond 0.1.\tau}{\neg F[Q(\vec{s})]_\mu \diamond 0.\mu \qquad \neg G \diamond 0.\tau}$$

$$\frac{\Gamma \parallel \Delta,\ F \vee G[Q(\vec{s})]_\mu \diamond 0.\tau}{F \diamond \tau \qquad G[Q(\vec{s})]_\mu \diamond \mu} \qquad \frac{\Gamma \parallel \Delta,\ F[Q(\vec{s})]_\mu \vee G \diamond 1.\tau}{F[Q(\vec{s})]_\mu \diamond \mu \qquad G \diamond \tau}$$

$$\frac{\Gamma \parallel \Delta,\ F \supset G[Q(\vec{s})]_\mu \diamond 0.\tau}{\neg F \diamond 0.\tau \qquad G[Q(\vec{s})]_\mu \diamond \mu} \qquad \frac{\Gamma \parallel \Delta,\ F[Q(\vec{s})]_\mu \supset G \diamond 1.\tau}{\neg F[Q(\vec{s})]_\mu \diamond 0.\mu \qquad G \diamond \tau}$$

$\gamma\delta$-règles :

$$\frac{\Gamma \parallel \Delta,\ \forall^k v\ F \diamond 0.\tau}{F \diamond \tau} \qquad \frac{\Gamma \parallel \Delta,\ \neg\exists^k v\ F \diamond 0.0.\tau}{\neg F \diamond 0.\tau}$$

$$\frac{\Gamma \parallel \Delta,\ \exists^k v\ F \diamond 0.\tau}{F \diamond \tau} \qquad \frac{\Gamma \parallel \Delta,\ \neg\forall^k v\ F \diamond 0.0.\tau}{\neg F \diamond 0.\tau}$$

Sélection de but (m est un indice frais) :

$$\frac{\Gamma,\ F[P(\vec{s})^-]_\tau,\ \Lambda \parallel \Delta,\ P(\vec{r}) \diamond \varepsilon}{{}^mF[\neg\bot]_\tau \diamond \tau.0 \cdot ({}^m\vec{s} = \vec{r})} \qquad \frac{\Gamma,\ F[P(\vec{s})^+]_\tau,\ \Lambda \parallel \Delta,\ \neg P(\vec{r}) \diamond 0}{{}^mF[\bot]_\tau \diamond \tau \cdot ({}^m\vec{s} = \vec{r})}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta,\ \neg P(\vec{s}) \diamond 0,\ \Lambda,\ P(\vec{r}) \diamond \varepsilon}{\bot \diamond \varepsilon \cdot (\vec{s} = \vec{r})} \qquad \frac{\Gamma \parallel \Delta,\ P(\vec{s}) \diamond \varepsilon,\ \Lambda,\ \neg P(\vec{r}) \diamond 0}{\bot \diamond \varepsilon \cdot (\vec{s} = \vec{r})}$$

FIG. 4.2: Calcul de tableaux orienté but **GDT**

$$
\mathbb{T} = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma}{\neg\forall^0\mathbf{a}\,.\,(\forall^0 b\,.\,{}^0\mathbf{a}\subseteq{}^0 b)\supset\boxed{E({}^0\mathbf{a})}\ \diamond\ 0.0.1}\,(\text{début})}{\neg\left((\forall^0 b\,.\,{}^0\mathbf{a}\subseteq{}^0 b)\supset\boxed{E({}^0\mathbf{a})}\right)\ \diamond\ 0.1}\,\delta}{\neg\boxed{E({}^0\mathbf{a})}\ \diamond\ 0}\,\alpha}{\forall^1 s\,.\,(\forall^1\mathbf{z}\,.\,\neg({}^1\mathbf{z}\in{}^1 s))\supset\boxed{\bot}\ \diamond\ 0.1\ \cdot\ ({}^0\mathbf{a}={}^1 s)}\,(\text{sdb})}{(\forall^1\mathbf{z}\,.\,\neg({}^1\mathbf{z}\in{}^1 s))\supset\boxed{\bot}\ \diamond\ 1}\,\gamma}{\neg\forall^1\mathbf{z}\,.\,\neg\boxed{{}^1\mathbf{z}\in{}^1 s}\ \diamond\ 0.0.0}\,\beta \qquad \boxed{\bot}\ \diamond\ \varepsilon}{\neg\neg\boxed{{}^1\mathbf{z}\in{}^1 s}\ \diamond\ 0.0}\,\delta}{\boxed{{}^1\mathbf{z}\in{}^1 s}\ \diamond\ \varepsilon}\,(\text{dneg})}{\mathbb{T}_1}\,(\text{sdb})
$$

$$
\mathbb{T}_1 = \dfrac{\dfrac{\forall^2 p^2 q^2 x\,.\,({}^2 p\subseteq{}^2 q\wedge\neg\boxed{\bot})\supset{}^2 x\in{}^2 q\ \diamond\ 0.0.0.0.1.0\ \cdot\ ({}^1\mathbf{z}={}^2 x\wedge{}^1 s={}^2 p)}{(^2 p\subseteq{}^2 q\wedge\neg\boxed{\bot})\supset{}^2 x\in{}^2 q\ \diamond\ 0.1.0}\,\gamma}{\dfrac{\dfrac{\neg({}^2 p\subseteq{}^2 q\wedge\neg\boxed{\bot})\ \diamond\ 0.1.0}{\neg\boxed{{}^2 p\subseteq{}^2 q}\ \diamond\ 0}\,\beta}{\mathbb{T}_2}\,(\text{sdb}) \qquad \dfrac{\dfrac{\neg\neg\boxed{\bot}\ \diamond\ 0.0}{\boxed{\bot}\ \diamond\ \varepsilon}\,(\text{dneg})}{}}\,\beta \qquad \dfrac{\boxed{{}^2 x\in{}^2 q}\ \diamond\ \varepsilon}{\mathbb{T}_3}\,(\text{sdb})
$$

$$
\mathbb{T}_2 = \dfrac{\dfrac{\dfrac{\neg\forall^3\mathbf{a}\,.\,(\forall^3 b\,.\,\boxed{\bot})\supset E({}^3\mathbf{a})\ \diamond\ 0.0.0.0\ \cdot\ ({}^2 p={}^3\mathbf{a}\wedge{}^2 q={}^3 b)}{\neg\left((\forall^3 b\,.\,\boxed{\bot})\supset E({}^3\mathbf{a})\right)\ \diamond\ 0.0.0}\,\delta}{\forall^3 b\,.\,\boxed{\bot}\ \diamond\ 0}\,\alpha}{\boxed{\bot}\ \diamond\ \varepsilon}\,\gamma
$$

$$
\mathbb{T}_3 = \dfrac{\dfrac{\forall^4 r^4 y\,.\,E({}^4 r)\supset\neg\neg\boxed{\bot}\ \diamond\ 0.0.1.0.0\ \cdot\ ({}^2 x={}^4 y\wedge{}^2 q={}^4 r)}{E({}^4 r)\supset\neg\neg\boxed{\bot}\ \diamond\ 1.0.0}\,\gamma}{\dfrac{\dfrac{\neg\boxed{E({}^4 r)}\ \diamond\ 0}{\exists^5\mathbf{e}\,.\,\boxed{\bot}\ \diamond\ 0\ \cdot\ ({}^4 r={}^5\mathbf{e})}\,(\text{sdb})}{\boxed{\bot}\ \diamond\ \varepsilon}\,\delta \qquad \dfrac{\neg\neg\boxed{\bot}\ \diamond\ 0.0}{\boxed{\bot}\ \diamond\ \varepsilon}\,(\text{dneg})}{}}\,\beta
$$

FIG. 4.3: **GDT**-tableau clos

qui peut être résolu par $\sigma = [{}^0\mathbf{a}/{}^3\mathbf{a}, {}^0\mathbf{a}/{}^1s, {}^0\mathbf{a}/{}^2p, {}^1\mathbf{z}/{}^2x, {}^1\mathbf{z}/{}^4y, {}^5\mathbf{e}/{}^2q, {}^5\mathbf{e}/{}^3b, {}^5\mathbf{e}/{}^4r]$. Pour vérifier que σ est admissible, considérons les ordonnancements $\lhd_\Gamma$ et $\lll^\sigma_\Gamma$:

$$ {}^0\mathbf{a} \lll^\sigma_\Gamma {}^1s, {}^2p \qquad {}^1\mathbf{z} \lll^\sigma_\Gamma {}^2x, {}^4y \qquad {}^5\mathbf{e} \lll^\sigma_\Gamma {}^2q, {}^3b, {}^4r \qquad {}^is \lhd_\Gamma {}^i\mathbf{z} \qquad \text{(for any } i \in \mathbb{N}) $$

La fermeture transitive de $(\lhd_\Gamma \circ \lll^\sigma_\Gamma)$ ne contient pas de boucles. En somme, l'ensemble Γ est réfuté.

Dans les sections suivantes nous allons étudier les liens entre **GDT** et un calcul de tableaux très bien connu.

4.3.3 Tableaux de connexions

La procédure d'élimination de modèles a été proposée par Loveland en forme d'un calcul résolutionnel avec des clauses spéciales [41]. Plus tard cette procédure a été reconsidéré en tant qu'un calcul de tableaux où la recherche est guidée par les connexions entre les clauses [39]. Dans cette incarnation, la méthode est connue sous le nom de *tableaux de connexions*. Ce calcul est un raffinement puissant des tableaux généraux. D'ailleurs, les stratégies de recherche fortes et les techniques d'implantation efficaces ont été développées pour les tableaux de connexions [40]. Les règles d'inférence du calcul **CT** sont données à la figure 4.4.

Début :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L_k),\ \Lambda \parallel}{{}^0L_1 \quad \cdots \quad {}^0L_k}$$

Expansion (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^m\vec{s} = \vec{r}) \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, \neg P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^m\vec{s} = \vec{r}) \quad \cdots \quad {}^mL_k}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta, \neg P(\vec{s}), \Lambda, P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})} \qquad \frac{\Gamma \parallel \Delta, P(\vec{s}), \Lambda, \neg P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})}$$

FIG. 4.4: Tableaux de connexions **CT**

Théorème 4.3.5 ([40]). *Un ensemble de clauses $\mathcal{S}$ est contradictoire si, et seulement si, il y existe un* **CT***-tableau construit à partir de $\mathcal{S}$.*

L'affirmation de complétude peut être d'ailleurs renforcée comme suit :

Théorème 4.3.6. *Si un ensemble de clauses $\mathcal{S}$ est contradictoire mais tout sous-ensemble propre de $\mathcal{S}$ est satisfaisable, alors pour toute clause $C \in \mathcal{S}$, un* **CT***-réfutation de $\mathcal{S}$ peut débuter par C.*

Démonstration. Comme le calcul **CT** est cohérent, toute clause de $\mathcal{S}$ participe dans toute **CT**-inférence qui réfute $\mathcal{S}$, soit en tant que la clause de début soit dans expansion. Considérons un tableau clos $\mathbb{T}$ où la clause C intervient à la profondeur minimale possible (appelons-la «la profondeur de C» tout simplement) :

$$\frac{\mathcal{S}}{\dfrac{{}^0L_1}{\overrightarrow{\mathbb{T}}_1} \quad \cdots \quad \dfrac{{}^0L_k}{\overrightarrow{\mathbb{T}}_k}}$$

Notez que $\overrightarrow{\mathbb{T}}_1, \ldots, \overrightarrow{\mathbb{T}}_k$ désignent des cortèges d'arbres car un noeud ${}^0L_i$ peut avoir plusieurs descendants.

Si la clause C est la clause de début $L_1 \vee \cdots \vee L_k$, alors nous avons l'inférence cherchée. Sinon, C intervient dans un de $\overrightarrow{\mathbb{T}}_i$. Considérons l'arbre $\mathbb{T}$ en détail :

$$\dfrac{\mathcal{S}}{\dfrac{{}^0L_1}{\overrightarrow{\mathbb{T}}_1} \quad \cdots \quad \dfrac{{}^0L_i}{\dfrac{{}^mM_1}{\overrightarrow{\mathbb{T}}_{i1}} \quad \cdots \quad \bot \cdot \gamma \quad \cdots \quad \dfrac{{}^mM_l}{\overrightarrow{\mathbb{T}}_{il}}} \quad \cdots \quad \dfrac{{}^0L_k}{\overrightarrow{\mathbb{T}}_k}}$$

Nous allons construire un nouveau tableau clos $\mathbb{T}^*$ qui débutera par la clause $M_1 \vee \cdots \vee M_l$. La clause C soit est cette même clause $M_1 \vee \cdots \vee M_l$ soit elle intervient dans un de $\overrightarrow{\mathbb{T}}_{ii'}$. Ainsi, la profondeur de C dans $\mathbb{T}^*$ sera inférieure à celle de C dans $\mathbb{T}$, ce qui va contredire le choix de $\mathbb{T}$.

Commençons par la version préliminaire de $\mathbb{T}^*$ (disons, $\mathbb{T}'$) :

$$\dfrac{\mathcal{S}}{\dfrac{{}^mM_1}{\overrightarrow{\mathbb{T}}_{i1}} \quad \cdots \quad \dfrac{{}^mM_j}{\dfrac{{}^0L_1}{\overrightarrow{\mathbb{T}}_1} \quad \cdots \quad \bot \cdot \gamma \quad \cdots \quad \dfrac{{}^0L_k}{\overrightarrow{\mathbb{T}}_k}} \quad \cdots \quad \dfrac{{}^mM_l}{\overrightarrow{\mathbb{T}}_{il}}}$$

L'arbre $\mathbb{T}'$ n'est pas un **CT**-tableau bien formé, car certaines des branches dans $\overrightarrow{\mathbb{T}}_{i1}, \ldots, \overrightarrow{\mathbb{T}}_{i(j-1)}, \overrightarrow{\mathbb{T}}_{i(j+1)}, \ldots, \overrightarrow{\mathbb{T}}_{il}$ pouvait être terminées dans $\mathbb{T}$ par une connexion avec le littéral ${}^0L_i$. Dans $\mathbb{T}'$ une telle terminaison est impossible. Considérons $\mathbb{T}'$ encore :

$$\dfrac{\mathcal{S}}{\dfrac{\dfrac{{}^mM_1}{\overrightarrow{\mathbb{T}}_{i1}} \quad \cdots \quad \dfrac{{}^mM_j}{\dfrac{{}^0L_1}{\overrightarrow{\mathbb{T}}_1} \quad \cdots \quad \bot \cdot \gamma \quad \cdots \quad \dfrac{{}^0L_k}{\overrightarrow{\mathbb{T}}_k}} \quad \cdots \quad \dfrac{{}^mM_l}{\overrightarrow{\mathbb{T}}_{il}}}{\vdots}}{{}^nL'}}{\bot \cdot ({}^0L_i \leftrightarrow {}^nL')}$$

Ici, l'expression $({}^0L_i \leftrightarrow {}^nL')$ désigne la contrainte correspondante. Nous pouvons «réparer» cette feuille et toutes les autres feuilles de ce genre comme suit :

$$\dfrac{\mathcal{S}}{\dfrac{\dfrac{{}^mM_1}{\overrightarrow{\mathbb{T}}_{i1}} \quad \cdots \quad \dfrac{{}^mM_j}{\dfrac{{}^0L_1}{\overrightarrow{\mathbb{T}}_1} \quad \cdots \quad \bot \cdot \gamma \quad \cdots \quad \dfrac{{}^0L_k}{\overrightarrow{\mathbb{T}}_k}} \quad \cdots \quad \dfrac{{}^mM_l}{\overrightarrow{\mathbb{T}}_{il}}}{\vdots}}{{}^nL'}}{\dfrac{{}^0L_1}{\overrightarrow{\mathbb{T}}_1} \quad \cdots \quad \bot \cdot ({}^0L_i \leftrightarrow {}^nL') \quad \cdots \quad \dfrac{{}^0L_k}{\overrightarrow{\mathbb{T}}_k}}$$

Maintenant nous n'avons que à rafraîchir les indices auprès des variables dans les sous-arbres dupliqués et rendre l'indice 0 à la clause de début. Nous obtenons un **CT**-tableau clos bien-formé $\mathbb{T}^*$. Notez que la substitution qui résout les contraintes dans $\mathbb{T}^*$ est facilement obtenu de la substitution pour $\mathbb{T}$ par duplication et réindexation nécessaires. Comme nous avons déjà remarqué, C intervient dans $\mathbb{T}^*$ à la profondeur inférieure que dans $\mathbb{T}$ ce qui nous amène à la contradiction. □

4.3.4 GDT et CT liés

En premier lieu, nous devons connecter les formules et les clauses. Nous commençons par un ensemble Γ de formules closes tel que aucun deux quantificateurs dans Γ ne bornent pas la

même variable. Nous définissons la fonction de *clausification naïve* $\mathcal{C}ls$ comme suit :

$$\mathcal{C}ls(\Gamma) = \bigcup_{F\in\Gamma} \mathcal{C}ls_{\varepsilon}(F) \qquad \mathcal{C}ls_{\vec{x}}(\neg\neg F) = \mathcal{C}ls_{\vec{x}}(F)$$

$$\mathcal{C}ls_{\vec{x}}(F \wedge G) = \mathcal{C}ls_{\vec{x}}(F) \cup \mathcal{C}ls_{\vec{x}}(G) \qquad \mathcal{C}ls_{\vec{x}}(\neg(F \wedge G)) = \mathcal{C}ls_{\vec{x}}(\neg F) \vee \mathcal{C}ls_{\vec{x}}(\neg G)$$

$$\mathcal{C}ls_{\vec{x}}(F \vee G) = \mathcal{C}ls_{\vec{x}}(F) \vee \mathcal{C}ls_{\vec{x}}(G) \qquad \mathcal{C}ls_{\vec{x}}(\neg(F \vee G)) = \mathcal{C}ls_{\vec{x}}(\neg F) \cup \mathcal{C}ls_{\vec{x}}(\neg G)$$

$$\mathcal{C}ls_{\vec{x}}(F \supset G) = \mathcal{C}ls_{\vec{x}}(\neg F) \vee \mathcal{C}ls_{\vec{x}}(G) \qquad \mathcal{C}ls_{\vec{x}}(\neg(F \supset G)) = \mathcal{C}ls_{\vec{x}}(F) \cup \mathcal{C}ls_{\vec{x}}(\neg G)$$

$$\mathcal{C}ls_{\vec{x}}(\forall\, u\, F) = \mathcal{C}ls_{\vec{x},u}(F) \qquad \mathcal{C}ls_{\vec{x}}(\neg\forall\, v\, F) = \mathcal{C}ls_{\vec{x}}(\neg F[\mathbf{v}(\vec{x})/v])$$

$$\mathcal{C}ls_{\vec{x}}(\exists\, v\, F) = \mathcal{C}ls_{\vec{x}}(F[\mathbf{v}(\vec{x})/v]) \qquad \mathcal{C}ls_{\vec{x}}(\neg\exists\, u\, F) = \mathcal{C}ls_{\vec{x},u}(\neg F)$$

$$\mathcal{C}ls_{\vec{x}}(P(\vec{s})) = \{\, P(\vec{s})\,\} \qquad \mathcal{C}ls_{\vec{x}}(\neg P(\vec{s})) = \{\, \neg P(\vec{s})\,\}$$

où $S_1 \vee S_2 = \{\, C_1 \vee C_2 \,|\, C_1 \in S_1, C_2 \in S_2 \,\}$ (S_1, S_2 sont des ensembles de clauses). Notez que dans les règles pour $\mathcal{C}ls_{\vec{x}}(\exists\, v\, F)$ et $\mathcal{C}ls_{\vec{x}}(\neg\forall\, v\, F)$, la variable v devient un symbole de Skolem $\mathbf{v}$.

Le théorème suivant est un corollaire évident du théorème 4.3.6.

Théorème 4.3.7. *Soit $\Gamma = \Delta \cup \{G\}$ un ensemble sans-collision de formules closes. Supposons que Δ est satisfaisable. Alors Γ est contradictoire si, et seulement si, $\mathcal{C}ls(\Gamma)$ peut être réfuté dans* **CT**. *D'ailleurs, si Γ est contradictoire, alors il y existe une* **CT***-réfutation de $\mathcal{C}ls(\Gamma)$ qui débute par une clause de $\mathcal{C}ls_{\varepsilon}(G)$.*

Un **GDT**-arbre (ou son sous-arbre) est dit *pointu* si toutes ses feuilles sont des littéraux. Par exemple, n'importe quel tableau clos est pointu. Pour tout sous-arbre pointu $\mathbb{T}$ d'un **GDT**-tableau nous définissons la *littéralisation* de $\mathbb{T}$, $\mathcal{L}it(\mathbb{T})$, comme une séquence d'arbres :

1. Si le noeud de racine dans $\mathbb{T}$ contient l'ensemble Γ des formules initiales, alors $\mathcal{L}it(\mathbb{T})$ est un arbre singulier :

$$\mathcal{L}it\left(\frac{\Gamma}{\mathbb{T}'}\right) \;=\; \frac{\mathcal{C}ls(\Gamma)}{\mathcal{L}it(\mathbb{T}')\theta_{\Gamma}}$$

   où la substitution skolémisante θ_{Γ} est appliquée aux formules et aux contraintes dans la littéralisation $\mathcal{L}it(\mathbb{T}')$.

2. Si le noeud de racine dans $\mathbb{T}$ contient une formule non-littérale, alors $\mathcal{L}it(\mathbb{T})$ est une concaténation :

$$\mathcal{L}it\left(\frac{F \diamond \tau\cdot\gamma}{\mathbb{T}_1}\right) \;=\; \mathcal{L}it(\mathbb{T}_1 + \gamma)$$

$$\mathcal{L}it\left(\frac{(G * H) \diamond 0.\tau\cdot\gamma}{\mathbb{T}_1 \qquad \mathbb{T}_2}\right) \;=\; \mathcal{L}it(\mathbb{T}_1 + \gamma),\;\; \mathcal{L}it(\mathbb{T}_2)$$

$$\mathcal{L}it\left(\frac{(G * H) \diamond 1.\tau\cdot\gamma}{\mathbb{T}_1 \qquad \mathbb{T}_2}\right) \;=\; \mathcal{L}it(\mathbb{T}_1),\;\; \mathcal{L}it(\mathbb{T}_2 + \gamma)$$

$$\mathcal{L}it\left(\frac{\neg(G * H) \diamond 0.0.\tau\cdot\gamma}{\mathbb{T}_1 \qquad \mathbb{T}_2}\right) \;=\; \mathcal{L}it(\mathbb{T}_1 + \gamma),\;\; \mathcal{L}it(\mathbb{T}_2)$$

$$\mathcal{L}it\left(\frac{\neg(G * H) \diamond 0.1.\tau\cdot\gamma}{\mathbb{T}_1 \qquad \mathbb{T}_2}\right) \;=\; \mathcal{L}it(\mathbb{T}_1),\;\; \mathcal{L}it(\mathbb{T}_2 + \gamma)$$

   où F est une formule non-littérale, $*$ est une opération propositionnelle binaire, et $\mathbb{T}_i + \gamma$ désigne l'arbre $\mathbb{T}_i$ où la contrainte γ est ajoutée à la contrainte du noeud de racine.

3. Si $\mathbb{T}$ contient un littéral dans le noeud de racine, alors $\mathcal{L}it(\mathbb{T})$ est un arbre singulier. Si le noeud de racine de $\mathbb{T}$ n'a pas de descendants, alors $\mathcal{L}it(\mathbb{T})$ contient un seul noeud aussi.

$$\mathcal{L}it\left(\frac{L \diamond \tau\cdot\gamma}{\mathbb{T}'}\right) \;=\; \frac{L\cdot\gamma}{\mathcal{L}it(\mathbb{T}')} \qquad\qquad \mathcal{L}it\,(L \diamond \tau\cdot\gamma) \;=\; (L\cdot\gamma)$$

Revenons à l'exemple à la figure 4.3. Pour cet ensemble initial Γ, nous avons :

$$\mathcal{C}ls(\Gamma) = \{\ \neg(p \subseteq q) \vee \neg(x \in p) \vee (x \in q),\ \ \neg E(r) \vee \neg(y \in r),\ \ (\mathbf{z}(s) \in s) \vee E(s),\ \ E(\mathbf{e}),\ \ \mathbf{a} \subseteq b,\ \ \neg E(\mathbf{a})\ \}$$

L'arbre littéralisé est la suivante (nous plaçons des contraintes sous les formules pour entrer dans la largeur de page). Cette inférence est un **CT**-tableau bien-formé et clos :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{C}ls(\Gamma)}{\neg E(\mathbf{a})}}{\mathbf{z}({}^1s) \in {}^1s}}{\dfrac{\neg\, {}^2p \subseteq {}^2q}{\bot}\ ({}^2p = \mathbf{a} \wedge {}^2q = {}^3b)} \qquad \bot\ (\mathbf{z}({}^1s) = {}^2x \wedge {}^1s = {}^2p)}}{\dfrac{\dfrac{\dfrac{{}^2x \in {}^2q}{\neg E({}^4r)}}{\bot}\ ({}^4r = \mathbf{e})}{\bot\ ({}^2x = {}^4y \wedge {}^2q = {}^4r)}}}{\bot}}{\bot\ (\mathbf{a} = {}^1s)}$$

Étendons les règles de clausification naïve aux formules avec des variables indexées :

$$\mathcal{C}ls_{\vec{x}}(\forall\, {}^k u\, F) = \mathcal{C}ls_{\vec{x}}(F) \qquad \mathcal{C}ls_{\vec{x}}(\neg\forall\, {}^k \mathbf{v}\, F) = \mathcal{C}ls_{\vec{x}}(\neg F)$$
$$\mathcal{C}ls_{\vec{x}}(\exists\, {}^k \mathbf{v}\, F) = \mathcal{C}ls_{\vec{x}}(F) \qquad \mathcal{C}ls_{\vec{x}}(\neg\exists\, {}^k u\, F) = \mathcal{C}ls_{\vec{x}}(\neg F)$$

Il est facile de voir que pour toute formule close $F \in \Gamma$, $\mathcal{C}ls_\varepsilon({}^k F)\theta_\Gamma = {}^k(\mathcal{C}ls_\varepsilon(F))$.

Lemme 4.3.8. *Soit Γ un ensemble sans-collision de formules closes. Soit $\mathbb{T}'$ un sous-arbre pointu d'un* **GDT***-tableau construit à partir de Γ et γ', une contrainte quelconque. Mettons $\mathbb{T} = \mathbb{T}' + \gamma'$ et supposons que $(F \diamond \tau \cdot \gamma)$ est le noeud de racine de $\mathbb{T}$. Alors les affirmations suivantes sont vraies :*

1. *Tout noeud dans $\mathcal{L}it(\mathbb{T})$ est une paire (littéral · contrainte).*
2. *Tous les littéraux qui interviennent dans $\mathbb{T}$ interviennent aussi dans $\mathcal{L}it(\mathbb{T})$; leur position relative (être au dessus de, être à gauche de) est préservée ; $\mathcal{L}it(\mathbb{T})$ ne contient pas d'autres littéraux.*
3. *Toutes les contraintes qui interviennent dans $\mathbb{T}$ interviennent aussi dans $\mathcal{L}it(\mathbb{T})$; $\mathcal{L}it(\mathbb{T})$ ne contient pas d'autres contraintes.*
4. *Soit $L_1, \ldots, L_n$ les littéraux dans les noeuds de racine dans $\mathcal{L}it(\mathbb{T})$. Alors la clause $(L_1 \vee \cdots \vee L_n)$ appartient au $\mathcal{C}ls_\varepsilon(F)$.*
5. *Que L désigne $F|_\tau$, si $\tau \in \Pi^+(F)$, et $\neg(F|_\tau)$, si non. Alors le noeud $(L \cdot \gamma)$ est un des noeuds de racine dans $\mathcal{L}it(\mathbb{T})$.*

Démonstration. Nous allons démontrer le lemme par induction sur le nombre de noeuds dans $\mathbb{T}$. Nous considérons plusieurs cas, selon la forme du noeud de racine de $\mathbb{T}$.

Cas où $F = G \vee H$ et $\tau = 0.\mu$. Comme $\mathbb{T}$ est pointu, le noeud de racine de $\mathbb{T}$ a des descendants. Comme $\mathbb{T}$ est un sous-arbre d'un **GDT**-tableau bien-formé (peut-être, avec des contraintes ajoutées), ces descendants sont $\mathbb{T}_1$ avec la racine $G \diamond \mu$ et $\mathbb{T}_2$ avec la racine $H \diamond \nu$, où ν est la position d'un atome quelconque dans H. Par définition, $\mathcal{L}it(\mathbb{T})$ est une concaténation de $\mathcal{L}it(\mathbb{T}_1 + \gamma)$ et $\mathcal{L}it(\mathbb{T}_2)$. Les arbres $\mathbb{T}_1 + \gamma$ et $\mathbb{T}_2$ sont pointus et ils ont moins de noeuds que $\mathbb{T}$. Ainsi, nous pouvons appliquer l'hypothèse d'induction. Vérifions les cinq affirmations du lemme :

1. Vrai : $\mathcal{L}it(\mathbb{T})$ est la concaténation de $\mathcal{L}it(\mathbb{T}_1 + \gamma)$ et $\mathcal{L}it(\mathbb{T}_2)$;
2. Vrai : $\mathbb{T}_1 + \gamma$ contient les mêmes littéraux que $\mathbb{T}_1$ et tous littéraux de $\mathcal{L}it(\mathbb{T}_1 + \gamma)$ restent à gauche des littéraux de $\mathcal{L}it(\mathbb{T}_2)$;
3. Vrai : en particulier, $\mathcal{L}it(\mathbb{T}_1 + \gamma)$ contient la contrainte γ ;
4. Soient $L_1, \ldots, L_k$ les littéraux dans les noeuds de racine dans l'arbre $\mathcal{L}it(\mathbb{T}_1 + \gamma)$. Soit $L_{k+1}, \ldots, L_n$ les littéraux dans les noeuds de racine dans $\mathcal{L}it(\mathbb{T}_2)$. Par l'hypothèse d'induction et la définition (étendue) de $\mathcal{C}ls$, $(L_1 \vee \cdots \vee L_n) \in \mathcal{C}ls_\varepsilon(F)$.

5. Notez que $F|_\tau = G|_\mu$ et l'arbre $\mathbb{T}_1$ a la contrainte vide dans le noeud de racine. Que L dénote $G|_\mu$, si $\mu \in \Pi^+(G)$, et $\neg(G|_\mu)$, si non. Par l'hypothèse d'induction, $(L \cdot \gamma)$ est un des noeuds de racine dans $\mathcal{L}it(\mathbb{T}_1 + \gamma)$.

Les autres cas où F est une formule non-littérale sont considérés d'une façon similaire.

Cas òu F est un littéral. Comme $\mathbb{T}$ obéit les règles d'inférence de **GDT**, le noeud de racine de $\mathbb{T}$ n'a pas plus qu'un descendant. Supposons qu'il en a un, $\mathbb{T}_1$. Par définition, $\mathcal{L}it(\mathbb{T}) = (F \cdot \gamma)/\mathcal{L}it(\mathbb{T}_1)$ et nous pouvons appliquer l'hypothèse d'induction pour $\mathbb{T}_1$. Les cinq affirmations du lemme peuvent alors être facilement démontrées pour $\mathcal{L}it(\mathbb{T})$. C'est aussi vrai pour le cas où le noeud de racine de $\mathbb{T}$ n'a pas de descendants. □

Théorème 4.3.9. *Soit Γ un ensemble sans-collision de formules closes. Pour tout* **GDT***-tableau pointu $\mathbb{T}$ qui est construit à partir de Γ, $\mathcal{L}it(\mathbb{T})$ est un* **CT***-tableau bien-formé ; si $\mathbb{T}$ est clos, alors $\mathcal{L}it(\mathbb{T})$ est aussi clos.*

Démonstration. Pour prouver la première partie du théorème, nous allons nous servir de l'induction sur le nombre des noeuds dans $\mathbb{T}$. Supposons d'abord que tous le noeuds avec littéraux sont des feuilles dans $\mathbb{T}$. Alors $\mathbb{T}$ est de la forme

$$\dfrac{\Gamma}{{}^0F \diamond \tau}$$
$$\vdots$$

où $F \in \Gamma$ et τ est une position d'un atome dans F. D'après le lemme 4.3.8, $\mathcal{L}it(\mathbb{T})$ est de la forme

$$\dfrac{\mathcal{C}ls(\Gamma)}{L_1\theta_\Gamma \quad \cdots \quad L_n\theta_\Gamma}$$

où $(L_1 \vee \cdots \vee L_n) \in \mathcal{C}ls_\varepsilon({}^0F)$. En effet, comme tous les littéraux dans $\mathbb{T}$ sont dans les feuilles, la littéralisation $\mathcal{L}it(\mathbb{T})$ est de la profondeur 2. Notez que $\mathcal{L}it(\mathbb{T})$ ne contient pas de contraintes, car $\mathbb{T}$ a été construit avec $\alpha\beta\gamma\delta$-règles qui ne produisent pas de contraintes. Par la définition de clausification naïve pour les formules avec des variables indexées, il y existe une clause $(L'_1 \vee \cdots \vee L'_n) \in \mathcal{C}ls_\varepsilon(F)$ telle que ${}^0L'_i = L_i\theta_\Gamma$ pour tout $i \in [1, n]$. Ainsi, $\mathcal{L}it(\mathbb{T})$ est un **CT**-tableau bien formé qui est construit à partir de $\mathcal{C}ls(\Gamma)$ par une application de la règle de début.

Maintenant, supposons que des littéraux interviennent dans $\mathbb{T}$ pas seulement dans les feuilles. Considérons un tel noeud N avec la profondeur maximale possible. Mettons que le littéral dans N est la négation d'une formule atomique $\neg P(\vec{r})$ (le cas d'un littéral positif est traite d'une manière analogique). Comme $\mathbb{T}$ est un **GDT**-tableau bien-formé, le noeud N est prolongé soit par sélection de nouveau but, soit par la règle de terminaison.

Supposons que N est étendu avec la règle de sélection de nouveau but. Alors le sous-arbre dont N est la racine est de la forme $N/\mathbb{T}_1$. Enlevons $\mathbb{T}_1$ de $\mathbb{T}$ et désignons le tableau obtenu par $\mathbb{T}'$. L'arbre $\mathbb{T}'$ est un **GDT**-tableau pointu bien-formé qui à moins de noeuds que $\mathbb{T}$, donc nous pouvons appliquer l'hypothèse d'induction. D'après le lemme 4.3.8, l'arbre $\mathcal{L}it(\mathbb{T}')$ à une feuille N' qui contient le littéral $\neg P(\vec{r})\theta_\Gamma$.

Le sous-arbre $\mathbb{T}_1$ est de la forme $({}^mF[\bot]_\tau \diamond \tau \cdot \gamma)/\mathbb{T}_2$, où la formule $F[P(\vec{s})^+]_\tau \in \Gamma$ et γ est la contrainte composée ${}^m\vec{s} = \vec{r}$. Évidemment, tous les noeuds avec des littéraux dans $\mathbb{T}_1$ sont des feuilles. D'après le lemme 4.3.8, la littéralisation $\mathcal{L}it(\mathbb{T}_1)$ est la séquence d'arbres-«singletones» $L_1, \ldots, (\bot \cdot \gamma), \ldots, L_n$, et la clause $(L_1 \vee \cdots \vee \bot \vee \cdots \vee L_n) \in \mathcal{C}ls_\varepsilon({}^mF[\bot]_\tau)$. Supposons que $\bot$ est le k-ème littéral dans la clause. Notez que $\mathcal{L}it(\mathbb{T}_1)$ ne contient pas de contraintes à part γ, car $\mathbb{T}_2$ à été généré par $\alpha\beta\gamma\delta$-règles.

Il est évident que la clause $(L_1 \vee \cdots \vee {}^mP(\vec{s}) \vee \cdots \vee L_n)$ appartient à $\mathcal{C}ls_\varepsilon({}^mF[P(\vec{s})]_\tau)$. Alors, par les règles de clausification naïve pour les formules avec des variables indexées, il y existe une clause $(L'_1 \vee \cdots \vee L'_n) \in \mathcal{C}ls_\varepsilon(F[P(\vec{s})]_\tau)$ telle que ${}^mL'_i = L_i\theta_\Gamma$ pour tout $i \in [1, n]\backslash\{k\}$ et ${}^mL'_k = ({}^mP(\vec{s}))\theta_\Gamma$. Dans **CT**, nous pouvons prolonger le noeud N' de l'arbre $\mathcal{L}it(\mathbb{T}')$ avec

les noeuds ${}^mL'_1, \ldots, (\bot \cdot \gamma\theta_\Gamma), \ldots, {}^mL'_n$. Par le précédent, c'est exactement $\mathcal{L}it(\mathbb{T})$ que nous obtenons.

Supposons maintenant que le noeud N dans $\mathbb{T}$ est étendu par la règle de terminaison. Alors il y existe un noeud M dans la branche de N qui contient le littéral $P(\vec{s})$ et le sous-arbre dont N est la racine est de la forme $N/(\bot \diamond \varepsilon \cdot \gamma)$ où γ est la contrainte composée $\vec{s} = \vec{r}$.

Comme avant, nous enlevons cette feuille de $\mathbb{T}$ et désignons l'arbre obtenu par $\mathbb{T}'$. Cet arbre est un **GDT**-tableau pointu bien-formé et il contient moins de noeuds que $\mathbb{T}$. Alors nous pouvons appliquer l'hypothèse d'induction. D'après le lemme 4.3.8, l'arbre $\mathcal{L}it(\mathbb{T}')$ a une feuille N' avec le littéral $\neg P(\vec{r})\theta_\Gamma$ et il y a un noeud M' dans la branche de N' qui contient le littéral $P(\vec{s})\theta_\Gamma$. Dans **CT**, nous pouvons étendre le noeud N' de l'arbre $\mathcal{L}it(\mathbb{T}')$ par le noeud $(\bot \cdot \gamma\theta_\Gamma)$. Le tableau obtenu est exactement $\mathcal{L}it(\mathbb{T})$.

Démontrons la deuxième partie du théorème. Supposons que $\mathbb{T}$ est un **GDT**-tableau clos. Alors toute branche de cet arbre contient un noeud avec $\bot$ et il y a une substitution admissible σ qui résout l'ensemble de contraintes $\bar{\gamma}$ dans $\mathbb{T}$. D'après le lemme 4.3.8, toute branche dans $\mathcal{L}it(\mathbb{T})$ contient un noeud avec $\bot$ aussi. D'ailleurs, l'ensemble des contraintes dans $\mathcal{L}it(\mathbb{T})$ est exactement $\bar{\gamma}\theta_\Gamma$. D'après le théorème 4.3.2, cet ensemble est satisfaisable, donc $\mathcal{L}it(\mathbb{T})$ est un **CT**-tableau clos. □

Lemme 4.3.10. *Soit F une formule avec des variables indexées. Soit $(L_1 \vee \cdots \vee L_n)$ une clause dans $\mathcal{C}ls_\varepsilon(F)$. Pour tout $i \in [1, n]$ il y existe une position τ dans $\Pi^+_{\mathbf{A}}(F)$ ou $\Pi^-_{\mathbf{A}}(F)$ telle que $L_i = F|_\tau$ ou $L_i = \neg(F|_\tau)$, respectivement, et pour tout atome A et tout* **GDT***-tableau $\mathbb{T}$ avec une feuille N de la forme $(F[A]_\tau \diamond \tau \cdot \gamma)$ il y existe un* **GDT***-tableau $\mathbb{T}'$ tel que*

1. *$\mathbb{T}'$ peut être obtenu de $\mathbb{T}$ par une série d'applications des $\alpha\beta\gamma\delta$-règles au-dessous de N.*
2. *le sous-arbre de $\mathbb{T}'$ dont N est la racine contient les littéraux L_1, …, L_{k-1}, L, L_{k+1}, …, L_n dans les feuilles, où $L = A$, si $\tau \in \Pi^+(F)$, et $L = \neg A$, si non.*

Démonstration. Ce lemme est facilement démontrable par induction sur la longueur de F.

Cas où F est un littéral. L'affirmation est évidente : τ est soit ε, soit 0 (car F est soit un atome, soit la négation d'un atome), et $\mathbb{T}' = \mathbb{T}$.

Cas où $F = G \vee H$. Pour quelque $k \in [1, n]$, la clause $(L_1 \vee \cdots \vee L_k)$ appartient à $\mathcal{C}ls_\varepsilon(G)$ et la clause $(L_{k+1} \vee \cdots \vee L_n)$ appartient à $\mathcal{C}ls_\varepsilon(H)$. Supposons que $i \leqslant k$ (le cas où $i > k$ est considéré d'une manière analogique). Prenons quelque $j \in [k+1, n]$. Par l'hypothèse d'induction, nous pouvons choisir des positions $\mu \in \Pi_{\mathbf{A}}(G)$ et $\nu \in \Pi_{\mathbf{A}}(H)$ qui satisferont l'affirmation du lemme. Mettons $\tau = 0.\mu$. Prenons un **GDT**-tableau $\mathbb{T}$ avec une feuille de la forme $(F[A]_\tau \diamond \tau \cdot \gamma)$. Nous pouvons prolonger $\mathbb{T}$ en appliquant la β-règle à cette feuille et générant deux descendants $(G[A]_\mu \diamond \mu)$ et $(H \diamond \nu)$. Par l'hypothèse d'induction, ces noeuds-là peuvent être étends par les $\alpha\beta\gamma\delta$-règles jusqu'à ce que nous obtenons l'arbre $\mathbb{T}'$.

Cas où $F = G \wedge H$. La clause $(L_1 \vee \cdots \vee L_k)$ appartient à $\mathcal{C}ls_\varepsilon(G) \cup \mathcal{C}ls_\varepsilon(H)$. Supposons qu'elle vient de $\mathcal{C}ls_\varepsilon(G)$ (l'autre cas est similaire). Soit i de l'intervalle $[1, n]$. Par l'hypothèse d'induction, nous pouvons choisir une position convenable $\mu \in \Pi_{\mathbf{A}}(G)$. Mettons $\tau = 0.\mu$. Prenons un **GDT**-tableau $\mathbb{T}$ avec une feuille de la forme $(F[A]_\tau \diamond \tau \cdot \gamma)$. Nous pouvons prolonger $\mathbb{T}$ en appliquant la α-règle à cette feuille et générant un descendant $(G[A]_\mu \diamond \mu)$. Par l'hypothèse d'induction, ce noeud peut être étend par les $\alpha\beta\gamma\delta$-règles jusqu'à ce que nous obtenons $\mathbb{T}'$.

Les autres cas sont traités d'une façon pareille. □

Théorème 4.3.11. *Soit Γ une séquence sans-collision de formules closes. Pour tout* **CT***-tableau $\mathbb{T}^\circ$ qui est construit à partir de $\mathcal{C}ls(\Gamma)$ et qui débute par une clause de la dernière formule dans Γ, il y existe un* **GDT***-tableau pointu $\mathbb{T}$, tel que $\mathbb{T}^\circ = \mathcal{L}it(\mathbb{T})$. Si $\mathbb{T}^\circ$ est clos, alors $\mathbb{T}$ est aussi clos.*

Démonstration. Pour démontrer la première partie du théorème, nous allons utiliser l'induction sur le nombre des noeuds dans $\mathbb{T}^\circ$. Supposons que $\mathbb{T}^\circ$ est obtenu d'un **CT**-tableau $\mathbb{T}^*$ par un pas d'extension (les cas de début et de terminaison sont traités d'une façon similaire). Supposons

d'ailleurs que $\mathbb{T}^*$ a une feuille N de la forme $\neg P(\vec{r})$ et qu'il y a une clause $(L_1 \vee \cdots \vee L_n) \in \mathcal{C}ls(\Gamma)$ telle que $L_k = P(\vec{s})$ et N a des noeuds-descendants ${}^mL_1, \ldots, (\bot \cdot \gamma), \ldots, {}^mL_n$ dans $\mathbb{T}^\circ$, où γ est la contrainte composée ${}^m\vec{s} = \vec{r}$.

L'arbre $\mathbb{T}^*$ est un **CT**-tableau bien-formé qui contient moins de noeuds que $\mathbb{T}^\circ$. Ainsi, nous pouvons appliquer l'hypothèse d'induction. Soit $\mathbb{T}'$ le **GDT**-tableau correspondant. Comme $\mathcal{L}it(\mathbb{T}') = \mathbb{T}^*$, il y a une feuille N' dans $\mathbb{T}'$ qui contient le littéral $\neg P(\vec{r}')$ tel que $\vec{r}'\theta_\Gamma = \vec{r}$. De plus, il y a une formule $F \in \Gamma$ et une clause $(L'_1 \vee \cdots \vee L'_n) \in \mathcal{C}ls_\varepsilon({}^mF)$ telles que $L'_i\theta_\Gamma = {}^mL_i$ pour tout $i \in [1, n]$ et $L'_k = P(\vec{s}')$.

Étant donné la formule ${}^mF$, la clause $(L'_1 \vee \cdots \vee L'_n)$, et l'indice k, le lemme 4.3.10 fixe une certaine position $\tau \in \Pi^+_{\mathbf{A}}({}^mF)$ telle que ${}^mF|_\tau = P(\vec{s}')$. Alors nous pouvons prolonger le noeud N' dans le **GDT**-tableau $\mathbb{T}'$ avec le noeud-descendant $({}^mF[\bot]_\tau \diamond \tau \cdot \gamma')$, où γ' est une contrainte composée ${}^m\vec{s}' = \vec{r}'$. Notons que $\gamma'\theta_\Gamma = \gamma$. D'après le lemme 4.3.10, l'arbre obtenu peut être étendu par les $\alpha\beta\gamma\delta$-règles appliquées au-dessous de N' jusqu'aux feuilles $L'_1, \ldots, \bot, \ldots, L'_n$. Soit $\mathbb{T}$ le **GDT**-tableau obtenu. Il est facile à voir que $\mathcal{L}it(\mathbb{T})$ peut être obtenu à partir de $\mathcal{L}it(\mathbb{T}')$ par expansion de N avec les noeuds $L'_1\theta_\Gamma, \ldots, (\bot \cdot \gamma'\theta_\Gamma), \ldots L'_n\theta_\Gamma$. Ainsi, $\mathcal{L}it(\mathbb{T}) = \mathbb{T}^\circ$.

L'autre partie du théorème est impliquée par le lemme 4.3.8 et le théorème 4.3.3. □

En somme, les théorèmes 4.3.9, 4.3.11, 4.3.7 démontrent la cohérence et la complétude du calcul **GDT** (théorème 4.3.4).

4.4 Traitement de l'égalité

Dans cette section nous développons un calcul cohérent et complet pour la logique classique du premier ordre avec égalité. Par simplicité, nous allons travailler avec les clauses : notre calcul **LPCT** n'est qu'une modification du calcul **CT**. La section précédente démontre comment on peut reformuler le calcul proposé en termes de formules générales et substitutions admissibles.

Nous notons l'égalité par le symbole $\approx$. La négation $\neg(s \approx t)$ est écrite $s \not\approx t$ et est dite «non-égalité» pour être distinguée d'inégalités que nous utilisons dans les contraintes. Nous considérons les égalités comme paires de termes non-ordonnées, c.-à-d. $a \approx b$ et $b \approx a$ désignent la même formule.

Le symbole $\simeq$ dénote la «pseudo-égalité», un symbole de prédicat binaire sans sémantique spécifique. Nous l'utilisons pour remplacer le symbole $\approx$ quand nous passons à la logique sans égalité. L'ordre d'argument y est significatif : $a \simeq b$ et $b \simeq a$ désignent les formules différentes. L'expression $s \not\simeq t$ désigne $\neg(s \simeq t)$.

Dans ce qui suit, les lettres p et q dénotent les termes non-variables, et les lettres l, r, s et t dénotent les termes arbitraires.

4.4.1 Tableaux de connexions et paramodulation

Il est bien tentant d'adapter les tableaux de connexions pour la logique avec égalité en y intégrant la règle de paramodulation : un littéral et une égalité paramodulante constitueront ainsi une connexion. Un tel calcul $\mathbf{CT}^\approx$ est montré à la figure 4.5.

Malheureusement, le calcul $\mathbf{CT}^\approx$ est incomplet. Considérons l'ensemble de clauses (contradictoire) suivant :

$$\mathcal{S} = \{a \approx b,\ \ c \approx d,\ \ \neg P(f(a), f(b)),\ \ \neg Q(g(c), g(d)),\ \ P(x, x) \vee Q(y, y)\}$$

Essayons de le réfuter dans $\mathbf{CT}^\approx$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{S}}{a \approx b}\,(\text{St})}{\neg P(f(b), f(b)) \cdot (a = a)}\,(\text{EqEx.2})}{\bot \cdot ({}^1x = f(b)) \qquad \dfrac{Q({}^1y, {}^1y)}{?}}\,(\text{Ex.2})}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{S}}{\neg Q(g(c), g(d))}\,(\text{St})}{\neg Q(g(d), g(d)) \cdot (c = c)}\,(\text{EqEx.1})}{\dfrac{P({}^1x, {}^1x)}{?} \qquad \bot \cdot ({}^1y = g(d))}\,(\text{Ex.2})}$$

Début :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L_k),\ \Lambda \parallel}{{}^0L_1 \quad \cdots \quad {}^0L_k}$$

Réduction :

$$\frac{\Gamma \parallel \Delta, s \not\approx t}{\bot \cdot (s = t)}$$

Expansion (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^m\vec{s} = \vec{r}) \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, \neg P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^m\vec{s} = \vec{r}) \quad \cdots \quad {}^mL_k}$$

Expansion égalitaire (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee l \approx r \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, L[p]}{{}^mL_1 \quad \cdots \quad L[{}^mr] \cdot (p = {}^ml) \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L[p] \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, l \approx r}{{}^mL_1 \quad \cdots \quad {}^mL[r] \cdot ({}^mp = l) \quad \cdots \quad {}^mL_k}$$

Paramodulation :

$$\frac{\Gamma \parallel \Delta, L[p], \Lambda, l \approx r}{L[r] \cdot (p = l)} \qquad \frac{\Gamma \parallel \Delta, l \approx r, \Lambda, L[p]}{L[r] \cdot (p = l)}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta, \neg P(\vec{s}), \Lambda, P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})} \qquad \frac{\Gamma \parallel \Delta, P(\vec{s}), \Lambda, \neg P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})}$$

FIG. 4.5: Tableaux de connexions avec paramodulation $\mathbf{CT}^{\approx}$

Nous ne pouvons pas continuer la première inférence, car le littéral $Q({}^1y, {}^1y)$ ne s'unifie pas avec $Q(g(c), g(d))$ et l'égalité $c \approx d$ ne s'applique pas non plus. L'autre inférence échoue d'une façon similaire.

La paramodulation «marche» bien dans les calculs résolutionnels [55] ainsi que dans les tableaux généraux [17, 23] et même dans les hyper-tableaux [6] grâce à l'ordre flexible de l'inférence qui est impossible dans un calcul orienté but.

Le calcul $\mathbf{CT}^{\approx}$ peut être rendu complet si on permet de paramoduler dans les variables et si on ajoute les axiomes de *réflexivité fonctionnelle* ($f(x) \approx f(x)$, $g(x) \approx g(x)$, etc) pour construire de nouveaux termes [42]. Bien que théoriquement admissible, cette approche est trop inefficace en pratique, car les axiomes réflexivité fonctionnelle permettent de substituer n'importe quel terme dans toute variable donnée.

Les démonstrateurs automatiques modernes basés sur les calculs de tableaux [51] utilisent des variants de la «méthode de modification» de Brand [8, 53, 5]. Cette méthode transforme un ensemble de clauses avec égalité en un ensemble équivalent (par rapport a satisfaisabilité) où l'égalité n'intervient pas. Outre cela, une procédure complète a été développée qui réunit recherche orientée but dans les tableaux avec saturation par paramodulation basique ordonnée dans une branche [52].

L'approche que nous proposons pour traitement de l'égalité dans les tableaux de connexions est basée sur la *paramodulation paresseuse* («lazy paramodulation»). Cette technique a été introduite par J. Gallier et W. Snyder en tant que méthode de E-unification générale [22]. Plus tard elle a été utilisée pour surmonter l'incomplétude de la stratégie d'ensemble de support (un

autre exemple de méthode orientée but) dans le calcul classique de paramodulation [66].

Alors, qu'est-ce que la paramodulation paresseuse ? Nous avons dit que le littéral $Q({}^1y, {}^1y)$ ne peut pas être unifié avec $Q(g(c), g(d))$. Retardons donc l'unification jusqu'à ce que l'égalité $c \approx d$ ne soit appliquée au deuxième littéral. Traitons l'égalité $Q({}^1y, {}^1y) = Q(g(c), g(d))$ pas comme une contrainte à résoudre mais comme un sous-but à démontrer.

Début : Réduction

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L_k),\ \Lambda \parallel}{{}^0L_1 \quad \cdots \quad {}^0L_k} \qquad \frac{\Gamma \parallel \Delta, s \not\approx t}{\bot \cdot (s = t)}$$

Expansion (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \quad {}^ms_1 \not\approx r_1 \quad \cdots \quad {}^ms_n \not\approx r_n \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, \neg P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \quad {}^ms_1 \not\approx r_1 \quad \cdots \quad {}^ms_n \not\approx r_n \quad \cdots \quad {}^mL_k}$$

Expansion égalitaire (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee l \approx r \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, L[p]}{{}^mL_1 \quad \cdots \quad L[{}^mr] \quad p \not\approx {}^ml \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L[p] \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, l \approx r}{{}^mL_1 \quad \cdots \quad {}^mL[r] \quad {}^mp \not\approx l \quad \cdots \quad {}^mL_k}$$

Paramodulation :

$$\frac{\Gamma \parallel \Delta, L[p], \Lambda, l \approx r}{L[r] \qquad p \not\approx l} \qquad \frac{\Gamma \parallel \Delta, l \approx r, \Lambda, L[p]}{L[r] \qquad p \not\approx l}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta, \neg P(\vec{s}), \Lambda, P(\vec{r})}{\bot \quad s_1 \not\approx r_1 \quad \cdots \quad s_n \not\approx r_n} \qquad \frac{\Gamma \parallel \Delta, P(\vec{s}), \Lambda, \neg P(\vec{r})}{\bot \quad s_1 \not\approx r_1 \quad \cdots \quad s_n \not\approx r_n}$$

FIG. 4.6: Tableaux de connexions avec paramodulation paresseuse (esquisse)

À la figure 4.6, nous donnons l'esquisse d'un calcul orienté but qui utilise la paramodulation paresseuse. L'ensemble $\mathcal{S}$ peut être facilement réfuté dans ce calcul :

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{S}}{a \approx b}}{\neg P(f(b), f(b))}}{\bot \quad \dfrac{{}^1x \not\approx f(b)}{\bot \cdot ({}^1x = f(b))} \quad \dfrac{Q({}^1y, {}^1y)}{\bot \quad \dfrac{g(c) \not\approx {}^1y}{\dfrac{g(d) \not\approx {}^1y}{\bot \cdot (g(d) = {}^1y)} \quad \dfrac{c \not\approx c}{\bot \cdot (c = c)}} \quad \dfrac{g(d) \not\approx {}^1y}{\bot \cdot (g(d) = {}^1y)}}}\ (\text{Ex.1}) \qquad \dfrac{a \not\approx a}{\bot \cdot (a = a)}}$$

Bien que l'approche semble valide, le calcul proposé n'est pas plus efficace que l'utilisation de la réflexivité fonctionnel. En effet, nous pouvons appliquer toute égalité à tout terme avec les règles ci-dessus. Est-ce que cette méthode peut être raffinée ? Sera-t-il complet ? Dans les sections suivantes nous allons donner les réponses positives à ces questions.

1. Élimination de symétrie :

$$\frac{L_1 \vee \cdots \vee s \approx t \vee \cdots \vee L_n}{L_1 \vee \cdots \vee s \simeq t \vee \cdots \vee L_n} \qquad \frac{L_1 \vee \cdots \vee s \approx t \vee \cdots \vee L_n}{L_1 \vee \cdots \vee t \simeq s \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee p \not\approx s \vee \cdots \vee L_n}{L_1 \vee \cdots \vee p \not\simeq s \vee \cdots \vee L_n} \qquad \frac{L_1 \vee \cdots \vee x \not\approx q \vee \cdots \vee L_n}{L_1 \vee \cdots \vee q \not\simeq x \vee \cdots \vee L_n}$$

2. Élimination de monotonicité :

$$\frac{L_1 \vee \cdots \vee P(s_1, \ldots, p, \ldots, s_n) \vee \cdots \vee L_n}{L_1 \vee \cdots \vee P(s_1, \ldots, \hat{u}, \ldots, s_n) \vee p \not\simeq \hat{u} \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee \neg P(s_1, \ldots, p, \ldots, s_n) \vee \cdots \vee L_n}{L_1 \vee \cdots \vee \neg P(s_1, \ldots, \hat{u}, \ldots, s_n) \vee p \not\simeq \hat{u} \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee f(s_1, \ldots, p, \ldots, s_n) \simeq t \vee \cdots \vee L_n}{L_1 \vee \cdots \vee f(s_1, \ldots, \hat{u}, \ldots, s_n) \simeq t \vee p \not\simeq \hat{u} \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee f(s_1, \ldots, p, \ldots, s_n) \not\simeq t \vee \cdots \vee L_n}{L_1 \vee \cdots \vee f(s_1, \ldots, \hat{u}, \ldots, s_n) \not\simeq t \vee p \not\simeq \hat{u} \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee t \simeq f(s_1, \ldots, p, \ldots, s_n) \vee \cdots \vee L_n}{L_1 \vee \cdots \vee t \simeq f(s_1, \ldots, \hat{u}, \ldots, s_n) \vee p \not\simeq \hat{u} \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee t \not\simeq f(s_1, \ldots, p, \ldots, s_n) \vee \cdots \vee L_n}{L_1 \vee \cdots \vee t \not\simeq f(s_1, \ldots, \hat{u}, \ldots, s_n) \vee p \not\simeq \hat{u} \vee \cdots \vee L_n}$$

3. Élimination de transitivité :

$$\frac{L_1 \vee \cdots \vee t \simeq q \vee \cdots \vee L_n}{L_1 \vee \cdots \vee t \simeq \hat{u} \vee q \not\simeq \hat{u} \vee \cdots \vee L_n}$$

$$\frac{L_1 \vee \cdots \vee p \not\simeq q \vee \cdots \vee L_n}{L_1 \vee \cdots \vee p \not\simeq \hat{u} \vee q \not\simeq \hat{u} \vee \cdots \vee L_n}$$

4. Introduction de réflexivité :

$$\frac{}{z \simeq z}$$

FIG. 4.7: Élimination contrainte de l'égalité

4.4.2 Élimination contrainte de l'égalité

L'élimination contrainte de l'égalité («constrained equality elimination» ou CEE) à été proposée par L. Bachmair et al. dans [5]. C'est une variante de la méthode de modification de Brand améliorée par l'utilisation de contraintes d'ordonnancement. Ici, nous expliquons la CEE-transformation sous une forme modifiée par rapport à la description originale dans [5]. Premièrement, nous admettons les prédicats autres que l'égalité. Deuxièmement, nous exigeons que toute occurrence d'un terme non-variable soit abstraité séparément pendant l'élimination de monotonicité. Troisièmement, nous appliquons les règles d'élimination de monotonicité après l'élimination de symétrie. Quatrièmement, nous opérons avec les clauses traditionnelles et intégrons les contraintes dans les règles d'inférence d'un calcul. Cinquièmement, nous traitons les occurrences de non-égalité de deux variables $x \not\approx y$ d'une façon différente. Ces modifications sont plutôt techniques et n'affectent pas le résultat principal (théorème 4.4.1).

Les quatre groupes de règles de réécriture à la figure 4.7 sont appliquées dans l'ordre d'apparition aux clauses de l'ensemble initial $\mathcal{S}$. Dénotons par $\mathrm{CEE}^i(\mathcal{S})$ le résultat intermédiaire de la transformation après le i-ème groupe. Les variables avec «chapeau» sont considérées comme fraîches dans la clause correspondante. Nous rappelons que les lettre p et q dénotent les termes

non-variables tandis que l, r, s et t dénote les termes arbitraires.

Le premier groupe remplace le symbole d'égalité $\approx$ par le symbole non-logique $\simeq$ et élimine le besoin des axiomes explicites de symétrie pour $\simeq$. Le deuxième groupe aplatit les termes et élimine ainsi le besoin des axiomes explicites de monotonicité pour $\simeq$. Le troisième groupe dissocie les littéraux de l'égalité et élimine ainsi le besoin des axiomes explicites de transitivité pour $\simeq$. La dernière règle ajoute explicitement l'axiome de réflexivité à l'ensemble de clauses.

Dans l'ensemble résultant $\mathrm{CEE}(\mathcal{S})$ $(= \mathrm{CEE}^4(\mathcal{S}))$, les résolutions entre les clauses correspondent aux paramodulations dans l'ensemble initial. Les variables introduites représentent, dans un certain sens, les «valeurs» des termes de coté gauche des nouvelles non-égalités. Nous entendons par «la valeur» le résultat final de toutes les paramodulations que l'on fait dans le terme.

Maintenant, nous assignons la contrainte atomique $p \succeq s$ à tout littéral négatif $p \not\simeq s$ qui intervient dans $\mathrm{CEE}(\mathcal{S})$. Nous assignons la contrainte $x = y$ à tout littéral négatif $x \not\simeq y$ dans $\mathrm{CEE}(\mathcal{S})$. Nous assignons la contrainte $s \succ t$ à tout littéral positif $s \simeq t$ dans $\mathrm{CEE}(\mathcal{S})$ à exception de l'axiome de réflexivité $z \simeq z$ qui ne reçoit pas de contrainte. Une *instance contrainte close* d'une clause $C \in \mathcal{S}$ est n'importe quelle clause $C\sigma$ telle que la substitution σ résout toutes les contraintes atomiques qui ont été assignées aux égalités et non-égalités dans C.

Le théorème suivant est l'homologue du théorème 4.1 de [5].

Théorème 4.4.1. *Un ensemble de clauses $\mathcal{S}$ est satisfaisable si, et seulement si, l'ensemble des instances contraintes closes des clauses de* $\mathrm{CEE}(\mathcal{S})$ *est satisfaisable.*

Considérons le calcul $\mathbf{CT}^{\simeq}$ de la figure 4.8. En essence, ce n'est qu'une extension de **CT** avec des contraintes d'ordonnancement pour les littéraux de l'égalité.

Dans les règles Début et Expansion, la clause initiale choisie n'est pas $(z \simeq z)$

Début :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L_k),\ \Lambda \parallel}{{}^0L_1 \quad \cdots \quad {}^0L_k}$$

Réduction :

$$\frac{\Gamma,\ (z \simeq z),\ \Lambda \parallel \Delta, s \not\simeq t}{\bot \cdot (s = t)}$$

Expansion (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^m\vec{s} = \vec{r}) \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, \neg P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^m\vec{s} = \vec{r}) \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee p \not\simeq t \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, l \simeq r}{{}^mL_1 \quad \cdots \quad \bot \cdot ({}^mp = l \wedge l > r \wedge r = {}^mt) \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee l \simeq r \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, p \not\simeq t}{{}^mL_1 \quad \cdots \quad \bot \cdot (p = {}^ml \wedge {}^ml > {}^mr \wedge {}^mr = t) \quad \cdots \quad {}^mL_k}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta, \neg P(\vec{s}), \Lambda, P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})} \qquad \frac{\Gamma \parallel \Delta, P(\vec{s}), \Lambda, \neg P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})}$$

$$\frac{\Gamma \parallel \Delta, p \not\simeq t, \Lambda, l \simeq r}{\bot \cdot (p = l \wedge l > r \wedge r = t)} \qquad \frac{\Gamma \parallel \Delta, l \simeq r, \Lambda, p \not\simeq t}{\bot \cdot (p = l \wedge l > r \wedge r = t)}$$

FIG. 4.8: Tableaux de connexions pour les CEE-clauses ($\mathbf{CT}^{\simeq}$)

Théorème 4.4.2. *Un ensemble de clauses $\mathcal{S}$ est contradictoire si, et seulement si, l'ensemble* CEE($\mathcal{S}$) *peut être réfuté dans le calcul* $\mathbf{CT}^{\simeq}$.

Démonstration. Ici nous donnons juste l'esquisse de la preuve, car les détails sont évidents. D'abord, démontrons la cohérence de $\mathbf{CT}^{\simeq}$ par rapport aux ensembles de clauses après la CEE-transformation. Considérons un $\mathbf{CT}^{\simeq}$-tableau clos $\mathbb{T}$ qui réfute l'ensemble CEE($\mathcal{S}$).

La substitution qui résout l'ensemble commun des contraintes dans $\mathbb{T}$ peut être complétée en une substitution close. Cela nous donne l'ensemble des instances closes des clauses de CEE($\mathcal{S}$). Cet ensemble est contradictoire, car nous pouvons transformer $\mathbb{T}$ en un **CT**-tableau bien-formé en effaçant les contraintes d'ordonnancement.

Il est facile de voir que ces instances closes sont des instances contraintes closes que l'on mentionne dans le théorème 4.4.1. En effet, tout littéral positif ($l \simeq r$) (sauf l'axiome de réflexivité) qui participe dans l'inférence reçoit la contrainte d'inégalité ($l \succ r$) pendant un pas d'extension ou de terminaison. Toute non-égalité ($s \not\simeq t$) est soit réduite à l'aide de l'axiome de réflexivité soit résolue avec quelque égalité positive. En tout cas, la contrainte ($s \succeq t$) sera satisfaite. Toute non-égalité ($x \not\simeq y$) est obligatoirement réduite par la réflexivité, ainsi la contrainte ($x = y$) est satisfaite aussi.

Démontrons la complétude de $\mathbf{CT}^{\simeq}$ par rapport aux clauses après la CEE-transformation. Considérons l'ensemble S de toutes les instances contraintes closes des clauses de CEE($\mathcal{S}$). D'après le théorème 4.4.1, S est contradictoire, et nous pouvons construire une réfutation de S dans le calcul **CT** qui ne débute pas par l'axiome de réflexivité ($z \simeq z$). Ensuite, nous montons l'inférence au premier ordre et transformons le **CT**-tableau en une $\mathbf{CT}^{\simeq}$-réfutation de CEE($\mathcal{S}$). □

4.4.3 Tableau de connexions avec paramodulation paresseuse

Nous somme prêts à introduire la version raffinée du calcul que l'on a esquissé dans la section 4.4.1. Les règles d'inférence du calcul **LPCT** sont données à la figure 4.9. Les variables «barrées» sont fraîches dans le tableau. Ces variables ne sont pas indexées.

Le calcul proposé contient plusieurs améliorations en comparaison avec le calcul esquissé à la figure 4.6. Premièrement, nous n'utilisons la paramodulation paresseuse que pendant l'extension ; les pas de paramodulation et de terminaison ne retardent pas l'unification. Deuxièmement, la «paresse» même est plus restreinte : deux termes non-variables dont on retarde l'unification doivent avoir le même symbole fonctionnel en haut. Troisièmement, nous utilisons les contraintes d'ordonnancement. Quatrièmement, nous utilisons les paramodulations basiques.

Notons qu'il y existe deux formes de la paramodulation basique. La première forme interdit de paramoduler dans les termes introduits par substitution. Le raffinement correspondant de la paramodulation paresseuse a été décrit par M. Moser [50]. Cette restriction est entièrement adoptée dans **LPCT**. En effet, nous plaçons des équations d'unification dans les contraintes et n'appliquons pas de substitutions aux littéraux.

La deuxième forme plus forte interdit d'ailleurs de paramoduler dans les termes introduits par des paramodulations précédentes [4]. Dans cette forme, les inférences basiques sont aussi utilisées dans **LPCT** (remarquez les variables avec la barre), quoi que pas partout : quand l'égalité paramodulante dans une expansion égalitaire provient de la clause par laquelle on étend la branche, le terme inséré est laissé «sur la surface», permis pour des paramodulations.

La cohérence de **LPCT** peut être démontrée directement, par la vérification du fait que les règles d'inférence ne produisent que ce qui s'ensuit de l'ensemble initial de clauses et des littéraux dans la branche considérée.

Nous démontrons la complétude de **LPCT** par transformation d'une $\mathbf{CT}^{\simeq}$-réfutation de l'ensemble des CEE-clauses en une **LPCT**-réfutation de l'ensemble des clauses initiales.

Théorème 4.4.3. *Pour tout ensemble contradictoire de clauses $\mathcal{S}$, il y existe une réfutation de $\mathcal{S}$ dans* **LPCT**.

Démonstration. Nous introduisons d'abord un calcul intermédiaire $\mathbf{LPCT}^{\simeq}$, dont les règles d'inférence sont celles de **LPCT** avec le symbole $\approx$ remplacé par $\simeq$.

Début :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L_k),\ \Lambda \parallel}{{}^0L_1 \quad \cdots \quad {}^0L_k}$$

Réduction :

$$\frac{\Gamma \parallel \Delta, s \not\approx t}{\bot \cdot (s = t)}$$

Expansion (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot (\vec{v} = \vec{r}) \quad {}^ms_1 \not\approx \bar{v}_1 \quad \cdots \quad {}^ms_n \not\approx \bar{v}_n \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee P(\vec{s}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, \neg P(\vec{r})}{{}^mL_1 \quad \cdots \quad \bot \cdot (\vec{v} = \vec{r}) \quad {}^ms_1 \not\approx \bar{v}_1 \quad \cdots \quad {}^ms_n \not\approx \bar{v}_n \quad \cdots \quad {}^mL_k}$$

Expansion égalitaire (m est un indice frais) :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee z \approx r \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, L[p]}{{}^mL_1 \quad \cdots \quad L[\bar{w}] \cdot (p = {}^mz > \bar{w}) \quad {}^mr \not\approx \bar{w} \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee f(\vec{s}) \approx r \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, L[p]}{{}^mL_1 \quad \cdots \quad L[\bar{w}] \cdot (p = f(\vec{v}) > \bar{w}) \quad {}^mr \not\approx \bar{w} \quad {}^ms_1 \not\approx \bar{v}_1 \quad \cdots \quad {}^ms_n \not\approx \bar{v}_n \quad \cdots \quad {}^mL_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee L[f(\vec{s})] \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, l \approx r}{{}^mL_1 \quad \cdots \quad {}^mL[\bar{w}] \cdot (f(\vec{v}) = l > r = \bar{w}) \quad {}^ms_1 \not\approx \bar{v}_1 \quad \cdots \quad {}^ms_n \not\approx \bar{v}_n \quad \cdots \quad {}^mL_k}$$

Paramodulation :

$$\frac{\Gamma \parallel \Delta, L[p], \Lambda, l \approx r}{L[\bar{w}] \cdot (p = l > r = \bar{w})} \qquad \frac{\Gamma \parallel \Delta, l \approx r, \Lambda, L[p]}{L[\bar{w}] \cdot (p = l > r = \bar{w})}$$

Terminaison :

$$\frac{\Gamma \parallel \Delta, \neg P(\vec{s}), \Lambda, P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})} \qquad \frac{\Gamma \parallel \Delta, P(\vec{s}), \Lambda, \neg P(\vec{r})}{\bot \cdot (\vec{s} = \vec{r})}$$

FIG. 4.9: Tableaux de connexions avec paramodulation paresseuse **LPCT**

À la première étape nous construisons un $\mathbf{CT}^{\simeq}$-tableau clos qui réfute l'ensemble CEE($\mathcal{S}$) (par le théorème 4.4.2) et le transformons en une $\mathbf{LPCT}^{\simeq}$-réfutation $\mathbb{T}$ de CEE^3($\mathcal{S}$). À la figure 4.10, nous démontrons comment les pas d'expansion et de terminaison dans $\mathbf{CT}^{\simeq}$ peuvent être simulés dans $\mathbf{LPCT}^{\simeq}$ ainsi que les feuilles dans les branches non-closes et les contraintes générées restent les mêmes. Remarquez que toute égalité et non-égalité dans CEE($\mathcal{S}$) a une variable de coté droite (par la définition de CEE).

À la deuxième étape nous restaurons les clauses aplaties. Appelons les variables avec «chapeau» qui ont été introduises par la CEE-transformation *les variables suspicieuses*. Appelons une clause *suspicieuse* si elle contient de variables suspicieuses. Appelons un pas de $\mathbf{LPCT}^{\simeq}$-inférence *suspicieux* si c'est un pas de début ou d'expansion ou d'expansion égalitaire qui engage une clause *suspicieuse*.

Soit $\mathcal{S}^{(0)} = \text{CEE}^3(\mathcal{S})$ et $\mathbb{T}^{(0)} = \mathbb{T}$. Nous allons construire une séquence d'ensembles de clauses et de $\mathbf{LPCT}^{\simeq}$-tableaux telle que les affirmations suivantes sont vraies pour tout $i > 0$:

- $\mathbb{T}^{(i)}$ est une réfutation de $\mathcal{S}^{(i)}$ dans $\mathbf{LPCT}^{\simeq}$;
- il y a moins de variables suspicieuses dans $\mathbb{T}^{(i)}$ que dans $\mathbb{T}^{(i-1)}$;
- toute variable suspicieuse $\hat{u}$ intervient exactement deux fois dans une clause de $\mathcal{S}(i)$: une fois dans une non-égalité de la forme $(p \not\approx \hat{u})$ et une fois dans quelque autre littéral (mais

$\mathbf{CT}^{\simeq}$-Terminaison $\implies$ $\mathbf{LPCT}$-Paramodulation :

$$\frac{\Gamma \parallel \Delta, f(\vec{x}) \not\simeq y, \Lambda, l \simeq u}{\bot \cdot (f(\vec{x}) = l > u = y)} \quad \Longrightarrow \quad \frac{\Gamma \parallel \Delta, f(\vec{x}) \not\simeq y, \Lambda, l \simeq u}{\dfrac{\bar{w} \not\simeq y \cdot (f(\vec{x}) = l > u = \bar{w})}{\bot \cdot (\bar{w} = y)}}$$

$\mathbf{CT}^{\simeq}$-Expansion $\implies$ $\mathbf{LPCT}$-Expansion :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{x}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{y})}{{}^m L_1 \quad \cdots \quad \bot \cdot ({}^m\vec{x} = \vec{y}) \quad \cdots \quad {}^m L_k} \quad \Longrightarrow$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee \neg P(\vec{x}) \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, P(\vec{y})}{{}^m L_1 \quad \cdots \quad \bot \cdot (\bar{v}_1 = y_1 \wedge \cdots \wedge \bar{v}_n = y_n) \quad \dfrac{{}^m x_1 \not\simeq \bar{v}_1}{\bot \cdot ({}^m x_1 = \bar{v}_1)} \quad \cdots \quad \dfrac{{}^m x_n \not\simeq \bar{v}_n}{\bot \cdot ({}^m x_n = \bar{v}_n)} \quad \cdots \quad {}^m L_k}$$

$\mathbf{CT}^{\simeq}$-Expansion $\implies$ $\mathbf{LPCT}$-ExpansionÉgalité :

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee f(\vec{x}) \not\simeq y \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, l \simeq u}{{}^m L_1 \quad \cdots \quad \bot \cdot ({}^m f(\vec{x}) = l > u = {}^m y) \quad \cdots \quad {}^m L_k} \quad \Longrightarrow$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee f(\vec{x}) \not\simeq y \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, l \simeq u}{{}^m L_1 \quad \cdots \quad \dfrac{\bar{w} \not\simeq {}^m y \cdot (f(\vec{\bar{v}}) = l > u = \bar{w})}{\bot \cdot (\bar{w} = {}^m y)} \quad \dfrac{{}^m x_1 \not\simeq \bar{v}_1}{\bot \cdot ({}^m x_1 = \bar{v}_1)} \quad \cdots \quad \dfrac{{}^m x_n \not\simeq \bar{v}_n}{\bot \cdot ({}^m x_n = \bar{v}_n)} \quad \cdots \quad {}^m L_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee z \simeq u \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, f(\vec{x}) \not\simeq y}{{}^m L_1 \quad \cdots \quad \bot \cdot (f(\vec{x}) = {}^m z > {}^m u = y) \quad \cdots \quad {}^m L_k} \quad \Longrightarrow$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee z \simeq u \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, f(\vec{x}) \not\simeq y}{{}^m L_1 \quad \cdots \quad \dfrac{\bar{w} \not\simeq y \cdot (f(\vec{x}) = {}^m z > \bar{w})}{\bot \cdot (\bar{w} = y)} \quad \dfrac{{}^m u \not\simeq \bar{w}}{\bot \cdot ({}^m u = \bar{w})} \quad \cdots \quad {}^m L_k}$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee f(\vec{z}) \simeq u \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, f(\vec{x}) \not\simeq y}{{}^m L_1 \quad \cdots \quad \bot \cdot (f(\vec{x}) = {}^m f(\vec{z}) > {}^m u = y) \quad \cdots \quad {}^m L_k} \quad \Longrightarrow$$

$$\frac{\Gamma,\ (L_1 \vee \cdots \vee f(\vec{z}) \simeq u \vee \cdots \vee L_k),\ \Lambda \parallel \Delta, f(\vec{x}) \not\simeq y}{{}^m L_1 \quad \cdots \quad \dfrac{\bar{w} \not\simeq y \cdot (f(\vec{x}) = f(\vec{\bar{v}}) > \bar{w})}{\bot \cdot (\bar{w} = y)} \quad \dfrac{{}^m u \not\simeq \bar{w}}{\bot \cdot ({}^m u = \bar{w})} \quad \dfrac{{}^m z_1 \not\simeq \bar{v}_1}{\bot \cdot ({}^m z_1 = \bar{v}_1)} \quad \cdots \quad \dfrac{{}^m z_n \not\simeq \bar{v}_n}{\bot \cdot ({}^m z_n = \bar{v}_n)} \quad \cdots \quad {}^m L_k}$$

FIG. 4.10: Transformation de $\mathbf{CT}^{\simeq}$ en $\mathbf{LPCT}^{\simeq}$

jamais comme l'argument gauche d'une (non)-égalité).

– toute clause non-suspicieuse dans $\mathcal{S}^{(i)}$ appartient à $\mathrm{CEE}^1(\mathcal{S})$.

Considérons une inférence suspicieuse I dans $\mathbb{T}^{(i-1)}$ telle qu'il n'y a pas de pas suspicieux au dessous de I. Soit $\hat{u}$ la variable suspicieuse qui intervient dans le tableau par ce pas. La clause suspicieuse correspondante est de la forme $(L[\hat{u}] \vee p \not\approx \hat{u} \vee C)$ (nous négligeons l'ordre des littéraux). Soit $\mathcal{S}^{(i)} = \mathcal{S}^{(i-1)} \cup \{ L[p] \vee C \}$.

Remarquez qu'un des deux littéraux contenant une occurrence de $\hat{u}$ peut être un «littéral actif» dans I. Ce littéral n'apparaîtra pas dans $\mathbb{T}^{(i-1)}$ sous sa forme originale. Cependant, nous pouvons affirmer que $\mathbb{T}^{(i-1)}$ contient deux sous-arbres disjoints, $\mathbb{T}^\circ$ et $\mathbb{T}^\bullet$ tels que :

– $\mathbb{T}^\circ$ et $\mathbb{T}^\bullet$ débutent au pas I ;

– $\hat{u}$ n'intervient pas en dehors de $\mathbb{T}^\circ$ et $\mathbb{T}^\bullet$;

– le littéral dans la racine de $\mathbb{T}^\bullet$ est de la forme $s \not\approx \hat{u}$ et $\hat{u}$ n'intervient pas dans s ;

– d'ailleurs, $\hat{u}$ n'intervient dans $\mathbb{T}^\bullet$ que dans les non-égalités $(t \not\approx \hat{u})$ et dans les contraintes $(t = \hat{u})$ qui sont introduisent par un pas de réduction ; $\hat{u}$ n'intervient pas dans ces termes t (en effet, tout ce que nous pouvons faire avec la non-égalité $(t \not\approx \hat{u})$, c'est la réduire ou la paramoduler dans t) ;

– $\hat{u}$ intervient exactement une fois dans le littéral de la racine de $\mathbb{T}^\circ$;

– $\hat{u}$ n'intervient pas dans la contrainte de la racine de $\mathbb{T}^\circ$.

Mettons que $\mathbb{T}^\circ$ est de la forme :

$$\frac{M[\hat{u}] \cdot \delta}{\mathbb{T}_1 \quad \cdots \quad \mathbb{T}_n}$$

Dans ce qui suit, $\mathbb{T}_k[\hat{u} \leftarrow t]$ désigne l'arbre $\mathbb{T}_k$ où toutes les occurrences de $\hat{u}$ (dans des littéraux et dans des contraintes) sont remplacées par t. Il est facile de voir qu'une telle substitution ne rend pas le tableau $\mathbb{T}_k$ mal-formé, à condition que $\hat{u}$ et t soient égaux par rapport aux contraintes dans $\mathbb{T}^{(i-1)}$. Une transformation d'arbre $[T]^{\mathbb{T}^\circ}$ est définie comme suit :

$$\left[\frac{t \not\approx \hat{u} \cdot \gamma}{\bot \cdot (t = \hat{u})}\right]^{\mathbb{T}^\circ} \quad \Longrightarrow \quad \frac{M[t] \cdot \gamma}{\mathbb{T}_1[\hat{u} \leftarrow t] \quad \cdots \quad \mathbb{T}_n[\hat{u} \leftarrow t]}$$

$$\left[\frac{t \not\approx \hat{u} \cdot \gamma}{T_1 \quad \cdots \quad T_n}\right]^{\mathbb{T}^\circ} \quad \Longrightarrow \quad \frac{M[t] \cdot \gamma}{[T_1]^{\mathbb{T}^\circ} \quad \cdots \quad [T_n]^{\mathbb{T}^\circ}}$$

$$\left[\frac{L \cdot \gamma}{T_1 \quad \cdots \quad T_n}\right]^{\mathbb{T}^\circ} \quad \Longrightarrow \quad \frac{L \cdot \gamma}{[T_1]^{\mathbb{T}^\circ} \quad \cdots \quad [T_n]^{\mathbb{T}^\circ}}$$

Considérons le tableau $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$. Nous pouvons affirmer que :

– La variable suspicieuse $\hat{u}$ n'intervient pas dans $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$.

– Étant donné que $(s \not\approx \hat{u})$ est le littéral dans la racine de $\mathbb{T}^\bullet$, le littéral $M[s]$ est le littéral dans la racine de $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$.

– Toute paramodulation que l'on fait dans un littéral de la forme $(t \not\approx \hat{u})$ dans $\mathbb{T}^\bullet$ est faite dans le terme t. C'est pourquoi on peut faire la même paramodulation dans le littéral correspondant $M[t]$ dans $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$.

– Tout littéral $(t \not\approx \hat{u})$ que l'on réduit dans $\mathbb{T}^\bullet$ devient le littéral $M[t]$ dans $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$. Ce littéral est d'ailleurs étendu par les sous-arbres $\mathbb{T}_i[\hat{u} \leftarrow t]$. Comme $\hat{u}$ et t sont égaux par rapport aux contraintes dans $\mathbb{T}^{(i-1)}$, l'arbre $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$ est clos.

Ensuite nous ajoutons la contrainte δ à la contrainte dans la racine de $[\mathbb{T}^\bullet]^{\mathbb{T}^\circ}$ et remplaçons $\mathbb{T}^\circ$ et $\mathbb{T}^\bullet$ dans $\mathbb{T}^{(i-1)}$ par cet arbre. Aussi, nous remplaçons $\mathcal{S}^{(i-1)}$ par $\mathcal{S}^{(i)}$ dans la racine. Le tableau résultant sera $\mathbb{T}^{(i)}$.

Dans $\mathbb{T}^{(i)}$, le pas qui correspond à I est fait avec la clause $L[p] \vee C$. Ensuite nous effectuons toutes les paramodulations dans p qui ont été faites dans $\mathbb{T}^\bullet$ et procédons par les inférences qui ont été faites dans $\mathbb{T}^\circ$. La variable $\hat{u}$ disparaît de $\mathbb{T}^{(i)}$ et aucune variable suspicieuse n'a été introduite. Il est facile de voir que les autres conditions sont aussi satisfaites.

En répétant cette procédure, nous arriverons à un tableau clos $\mathbb{T}^{(N)}$ où les variables suspicieuses n'interviennent pas du tout. Ce tableau est essentiellement une $\mathbf{LPCT}^{\simeq}$-réfutation de l'ensemble $\mathrm{CEE}^1(\mathcal{S})$. Il nous reste à annuler l'élimination de symétrie. Nous remplaçons le symbole $\simeq$ par $\approx$ et réorientons les égalités à leur forme originale dans $\mathcal{S}$.

En somme, nous obtenons un **LPCT**-réfutation bien-formée de $\mathcal{S}$. □

Malgré le chemin que nous avons pris pour démontrer la complétude, **LPCT** n'est pas une simple reformulation de CEE. En effet, il y a une différence essentielle entre l'aplatissement et la paramodulation paresseuse. Nous avons remarqué que les variables avec chapeau qui apparaissent dans les CEE-clauses peuvent être considérées comme les «valeurs» des termes qu'elles remplacent. Autrement dit, le terme qui est finalement substitué pour la variable $\hat{u}$ est en fait le résultat de toutes les paramodulations dans le terme t qui a été abstrait par $\hat{u}$ pendant la CEE-transformation. C'est pourquoi, dans une CEE-clause donnée, chaque terme a une seule «valeur». Ce n'est pas le cas pour **LPCT**.

Soit $\mathcal{S}$ l'ensemble $\{\, x \approx c \vee x \approx g(h(x)),\ f(c) \approx d,\ f(g(z)) \approx d,\ f(a) \not\approx d \,\}$. Le tableau suivant construit dans une version simplifiée de **LPCT** ne peut pas être obtenu d'une $\mathbf{CT}^{\simeq}$-réfutation de $\mathrm{CEE}(\mathcal{S})$.

$$
\begin{array}{c}
\mathcal{S} \\ \hline
f(a) \not\approx d \\ \hline
\begin{array}{cc}
\begin{array}{c}
x \approx c \\ \hline
\begin{array}{cc}
\begin{array}{c}
f(c) \not\approx d \\ \hline
f(c) \approx d \\ \hline
\begin{array}{cc}
\begin{array}{c} d \not\approx d \\ \hline \bot \end{array} &
\begin{array}{c} f(c) \not\approx f(c) \\ \hline \bot \end{array}
\end{array}
\end{array} &
\begin{array}{c}
x \not\approx a \\ \hline
\bot \cdot (x = a)
\end{array}
\end{array}
\end{array} &
\begin{array}{c}
x \approx g(h(x)) \\ \hline
\begin{array}{cc}
\begin{array}{c}
f(g(h(x))) \not\approx d \\ \hline
f(g(z)) \approx d \\ \hline
\begin{array}{cc}
\begin{array}{c} d \not\approx d \\ \hline \bot \end{array} &
\begin{array}{c} f(g(z)) \not\approx f(g(h(x))) \\ \hline \bot \cdot (z = h(x)) \end{array}
\end{array}
\end{array} &
\begin{array}{c}
x \not\approx a \\ \hline
\bot \cdot (x = a)
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
$$

Ici, nous remplaçons la constante a dans la clause de début par deux termes différents, c et $g(h(x))$. Si nous faisons les inférences avec des CEE-clauses, nous devrons prendre deux instances différentes de la clause de début. À la base de cet exemple, on peut démontrer que **LPCT** donne un raccourcissement exponentiel de la taille minimale de réfutation par rapport à $\mathbf{CT}^{\simeq}$ (mais en même temps le nombre des inférences possibles s'accroît).

Un autre point remarquable est la faiblesse d'unification. La procédure d'unification paresseuse que l'on utilise dans **LPCT** (qui compare les symboles fonctionnels en haut tout de suite et retarde le reste) est la même procédure qui a été proposée pour la paramodulation paresseuse dans [22]. Cette forme d'unification est beaucoup plus faible que «l'unification top» (introduite dans [19] et utilisée dans [66]) qui descend jusqu'aux variables. L'unification top permet de réduire radicalement les «obligations d'unification» retardées. En particulier, l'unifiabilité de deux termes clos peut être déterminée immédiatement.

Malheureusement, l'unification top et les contraintes d'ordonnancement ne peuvent pas être utilisées ensemble dans le cadre des tableaux de connexions. Considérons l'ordonnancement $a > b > c$ et l'ensemble $\mathcal{S} = \{\, P(c) \vee Q(c), \neg P(a), \neg Q(b), b \approx c, a \approx c \,\}$. Les contraintes d'ordonnancement interdisent les paramodulations dans c. Ainsi, la seule possibilité de réfuter $\mathcal{S}$ dans **LPCT** est de résoudre $P(c)$ avec $\neg P(a)$ ou $Q(c)$ avec $\neg Q(b)$. Or, ces paires ne sont pas unifiables du point de vue de l'unification top.

Il n'est pas clair si une méthode plus forte d'unification vaut mieux que les inférences ordonnées. D'ailleurs, nous ne connaissons aucune adaptation des tableaux de connexions pour la paramodulation paresseuse avec l'unification top. Développer et étudier un tel calcul est une des directions pour la recherche future.

Chapitre 5

Conclusion

Dans ce travail, nous avons présenté un nouvel assistant de preuve qui vérifie des textes formels avec des démonstrations déclaratives. Nous nous appuyons sur les éléments «naturels» du texte (genres de prémisses, schémas de preuve, prédicats-classes) pour faciliter la vérification. Nous employons un démonstrateur automatique puissant en logique du premier ordre pour compléter les taches de preuve.

Dans le chapitre 1, nous avons décrit le langage de théories formelles ForTheL proche de la langue et du style de textes mathématiques. Nous avons montré comment une proposition dans ce langage peut être traduite en une formule du premier ordre.

Dans le chapitre 2, nous avons défini la notion d'un texte correct dans ce langage. Cette notion comprend la «correction ontologique» qui peut être vue comme un analogue de la correction des types ou, plus généralement, la correction d'emploi de fonctions et relations partielles. Pour définir la correction ontologique, nous avons introduit la notion de «propriété locale» : une proposition qui est valide dans le contexte d'une occurrence donné à l'intérieur d'une formule.

Dans le chapitre 3, nous avons présenté l'assistant de preuve SAD où notre approche est implantée. Nous avons décrit la procédure de vérification et les algorithmes particuliers de démonstration de «haut niveau» : génération des «évidences» (simples propriétés-lemmes locales), transformation de la proposition démontrée au cours de la démonstration, filtrage de prémisses, décomposition de but, application de définitions.

Le chapitre 4 a poursuivi l'étude de la démonstration automatique orientée but dans le cadre de notre projet. Nous avons montré que le calcul orienté but **GDT** introduit dans les travaux précédents est essentiellement équivalent à la méthode d'élimination de modèles (tableaux des connections). Ensuite nous avons proposé une variante des tableaux de connexions cohérente et complète pour la logique du premier ordre avec égalité. Ce calcul est basé sur la paramodulation paresseuse.

Plusieurs textes non-triviaux formalisés en ForTheL et vérifiés dans SAD sont présentés dans l'annexe.

Sans doute, le travail présent n'est qu'une étape dans le projet général — mais une étape nécessaire. Les directions futures de recherche et du développement du système SAD sont nombreuses. Énumérons en quelques-unes qui nous paraissent plus immédiates ou plus importantes que les autres.

La réalisation courante du langage ForTheL manque de moyens «confortables» pour parler des objets d'ordre supérieur : des séquences, des matrices, des fonctions, des opérations sur les fonctions telles que limite, sommation et ainsi de suite. Nous avons déjà étudié une approche possible basée sur des combinateurs [46], mais la solution n'est pas tout à fait satisfaisante : un démonstrateur multi-sorte est nécessaire.

La collection de méthodes de démonstration de «haut niveau» doit évoluer constamment. L'application de définitions doit être généralisée à l'application de lemmes. La question ouverte ici est comment définir l'applicabilité d'un lemme. Contrairement au cas de définitions, les conditions et les conclusions dans un lemme ne sont pas clairement séparées. De plus, il n'est pas évident comment choisir la substitution à appliquer au lemme en question.

La procédure de filtrage de contexte doit être sérieusement révisée et raffinée pour rendre

possible la vérification de grands textes, surtout dans les théories algébriques. Notre but est d'éliminer autant que possible toutes prémisses trop «actives» de tâches du démonstrateur.

Il est aussi intéressant de concevoir une procédure de variation de démonstrations sur laquelle la vérification de preuves par analogie (même des plus simples) pourrait être basée.

Une autre question importante est quelles propriétés caractéristiques des tâches de démonstration générées par le système (à partir de textes mathématiques formalisés) peuvent être prises en compte par le démonstrateur automatique et faciliter la recherche de dérivations. À part la reconnaissance des littéraux-«sortes» dans les formules (ce qui est réalisé dans SPASS), on peut aussi concevoir la génération automatique d'ordonnancement de symboles de signature (e.g. selon la hiérarchie des définitions) et le traitement spécifique de certains genres de prémisses (e.g. les identités algébriques).

Finalement et le plus important, la collection de textes formalisés et vérifiés doit s'augmenter. Nous ne planifions pas d'ériger une grande théorie «de tout» à la Mizar. Toutefois, une bibliothèque de petites théories préliminaires qu'on pourrait facilement connecter l'une avec l'autre est, à notre avis, réalisable. Les théorèmes de la liste «Les Cent Grands Théorèmes» [33] sont un défi intéressant à cet égard.

Index

Bibliographie

[1] Anufriev, F.: *An algorithm of theorem proof search in logical calculi.* Teoriya avtomatov, 1, 1969. En russe.

[2] Anufriev, F., V. Fediurko, A. Letichevski, Z. Aselderov et I. Didukh: *On one algorithm of theorem proof search in group theory.* Kibernetika, 1:23–29, 1966. En russe.

[3] Atayan, V. et K. Vershinin: *On formalization of some inference search methods.* Dans *Avtomatizatsiya obrabotki matematicheskikh tekstov*, p. 36–52. Institute of Cybernetics, Kiev, 1980. En russe.

[4] Bachmair, L., H. Ganzinger, C. Lynch et W. Snyder: *Basic paramodulation.* Information and computation, 121(2):172–192, 1995.

[5] Bachmair, L., H. Ganzinger et A. Voronkov: *Elimination of equality via transformation with ordering constraints.* Dans Kirchner, C. et H. Kirchner (réds.): *Automated Deduction: 15th International Conference, CADE-15*, t. 1421 de *Lecture Notes in Computer Science*, p. 175–190. Springer, 1998.

[6] Baumgartner, P., U. Furbach et B. Pelzer: *Hyper Tableaux with Equality.* Dans Pfenning, F. (réd.): *Automated Deduction: 21st International Conference, CADE-21*, t. 4603 de *Lecture Notes in Computer Science*, p. 492–507. Springer, 2007.

[7] Bertot, Y. et P. Castéran: *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, t. XXV de *Texts in Theoretical Computer Science (EATCS)*. Springer, 2004.

[8] Brand, D.: *Proving Theorems with the Modification Method.* SIAM Journal of Computing, 4:412–430, 1975.

[9] Buchberger, B., A. Crăciun, T. Jebelean, L. Kovács, T. Kutsia, K. Nakagawa, F. Piroi, N. Popov, J. Robu, M. Rosenkranz et W. Windsteiger: *Theorema: Towards computer-aided mathematical theory exploration.* Journal of Applied Logic, 4(4):470–504, 2006.

[10] Corella, F.: *What holds in a context ?* Journal of Automated Reasoning, 10(2):79–93, 1993.

[11] D'Agostino, M., D. Gabbay, R. Hähnle et J. Posegga (réds.): *Handbook of Tableau Methods.* Springer, 1999.

[12] Degtyarev, A.: *The strategy of monotone paramodulation.* Dans *Proc. 5th All-Union Conference on mathematical logic*, Novosibirsk, USSR, 1979. En russe.

[13] Degtyarev, A., Yu. Kapitonova, A. Letichevsky, A. Lyaletski et M. Morokhovets: *A brief historical sketch on Kiev school of automated theorem proving.* Dans *Proc. 2nd International THEOREMA Workshop*, p. 151–156. Linz, Austria, 1998.

[14] Degtyarev, A. et A. Lyaletski: *Logical inference in SAD.* Dans Kapitonova, Yu. (réd.): *Matematicheskie osnovy sistem iskusstvennogo intellekta*, p. 3–11. Institute of Cybernetics, Kiev, 1981. En russe.

[15] Degtyarev, A., A. Lyaletski et M. Morokhovets: *Evidence Algorithm and sequent logical inference search.* Dans Ganzinger, H., D. A. McAllester et A. Voronkov (réds.): *Logic Programming and Automated Reasoning: 6th International Conference, LPAR'99*, t. 1705 de *Lecture Notes in Computer Science*, p. 44–61. Springer, 1999.

[16] Degtyarev, A., A. Lyaletski et M. Morokhovets: *On the EA-style integrated processing of self-contained mathematical texts.* Dans Kerber, M. et M. Kohlhase (réds.): *Symbolic Computation and Automated Reasoning: The CALCULEMUS-2000 Symposium*, p. 126–141. A.K. Peters, Ltd., USA, 2001.

[17] Degtyarev, A. et A. Voronkov: *What you always wanted to know about rigid E-unification.* Journal of Automated Reasoning, 20(1):47–80, 1998.

[18] Dixon, L. et J. D. Fleuriot: *A Proof-Centric approach to Mathematical Assistants.* Journal of Applied Logic, 4(4):505–532, 2006.

[19] Dougherty, D. J. et P. Johann: *An Improved General E-Unification Method.* Journal of Symbolic Computation, 14(4):303–320, 1992.

[20] Fuchs, N. E., U. Schwertel et R. Schwitter: *Attempto Controlled English — Not Just Another Logic Specification Language.* Dans Flener, P. (réd.): *Logic-Based Program Synthesis and Transformation: 8th International Workshop, LOPSTR'98*, t. 1559 de *Lecture Notes in Computer Science*, p. 1–20. Springer, 1999.

[21] Fuchs, N. E. et R. Schwitter: *Web-Annotations for Humans and Machines.* Dans Franconi, E., M. Kifer et W. May (réds.): *The Semantic Web: Research and Applications: 4th European Conference ESWC 2007*, t. 4519 de *Lecture Notes in Computer Science*, p. 458–472. Springer, 2007.

[22] Gallier, J. et W. Snyder: *Complete Sets of Transformations for General E-unification.* Theoretical Computer Science, 67:203–260, 1989.

[23] Giese, M.: *A Model Generation Style Completeness Proof for Constraint Tableaux with Superposition.* Dans Egly, U. et C. G. Fermüller (réds.): *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference, TABLEAUX 2002*, t. 2381 de *Lecture Notes in Computer Science*, p. 130–144. Springer, 2002.

[24] Glushkov, V.: *Some problems of automata theory and artificial intelligence.* Kibernetika, 2:3–13, 1970. En russe.

[25] Glushkov, V.: *The System for Automation of Proving.* Dans *Avtomatizatsiya obrabotki matematicheskikh tekstov.* Institute of Cybernetics, Kiev, 1980. En russe.

[26] Glushkov, V.: *Foundations of mathematics and automation of deductive constructing.* Dans *Kibernetika. Voprosy teoriyi i praktiki.* Nauka, Moskva, 1986. En russe.

[27] Glushkov, V., V. Kostyrko, A. Letichevski, F. Anufriev et Z. Aselderov: *On a language for description of formal theories.* Teoreticheskaya kibernetika, 3, 1970. En russe.

[28] Glushkov, V., K. Vershinin, Yu. Kapitonova, A. Letichevski, N. Malevaniy et V. Kostyrko: *On a formal language for description of mathematical texts.* Dans *Avtomatizatsiya poiska dokazatel'stv teorem v matematike*, p. 3–36. Institute of Cybernetics, Kiev, 1974. En russe.

[29] Gordon, M. J. C. et T. F. Melham: *Introduction to HOL: a theorem proving environment for higher order logic.* Cambridge University Press, 1993.

[30] Grundy, J.: *Transformational Hierarchical Reasoning.* The Computer Journal, 39(4):291–302, 1996.

[31] Harrison, J.: *Proof Style.* Dans Giménex, E. et C. Pausin-Mohring (réds.): *Types for Proofs and Programs: International Workshop TYPES'96*, t. 1512 de *Lecture Notes in Computer Science*, p. 154–172. Springer, 1996.

[32] Jones, S. P.: *Haskell 98 Language and Libraries.* Cambridge University Press, 2003.

[33] Kahl, N.: *The Hundred Greatest Theorems.* Attribué à P. Abad et J. Abad. Disponible sur http://personal.stevens.edu/~nkahl/Top100Theorems.html.

[34] Kamareddine, F. et R. P. Nederpelt: *A Refinement of de Bruijn's Formal Language of Mathematics.* Journal of Logic, Language and Information, 13(3):287–340, 2004.

[35] Kapitonova, Yu., K. Vershinin, A. Degtyarev, A. Zhezherun et A. Lyaletski: *System for processing mathematical texts.* Kibernetika, 2, 1979. En russe.

[36] Kaufmann, M., P. Manolios et J. S. Moore: *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.

[37] Kleene, S. C.: *Introduction to Metamathematics.* Van Nostrand, 1952.

[38] Lamport, L.: *Types considered harmful.* DEC SRC internal note, 1992.

[39] Letz, R., J. Schumann, S. Bayerl et W. Bibel: *SETHEO: a high-performance theorem prover.* Journal of Automated Reasoning, 8(2):183–212, 1992.

[40] Letz, R. et G. Stenz: *Model Elimination and Connection Tableau Procedures.* Dans Robinson, A. et A. Voronkov (réds.): *Handbook for Automated Reasoning*, t. II, chap. 28, p. 2017–2116. Elsevier, 2001.

[41] Loveland, D. W.: *Mechanical theorem proving by model elimination.* Journal of the ACM, 16(3):349–363, 1968.

[42] Loveland, D. W.: *Automated Theorem Proving: A Logical Basis*, t. 6 de *Fundamental Studies in Computer Science.* North-Holland, 1978.

[43] Lyaletski, A.: *Gentzen calculi and admissible substitutions.* Dans *Actes preliminaires du Symposium Franco-Sovetique «Informatika-91»*, p. 99–111, Grenoble, France, 1991.

[44] Lyaletski, A.: *On Herbrand theorem.* The Bulletin of Symbolic Logic, 7(1):132–133, 2001.

[45] Lyaletski, A., A. Paskevich et K. Verchinine: *Theorem Proving and Proof Verification in the System SAD.* Dans Asperti, A., G. Bancerek et A. Trybulec (réds.): *Mathematical Knowledge Management: 3rd International Conference, MKM 2004*, t. 3119 de *Lecture Notes in Computer Science*, p. 236–250. Springer, 2004.

[46] Lyaletski, A., A. Paskevich et K. Verchinine: *SAD as a mathematical assistant — how should we go from here to there ?* Journal of Applied Logic, 4(4):560–591, 2006.

[47] Lyaletski, A., K. Verchinine, A. Degtyarev et A. Paskevich: *System for Automated Deduction (SAD): Linguistic and Deductive Peculiarities.* Dans Klopotek, M. A., S. T. Wierzchon et M. Michalewicz (réds.): *Intelligent Information Systems: IIS'2002 Symposium*, Advances in Soft Computing, p. 413–422. Physica-Verlag, 2002.

[48] McCune, W.: *Otter 3.0 Reference Manual and Guide.* Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, USA, 1994.

[49] Monk, L. G.: *Inference rules using local contexts.* Journal of Automated Reasoning, 4(4):445–462, 1988.

[50] Moser, M.: *Improving Transformation Systems for General E-Unification.* Dans Kirchner, C. (réd.): *Rewriting Techniques and Applications: 5th International Conference, RTA 1993*, t. 690 de *Lecture Notes in Computer Science*, p. 92–105. Springer, 1993.

[51] Moser, M., O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann et K. Mayr: *SETHEO and E-SETHEO — The CADE-13 Systems.* Journal of Automated Reasoning, 18(2):237–246, 1997.

[52] Moser, M., C. Lynch et J. Steinbach: *Model elimination with basic ordered paramodulation.* Rap. tech. AR-95-11, Fakultät für Informatik, Technische Universität München, München, 1995.

[53] Moser, M. et J. Steinbach: *STE-modification revisited.* Rap. tech. AR-97-03, Fakultät für Informatik, Technische Universität München, München, 1997.

[54] Nederpelt, R. P., J. H. Geuvers et R. C. de Vrijer (réds.): *Selected Papers on Automath*, t. 133 de *Studies in Logic and the Foundations of Mathematics.* North-Holland, 1994.

[55] Nieuwenhuis, R. et A. Rubio: *Paramodulation-based theorem proving.* Dans Robinson, A. et A. Voronkov (réds.): *Handbook for Automated Reasoning*, t. I, chap. 7, p. 371–443. Elsevier, 2001.

[56] Nipkow, T., L. C. Paulson et M. Wenzel: *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, t. 2283 de *Lecture Notes in Computer Science*. Springer, 2002.

[57] Owre, S., J. M. Rushby et N. Shankar: *PVS: a Prototype Verification System.* Dans Kapur, D. (réd.): *Automated Deduction: 11th International Conference, CADE-11*, t. 607 de *Lecture Notes in Computer Science*, p. 748–752. Springer, 1992.

[58] Paskevich, A.: *Connection Tableaux with Lazy Paramodulation.* Dans Furbach, U. et N. Shankar (réds.): *Automated Reasoning: 3rd International Joint Conference IJCAR 2006*, t. 4130 de *Lecture Notes in Computer Science*, p. 112–124. Springer, 2006.

[59] Paskevich, A., K. Verchinine, A. Lyaletski et A. Anisimov: *Reasoning inside a formula and ontological correctness of a formal mathematical text.* Dans Kauers, M., M. Kerber, R. Miner et W. Windsteiger (réds.): *Calculemus/MKM 2007 — Work in Progress*, n° 07-06 dans *RISC-Linz Report Series*, p. 77–91. University of Linz, Austria, 2007.

[60] Pfenning, F.: *Logical Frameworks.* Dans Robinson, A. et A. Voronkov (réds.): *Handbook for Automated Reasoning*, t. II, chap. 17, p. 1061–1145. Elsevier, 2001.

[61] Riazanov, A. et A. Voronkov: *The design and implementation of VAMPIRE.* AI Communications, 15(2–3):91–110, 2002.

[62] Richardson, J., A. Smaill et I. Green: *System Description: Proof Planning in Higher-Order Logic with Lambda-Clam.* Dans *Automated Deduction: 15th International Conference, CADE-15*, t. 1421 de *Lecture Notes in Computer Science*, p. 129–133. Springer, 1998.

[63] Robinson, P. J. et J. Staples: *Formalising the hierarchical structure of practical mathematical reasoning.* Journal of Logic and Computation, 3(1):47–61, 1993.

[64] Schulz, S.: *System Description: E 0.81.* Dans Basin, D. et M. Rusinowitch (réds.): *Automated Reasoning: 2nd International Joint Conference, IJCAR-2004*, t. 3097 de *Lecture Notes in Computer Science*, p. 223–228. Springer, 2004.

[65] Siekmann, J., C. Benzmüller et S. Autexier: *Computer supported mathematics with* ΩMEGA. Journal of Applied Logic, 4(4):533–559, 2006.

[66] Snyder, W. et C. Lynch: *Goal directed strategies for paramodulation.* Dans Book, R. (réd.): *Rewriting Techniques and Applications: 4th International Conference, RTA 1991*, t. 448 de *Lecture Notes in Computer Science*, p. 150–161. Springer, 1991.

[67] Sowa, J. F.: *Common Logic Controlled English (unpublished draft).* Disponible sur http://www.jfsowa.com/clce/specs.htm, 2004.

[68] Sutcliffe, G., C. B. Suttner et T. Yemenis: *The TPTP Problem Library.* Dans Bundy, A. (réd.): *Automated Deduction: 12th International Conference, CADE-12*, t. 814 de *Lecture Notes in Computer Science*, p. 252–266. Springer, 1994.

[69] Trybulec, A. et H. Blair: *Computer assisted reasoning with Mizar.* Dans Joshi, A. K. (réd.): *Proc. 9th International Joint Conference on Artificial Intelligence, IJCAI 1985*, p. 26–28. Morgan-Kaufmann, 1985.

[70] Urban, J.: *Translating Mizar for First Order Theorem Provers.* Dans Asperti, A., B. Buchberger et J. H. Davenport (réds.): *Mathematical Knowledge Management: 2nd International Conference, MKM 2003*, t. 2594 de *Lecture Notes in Computer Science*, p. 203–215. Springer, 2003.

[71] Verchinine, K., A. Degtyarev, A. Lyaletski et A. Paskevich: *SAD, a System for Automated Deduction: a Current State.* Dans Kamareddine, F. (réd.): *Proc. Workshop on 35 Years of Automath*, Heriot-Watt University, Edinburgh, Scotland, 2002.

[72] Verchinine, K., A. Lyaletski et A. Paskevich: *System for Automated Deduction (SAD): a tool for proof verification.* Dans Pfenning, F. (réd.): *Automated Deduction: 21st International Conference, CADE-21*, t. 4603 de *Lecture Notes in Computer Science*, p. 398–403. Springer, 2007.

[73] Vershinin, K.: *On connection of a formal language of mathematical theories with axiomatic systems of the set theory.* Kibernetika, 4, 1973. En russe.

[74] Vershinin, K.: *Using auxiliary statements in the process of proof search.* Semiotika i Informatika, 12:12–21, 1979. En russe.

[75] Vershinin, K. et A. Paskevich: *ForTheL — the language of formal theories.* International Journal of Information Theories and Applications, 7(3):120–126, 2000.

[76] Weidenbach, C., R. Schmidt, T. Hillenbrand, R. Rusev et D. Topic: *SPASS Version 3.0.* Dans Pfenning, F. (réd.): *Automated Deduction: 21st International Conference, CADE-21*, t. 4603 de *Lecture Notes in Computer Science*, p. 514–520. Springer, 2007.

[77] Wenzel, M.: *Isabelle/Isar — a generic framework for human-readable proof documents.* Dans Matuszewski, R. et A. Zalewska (réds.): *From Insight to Proof — Festschrift in Honour of Andrzej Trybulec*, t. 10(23) de *Studies in Logic, Grammar, and Rhetoric*, p. 277–298. University of Białystok, 2007.

[78] Wiedijk, F. (réd.): *The Seventeen Provers of the World*, t. 3600 de *Lecture Notes in Computer Science*. Springer, 2006.

[79] Wiedijk, F. et J. Zwanenburg: *First Order Logic with Domain Conditions.* Dans Basin, D. et B. Wolff (réds.): *Theorem Proving in Higher Order Logics: 16th International Conference TPHOL 2003*, t. 2758 de *Lecture Notes in Computer Science*, p. 221–237. Springer, 2003.

A Théorème de Tarski-Knaster

Dans cette annexe nous donnons une formalisation en ForTheL vérifiée du théorème de Tarski-Knaster sur les pointes fixes d'une fonction monotone sur un treillis complet.

Le texte ci-dessous illustre certaines extensions de ForTheL qui ne sont pas décrites au chapitre 1 : les groupes de tokens abrégés (`[set/-s]` est équivalent à `[set/sets]`), les variables predéclarées (qui sont inscrites aux certaines notions par défaut), les chaînes de relations symboliques (e.g. `x <= y <= z`) et les définitions d'ensembles en compréhension.

```
[set/-s] [subset/-s] [element/-s] [belong/-s]

Signature SetSort. A set is a notion.
Signature ElmSort. An element is a notion.

Let S,T denote sets.
Let x,y,z,u,v,w denote elements.

Signature EOfElem. An element of S is an element.

Let x << S denote (x is an element of S).
Let x belongs to S denote (x is an element of S).

Definition DefEmpty.  S is empty iff S has no elements.

Definition DefSub.
  A subset of S is a set T such that every (x << T) belongs to S.

Let S [= T denote S is a subset of T.

Signature LessRel.  x <= y is an atom.

Axiom ARefl. x <= x.
Axiom ASymm. x <= y <= x => x = y.
Axiom Trans. x <= y <= z => x <= z.

[bound/-s] [supremum/-s] [infimum/-s] [lattice/-s]

Definition DefLB.   Let S be a subset of T.
  A lower bound of S in T is an element u of T
    such that for every (x << S) u <= x.

Definition DefUB.   Let S be a subset of T.
  An upper bound of S in T is an element u of T
    such that for every (x << S) x <= u.

Definition DefInf.  Let S be a subset of T.
  An infimum of S in T is an element u of T
```

```
    such that u is a lower bound of S in T and
      for every lower bound v of S in T we have v <= u.

Definition DefSup.  Let S be a subset of T.
  A supremum of S in T is an element u of T
    such that u is a upper bound of S in T and
      for every upper bound v of S in T we have u <= v.

Lemma SupUn.  Let S be a subset of T.
  Let u,v be supremums of S in T. Then u = v.

Lemma InfUn.  Let S be a subset of T.
  Let u,v be infimums of S in T. Then u = v.

Definition DefCLat.   A complete lattice is a set S such that
  every subset of S has an infimum in S and a supremum in S.

[function/-s] [point/-s]

Signature ConMap.   A function is a notion.

Let f stand for a function.

Signature DomSort.  Dom f is a set.
Signature RanSort.  Ran f is a set.

Definition DefDom.  f is on S iff Dom f = Ran f = S.

Signature ImgSort.  Let x belong to Dom f.
  f(x) is an element of Ran f.

Definition DefFix.  A fixed point of f is
  an element x of Dom f such that f(x) = x.

Definition DefMonot.  f is monotone iff
  for all (x,y << Dom f)  x <= y  =>  f(x) <= f(y).

Theorem Tarski.
  Let U be a complete lattice and f be a monotone function on U.
  Let S be the set of fixed points of f.
  S is a complete lattice.
Proof.
  Let T be a subset of S.
  Let us show that T has a supremum in S.
    Take P = { x << U | f(x) <= x and x is an upper bound of T in U }.
    Take an infimum p of P in U.
    f(p) is a lower bound of P in U and an upper bound of T in U.
    Hence p is a fixed point of f and a supremum of T in S.
  end.
  Let us show that T has an infimum in S.
    Take Q = { x << U | x <= f(x) and x is a lower bound of T in U }.
    Take a supremum q of Q in U.
    f(q) is an upper bound of Q in U and a lower bound of T in U.
    Hence q is a fixed point of f and an infimum of T in S.
  end.
qed.
```

B Lemme de Newman

Dans cette annexe nous donnons une formalisation en ForTheL vérifiée du lemme de Newman sur la confluence des systèmes de réécriture de termes. Les extensions de la syntaxe dans le texte ci-dessous sont les mêmes que dans l'annexe précédente.

```
[element/-s] [system/-s] [reduct/-s]

Signature ElmSort.  An element is a notion.
Signature RelSort.  A rewriting system is a notion.

Let a,b,c,d,u,v,w,x,y,z denote elements.
Let R,S,T denote rewriting systems.

Signature Reduct.   A reduct of x in R is an element.

Let x -R> y stand for y is a reduct of x in R.

Signature WFOrd.    x -<- y is a relation.

Signature TCbr.     x -R+> y is a relation.

Definition TCDef.   x -R+> y <=> x -R> y \/ exists z : x -R> z -R+> y.

Axiom TCTrans.      x -R+> y -R+> z => x -R+> z.

Definition TCRDef.  x -R*> y <=> x = y \/ x -R+> y.

Lemma TCRTrans.     x -R*> y -R*> z => x -R*> z.

Definition CRDef.   R is confluent iff
  for all a,b,c  such that a -R*> b,c
  there exists d such that b,c -R*> d.

Definition WCRDef.  R is locally confluent iff
  for all a,b,c  such that a -R> b,c
  there exists d such that b,c -R*> d.

Definition Termin.  R is terminating iff for all a,b
  a -R+> b => b -<- a.

Definition NFRDef.  A normal form of x in R is an element y
                such that x -R*> y and y has no reducts in R.

Lemma TermNF.   Let R be a terminating rewriting system.
                Every element x has a normal form in R.
Proof by induction. Obvious.
```

```
Lemma Newman.
  Any locally confluent terminating rewriting system is confluent.
Proof.
  Let R be locally confluent and terminating.
  Let us demonstrate by induction that for all a,b,c
  such that a -R*> b,c there exists d such that b,c -R*> d.

    Assume that a -R+> b,c.

    Take u such that a -R> u -R*> b.
    Take v such that a -R> v -R*> c.
    Take w such that u,v -R*> w.
    Take a normal form d of w in R.

    b -R*> d. Indeed take x such that b,d -R*> x.
    c -R*> d. Indeed take y such that c,d -R*> y.
  end.
qed.
```

C Théorème des restes chinois

Plus bas nous donnons une formalisation en ForTheL vérifiée du théorème des restes chinois dans un anneau commutatif unitaire et du théorème de Bézout dans un anneau euclidien. Les extensions de la syntaxe dans le texte ci-dessous sont les mêmes que dans l'annexe A.

```
[element/-s]

Signature ElmSort. An element is a notion.

Let a,b,c,x,y,z,u,v,w denote elements.

Signature SortsC.  0 is an element.
Signature SortsC.  1 is an element.
Signature SortsU.  -x is an element.
Signature SortsB.  x + y is an element.
Signature SortsB.  x * y is an element.

Let x is nonzero stand for x != 0.
Let x - y stand for x + -y.

Axiom AddComm.   x + y = y + x.
Axiom AddAsso.  (x + y) + z = x + (y + z).
Axiom AddZero.   x + 0 = x = 0 + x.
Axiom AddInvr.   x + -x = 0 = -x + x.

Axiom MulComm.   x * y = y * x.
Axiom MulAsso.  (x * y) * z = x * (y * z).
Axiom MulUnit.   x * 1 = x = 1 * x.

Axiom AMDistr.  x * (y + z) = (x * y) + (x * z) and
                (y + z) * x = (y * x) + (z * x).

Lemma MulMnOne. -1 * x = -x = x * -1.
Lemma MulZero.  x * 0 = 0 = 0 * x.

Axiom Cancel.   If x * y = 0 then x = 0 or y = 0.

Axiom UnNeZr.   1 != 0.

[set/-s] [belong/-s]

Signature SetSort.  A set is a notion.

Let X,Y,Z,U,V,W denote sets.

Signature EOfElem.  An element of W is an element.
```

```
Let x << W denote (x is an element of W).
Let x belongs to W denote (x is an element of W).

Axiom SetEq.  If every element of X belongs to Y
    and every element of Y belongs to X then X = Y.

Definition DefSSum. X + Y = { x + y | x << X and y << Y }.

Definition DefSInt. X ** Y = { h | h << X and h << Y }.

[ideal/-s]

Definition DefIdeal.
  An ideal is a set X such that for every x << X
    forall y << X (x + y) << X and
    forall z (z * x) << X.

Let I,J denote ideals.

Lemma IdeSum.   I + J is an ideal.
Proof.
  Let x, y belong to I + J and z be an element.
  Take k << I and l << J such that x = k + l.
  Take m << I and n << J such that y = m + n.
  k + m belongs to I and l + n belongs to J.
  z * k belongs to I and z * l belongs to J.
  We have x + y = (k + m) + (l + n) (by AddComm, AddAsso).
  We have z * x = (z * k) + (z * l) (by AMDistr).
  Hence (x + y), (z * x) belong to I + J.
qed.

Lemma IdeInt.   I ** J is an ideal.

Definition DefMod.  x = y (mod I)  iff  x - y << I.

Theorem ChineseRemainder.
  Suppose that every element belongs to I + J.
  Let x, y be elements.
  There exists an element w such that
    w = x (mod I) and w = y (mod J).
Proof.
  Take a << I and b << J such that a + b = 1.
  Take w = (y * a) + (x * b).

  Let us show that w - x belongs to I.
    w - x = (y * a) + ((x * b) - x) (by AddAsso).
    x * (b - 1) belongs to I.
  end.

  Let us show that w - y belongs to J.
    w - y = (x * b) + ((y * a) - y) (by AddAsso,AddComm).
    y * (a - 1) belongs to J.
  end.
qed.
```

```
[number/-s]

Signature NatSort.  A natural number is a notion.

Signature EucSort.  Let x be a nonzero element.
  |x| is a natural number.

Signature NatLess.  Let i,j be natural numbers.
  i -<- j is a relation.

Axiom Division.
  Let x, y be elements and y != 0.
  There exist elements q,r such that
    x = (q * y) + r and (r != 0 => |r| -<- |y|).

[divisor/-s] [divide/-s]

Definition DefDiv.
  x divides y  iff  for some z (x * z = y).

Let x | y stand for x divides y.
Let x is divided by y stand for y | x.

Definition DefDvs.
  A divisor of x is an element that divides x.

Definition DefGCD.
  A gcd of x and y is a common divisor c of x and y
    such that any common divisor of x and y divides c.

Definition DefRel.
  x, y are relatively prime iff 1 is a gcd of x and y.

Definition DefPrIdeal.
  <c> is { c * x | x is an element }.

Lemma PrIdeal.  <c> is an ideal.
Proof.
  Let x,y belong to <c> and z be an element.
  Take an element u such that c * u = x.
  Take an element v such that c * v = y.
  We have x + y = c * (u + v) (by AMDistr).
  We have z * x = c * (u * z) (by MulComm,MulAsso).
  Hence (x + y), (z * x) belong to <c>.
qed.

Theorem Bezout.  Let a, b be elements.
  Assume that a is nonzero or b is nonzero.
  Let c be a gcd of a and b. Then c belongs to <a> + <b>.
Proof.
  Take an ideal I equal to <a> + <b>.
  We have 0,a << <a> and 0,b << <b>.
  Hence there exists a nonzero element of <a> + <b>.

  Take a nonzero u << I
    such that for no nonzero v << I (|v| -<- |u|).
```

```
  Indeed we can show by induction on |w| that
    for every nonzero w << I there exists nonzero u << I
      such that for no nonzero v << I (|v| -<- |u|).
    Obvious.

  u is a common divisor of a and b.
  proof.
    Assume the contrary.
    For some elements x,y  u = (a * x) + (b * y).
    proof.
      Take k << <a> and l << <b> such that u = k + l.
      Hence the thesis.
    end.

    Case u does not divide a.
      Take elements q,r such that a = (q * u) + r
        and (r = 0 \/ |r| -<- |u|).
      r is nonzero.
      - (q * u) belongs to I.
      a belongs to I.
      r = - (q * u) + a.
      Hence r belongs to I.
    end.

    Case u does not divide b.
      Take elements q,r such that b = (q * u) + r
        and (r = 0 \/ |r| -<- |u|).
      r is nonzero.
      - (q * u) belongs to I.
      b belongs to I.
      r = - (q * u) + b.
      Hence r belongs to I.
    end.
  end.

  Hence u divides c.
  Hence the thesis.
qed.
```

D Une vérification dans SAD

Plus bas nous donnons et commentons le protocole de vérification dans le système SAD du Texte sur l'ensemble vide (voir la figure 1.2 dans la section 1.5).

```
[ForTheL] empty.ftl: parsing successful
[Reason] empty.ftl: verification started
```

Il est temps pour les premiers commentaires. Comme nous voyons, chaque message du système est «signé» par le module responsable (l'architecture de SAD est représentée à la figure 3.1), avec une exception : les messages du vérificateur sont signés `[Reason]` ainsi que les messages du raisonneur lui-même. Cela est une trace des jeunes jours de SAD, quand la distinction entre les deux était assez vague.

```
[ForTheL] line 3: hypothesis SetSort.
[ForTheL] line 4: forall v0 ((DHD :: aSet(v0)) implies truth).
```

Le vérificateur tombe sur les premières sections du texte. Le genre de la section et (pour les phrases) son image-formule sont affichés. Ces messages sont signés par le nom du langage correspondant : `[ForTheL]`, `[FOL]` ou `[TPTP]`. Le numérotage de lignes correspond à celui de la figure 1.2. Notez que les lignes 1 et 2 contiennent des instructions pour l'analyseur syntaxique (les groupes de tokens) et sont ainsi invisibles au vérificateur.

Les axiomes, définitions et extensions de signature sont tous étiquetés «`hypothesis`». Les théorèmes et lemmes sont tous étiquetés «`conjecture`». L'annotation `DHD ::` marque l'atome de tête dans une définition ou une extension de signature. Comme nous le voyons par l'image-formule de la phrase, la section `SetSort` est en fait une extension de signature.

```
[ForTheL] line 5: hypothesis ElmSort.
[ForTheL] line 6: assume aSet(S).
[Reason] trivial check: aSet(S) vs SetSort
[ForTheL] line 7: forall v0 ((DHD :: aElementOf(v0,S)) implies truth).
```

Dans la ligne 6, le système rencontre la première occurrence qui n'est pas un atome ou un terme de tête dans une définition ou une extension de signature. Cette occurrence doit alors être contrôlée pour la correction ontologique. Comme l'extension de signature `SetSort` pour le nom-classe `set` n'a pas de conditions à satisfaire, le contrôle est trivial.

```
[ForTheL] line 10: hypothesis DefSubset.
[ForTheL] line 11: assume aSet(S).
[Reason] trivial check: aSet(S) vs SetSort
[ForTheL] line 12: forall v0 ((DHD :: aSubsetOf(v0,S)) iff (aSet(v0) and
                    forall v1 (aElementOf(v1,v0) implies aElementOf(v1,S)))).
[Reason] trivial check: aSet(u0) vs SetSort
[Reason] trivial check: aElementOf(u1,u0) vs ElmSort
[Reason] trivial check: aElementOf(u1,S) vs ElmSort
```

Nous rencontrons ensuite la définition de la notion de sous-ensemble. La partie droite de l'équivalence doit être contrôlée pour la correction ontologique. Bien que l'extension de signature `ElmSort` pour le nom-classe `element of _` a une condition (notamment : l'argument doit

être un ensemble), le contrôle est toujours trivial, grâce aux évidences pour les occurrences correspondantes de `v0` et `S` qui affirment que ces termes sont effectivement des ensembles.

Comme toutes les conditions de définitions et extensions de signature dans le texte considéré sont aussi simples, toutes les vérifications ontologiques seront triviales.

Une remarque technique : les noms de variables bornées peuvent changer dans des contextes différents. Dans notre implantation, nous employons les indices de de Bruijn pour la représentation interne des variables bornées et leurs noms à afficher sont générés de façon ad hoc.

```
[ForTheL] line 13: hypothesis DefEmpty.
[ForTheL] line 14: assume aSet(S).
[Reason] trivial check: aSet(S) vs SetSort
[ForTheL] line 15: ((DHD :: isEmpty(S)) iff not exists v0 aElementOf(v0,S)).
[Reason] trivial check: aElementOf(u0,S) vs ElmSort
[ForTheL] line 16: hypothesis ExEmpty.
[ForTheL] line 17: exists v0 (aSet(v0) and isEmpty(v0)).
[Reason] trivial check: aSet(u0) vs SetSort
[Reason] trivial check: isEmpty(u0) vs DefEmpty
```

La partie préliminaire du texte est finie et le théorème principal commence.

```
[ForTheL] line 18: conjecture.
[ForTheL] line 19: assume aSet(S).
[Reason] trivial check: aSet(S) vs SetSort
[ForTheL] line 20: (forall v0 (aSet(v0) implies aSubsetOf(S,v0)) iff
                                                                isEmpty(S)).
[Reason] trivial check: aSet(u0) vs SetSort
[Reason] trivial check: aSubsetOf(S,u0) vs DefSubset
[Reason] trivial check: isEmpty(S) vs DefEmpty
[Reason] current thesis (mot): (forall v0 (aSet(v0) implies aSubsetOf(S,v0))
                                                          iff isEmpty(S))
```

Au début de la démonstration, la thèse est précisément la proposition à démontrer. La mention `(mot)` signifie que nous n'avons rencontrée dans la vérification courante aucune assomption non-motivée.

```
[ForTheL] line 22: (isEmpty(S) implies forall v0 (aSet(v0) implies
                                                      aSubsetOf(S,v0))).
[Reason] trivial check: isEmpty(S) vs DefEmpty
[Reason] trivial check: aSet(u0) vs SetSort
[Reason] trivial check: aSubsetOf(S,u0) vs DefSubset
```

Bien que la phrase de la ligne 22 soit une affirmation, le vérificateur n'essaie pas de la prouver avant de vérifier la démonstration donnée dans le texte.

```
[ForTheL] line 23: (isEmpty(S) implies forall v0 (aElementOf(v0,S) implies
                            forall v1 (aSet(v1) implies aElementOf(v0,v1)))).
[Reason] trivial check: isEmpty(S) vs DefEmpty
[Reason] trivial check: aElementOf(u0,S) vs ElmSort
[Reason] trivial check: aSet(u1) vs SetSort
[Reason] trivial check: aElementOf(u0,u1) vs ElmSort
[Reason] line 23: goal: if S is empty then any element of S belongs to any
                                                                    set T.
[Reason] subgoal: (isEmpty(S) implies forall v0 (aElementOf(v0,S) implies
                             forall v1 (aSet(v1) implies aElementOf(v0,v1))))
[Reason] prover task:
  forall v0 ((DHD :: aSet(v0)) implies truth)
  forall v0 (aSet(v0) implies forall v1 ((DHD :: aElementOf(v1,v0)) implies
```

```
                                                                    truth))
  forall v0 (aSet(v0) implies forall v1 (aSubsetOf(v1,v0) implies aSet(v1)))
  exists v0 (aSet(v0) and isEmpty(v0))
  aSet(S)
  |- (isEmpty(S) implies forall v0 (aElementOf(v0,S) implies forall v1
                                        (aSet(v1) implies aElementOf(v0,v1))))
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] timeout in 980 ms
[Moses] inference steps: 37940 - born nodes: 111648 - depth bound: 272
```

Voici la première tâche de preuve pour le raisonneur. Le raisonneur filtre les prémisses, ne simplifie pas le but (puisque il n'y a rien à simplifier) et envoie le séquent au démonstrateur qui n'arrive pas à le démontrer. La raison en est que notre procédure de filtrage «bloque» toutes les définitions du texte et, pour ce pas de démonstration particulier, les définitions sont nécessaires. Considérez le séquent donné au démonstrateur et notez comment l'image-formule de la définition `DefSubset` est transformée en une «règle de sorte» : tout sous-ensemble d'un ensemble est un ensemble. La définition `DefEmpty` est enlevée entièrement (dans ce cas la procédure de filtrage actuelle est plus stricte que ce qui est décrit dans la section 3.3.1).

```
[Reason] unfold: (isEmpty(S),23)
[Reason] subgoal: ((not exists v0 aElementOf(v0,S) implies not isEmpty(S)) or
               forall v0 (aElementOf(v0,S) implies forall v1 (aSet(v1) implies
                                                    aElementOf(v0,v1))))
[Reason] prover task:
  forall v0 ((DHD :: aSet(v0)) implies truth)
  forall v0 (aSet(v0) implies forall v1 ((DHD :: aElementOf(v1,v0)) implies
                                                                    truth))
  forall v0 (aSet(v0) implies forall v1 (aSubsetOf(v1,v0) implies aSet(v1)))
  exists v0 (aSet(v0) and isEmpty(v0))
  aSet(S)
  |- ((not exists v0 aElementOf(v0,S) implies not isEmpty(S)) or forall v0
    (aElementOf(v0,S) implies forall v1 (aSet(v1) implies aElementOf(v0,v1))))
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] proved in 0 ms - proof tree nodes: 2 - proof tree depth: 2
[Moses] inference steps: 4 - born nodes: 4 - depth bound: 1
[Reason] ...proved
```

Le raisonneur applique (d'une manière conservatrice) la définition de l'ensemble vide dans la formule-but et envoie le séquent obtenu au démonstrateur. Cette fois, la démonstration est trouvée instantanément.

```
[Reason] line 22: goal: If S is empty then S is a subset of every set.
[Reason] subgoal: (isEmpty(S) implies forall v0 (aSet(v0) implies
                                                      aSubsetOf(S,v0)))
[Reason] prover task:
  forall v0 ((DHD :: aSet(v0)) implies truth)
  forall v0 (aSet(v0) implies forall v1 ((DHD :: aElementOf(v1,v0)) implies
                                                                    truth))
  forall v0 (aSet(v0) implies forall v1 (aSubsetOf(v1,v0) implies aSet(v1)))
  exists v0 (aSet(v0) and isEmpty(v0))
  aSet(S)
  (isEmpty(S) implies forall v0 (aElementOf(v0,S) implies forall v1 (aSet(v1)
                                                  implies aElementOf(v0,v1))))
  |- (isEmpty(S) implies forall v0 (aSet(v0) implies aSubsetOf(S,v0)))
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] timeout in 990 ms
```

```
[Moses] inference steps: 61251 - born nodes: 212658 - depth bound: 246
[Reason] unfold: (isEmpty(S),22) (aSubsetOf(S,v0),22) (isEmpty(S),23)
[Reason] subgoal: ((not exists v0 aElementOf(v0,S) implies not isEmpty(S)) or
             forall v0 (aSet(v0) implies (forall v1 (aElementOf(v1,S) implies
                                      aElementOf(v1,v0)) or aSubsetOf(S,v0))))
[Reason] prover task:
  forall v0 ((DHD :: aSet(v0)) implies truth)
  forall v0 (aSet(v0) implies forall v1 ((DHD :: aElementOf(v1,v0)) implies
                                                                    truth))
  forall v0 (aSet(v0) implies forall v1 (aSubsetOf(v1,v0) implies aSet(v1)))
  exists v0 (aSet(v0) and isEmpty(v0))
  aSet(S)
  ((not exists v0 aElementOf(v0,S) or isEmpty(S)) implies forall v0
    (aElementOf(v0,S) implies forall v1 (aSet(v1) implies aElementOf(v0,v1))))
  |- ((not exists v0 aElementOf(v0,S) implies not isEmpty(S)) or forall v0
     (aSet(v0) implies (forall v1 (aElementOf(v1,S) implies aElementOf(v1,v0))
                                                    or aSubsetOf(S,v0))))
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] proved in 0 ms - proof tree nodes: 2 - proof tree depth: 2
[Moses] inference steps: 4 - born nodes: 4 - depth bound: 1
[Reason] ...proved
[Reason] current thesis (mot): (forall v0 (aSet(v0) implies aSubsetOf(S,v0))
                                                      implies isEmpty(S))
```

Une fois la démonstration courte sur la ligne 23 vérifiée, le système recule pour vérifier l'affirmation sur la ligne 22. La première tentative est manquée à cause d'une définition nécessaire bloquée au filtrage. Le raisonneur applique alors toutes les définitions applicables et la deuxième tentative est réussie. Comme on vient de démontrer une moitié de l'équivalence cherchée, la thèse courante se réduit à l'autre moitié.

```
[ForTheL] line 24: assume forall v0 (aSet(v0) implies aSubsetOf(S,v0)).
[Reason] trivial check: aSet(u0) vs SetSort
[Reason] trivial check: aSubsetOf(S,u0) vs DefSubset
[Reason] current thesis (mot): isEmpty(S)
```

Une réduction de thèse de plus, cette fois due à une assomption motivée.

```
[ForTheL] line 25: isEmpty(S).
[Reason] trivial check: isEmpty(S) vs DefEmpty
[Reason] current thesis (mot): isEmpty(S)
[ForTheL] line 26: assume aElementOf(z,S).
[Reason] trivial check: aElementOf(z,S) vs ElmSort
[Reason] current thesis (mot): contradiction
```

Ici, une sous-démonstration est initiée pour démontrer que S est vide. La première assomption la tourne en une démonstration par contradiction, ce qui n'échappe pas à la motivatrice.

```
[ForTheL] line 27: (aSet(E) and isEmpty(E)).
[Reason] trivial check: aSet(E) vs SetSort
[Reason] trivial check: isEmpty(E) vs DefEmpty
[Reason] line 27: goal: Take an empty set E.
[Reason] subgoal: exists v0 (aSet(v0) and isEmpty(v0))
[Reason] prover task:
  forall v0 ((DHD :: aSet(v0)) implies truth)
  forall v0 (aSet(v0) implies forall v1 ((DHD :: aElementOf(v1,v0)) implies
                                                                    truth))
  forall v0 (aSet(v0) implies forall v1 (aSubsetOf(v1,v0) implies aSet(v1)))
```

```
  exists v0 (aSet(v0) and isEmpty(v0))
  aSet(S)
  (isEmpty(S) implies forall v0 (aSet(v0) implies aSubsetOf(S,v0)))
  forall v0 (aSet(v0) implies aSubsetOf(S,v0))
  aElementOf(z,S)
  |- exists v0 (aSet(v0) and isEmpty(v0))
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] proved in 0 ms - proof tree nodes: 4 - proof tree depth: 2
[Moses] inference steps: 3 - born nodes: 4 - depth bound: 1
[Reason] ...proved
[Reason] current thesis (mot): contradiction
[ForTheL] line 28: aElementOf(z,E).
[Reason] trivial check: aElementOf(z,E) vs ElmSort
[Reason] line 28: goal: z is an element of E (by DefSubset).
[Reason] subgoal: aElementOf(z,E)
[Reason] prover task:
  forall v0 (aSet(v0) implies forall v1 ((DHD :: aSubsetOf(v1,v0)) iff
      (aSet(v1) and forall v2 (aElementOf(v2,v1) implies aElementOf(v2,v0)))))
  aSet(S)
  (isEmpty(S) implies forall v0 (aSet(v0) implies aSubsetOf(S,v0)))
  forall v0 (aSet(v0) implies aSubsetOf(S,v0))
  aElementOf(z,S)
  (aSet(E) and isEmpty(E))
  |- aElementOf(z,E)
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] proved in 0 ms - proof tree nodes: 9 - proof tree depth: 4
[Moses] inference steps: 9 - born nodes: 15 - depth bound: 3
[Reason] ...proved
```

Dans la dernière tâche de preuve, l'image-formule de la définition `DefSubset` est envoyée au démonstrateur telle quelle, mais la définition `DefEmpty` et l'axiome `ExEmpty` sont bloqués, à cause des références explicites données dans le texte en ForTheL.

```
[Reason] current thesis (mot): contradiction
[ForTheL] line 29: contradiction.
[Reason] line 29: goal: We have a contradiction (by DefEmpty).
[Reason] subgoal: contradiction
[Reason] prover task:
  forall v0 (aSet(v0) implies forall v1 (aSubsetOf(v1,v0) implies aSet(v1)))
  forall v0 (aSet(v0) implies ((DHD :: isEmpty(v0)) iff not exists v1
                                                          aElementOf(v1,v0)))
  aSet(S)
  (isEmpty(S) implies forall v0 (aSet(v0) implies aSubsetOf(S,v0)))
  forall v0 (aSet(v0) implies aSubsetOf(S,v0))
  aElementOf(z,S)
  (aSet(E) and isEmpty(E))
  aElementOf(z,E)
  |- contradiction
[Moses] launch (time limit: 1 sec, initial db: 1, final db: 0)
[Moses] proved in 0 ms - proof tree nodes: 6 - proof tree depth: 3
[Moses] inference steps: 5 - born nodes: 7 - depth bound: 2
[Reason] ...proved
[Reason] current thesis (mot): truth
```

Après la dernière affirmation dans la sous-démonstration, la thèse courante est entièrement démontrée et donc réduite à $\top$.

```
[Reason] line 25: goal: S is empty.
[Reason] trivial subgoal: truth
[Reason] current thesis (mot): truth
```

C'est pourquoi, quand nous revenons pour vérifier notre affirmation initiale sur la ligne 25, il ne reste plus rien pour le raisonneur à démontrer.

```
[Reason] line 20: goal: S is a subset of every set iff S is empty.
[Reason] trivial subgoal: truth
[Reason] empty.ftl: verification successful
```

De même pour l'affirmation du théorème.

```
[Main] sections 24 - goals 7 - subgoals 9 - trivial 2 - proved 5
[Main] symbols 31 - checks 27 - trivial 27 - proved 0 - unfolds 4
[Main] parser 00:00.03 - reason 00:00.01 - prover 00:02.01/00:00.00
[Main] total 00:02.06
```

Le protocole finit avec quelques données statistiques. Nous apprenons que sept buts dans le texte (donc sept tâches pour le raisonneur) ont produit neuf sous-tâches dont deux ont été triviales et cinq ont été arrangées par le démonstrateur. Ainsi, deux sous-tâches ont été ratées entraînant l'application de définitions (dans quatre occurrences en somme) et un autre appel au démonstrateur. Le contrôle ontologique a détecté 31 symbole non-logique pour lesquelles 27 vérifications ontologiques ont été effectuées. Quatre symboles de tête dans deux extensions de signature et deux définitions font la différence. La vérification à pris au total deux secondes et six centièmes.

E Statistiques

Plus bas nous considérons six formalisations en ForTheL et leur vérification dans SAD avec cinq démonstrateurs automatiques différents.

Formalisation	sections	buts	symboles[1]	...triviaux[2]
Lemme de Newman (l'annexe B)	55	12	116	116
Théorème de Tarski-Knaster (l'annexe A)	89	13	120	106
Théorème des restes chinois (l'annexe C)	162	52	378	377
Inégalité de Cauchy-Schwarz	164	52	521	516
La racine carrée d'un nombre premier[3]	193	77	482	480
Théorème de Ramsey infini	326	99	886	785

Les données ci-dessous sont obtenues sur une machine Intel Xeon 5110 1.60 GHz Dual-Core (3214.19 BogoMIPS) avec 2 Go de mémoire vive. L'exécution est mono-fil (*single-threaded*). Le temps d'exécution d'un démonstrateur est limité à 3 secondes par une tâche.

Démonstrateur	buts	...prouvées	sous-tâches	...triviales	...ratées	temps
Lemme de Newman						
E 0.999	12	12	41	0	18 (44%)	49s
Moses	12	5 (42%)	49	0	33 (67%)	1m 34s
Otter 3.3f	12	6 (50%)	47	0	32 (68%)	3m 22s
SPASS 3.0	12	12	41	0	18 (44%)	37s
Vampire 7.44	12	12	41	0	18 (44%)	18s
Théorème de Tarski-Knaster						
E 0.999	13	13	91	19 (21%)	40 (49%)	34s
Moses	13	7 (54%)	81	19 (23%)	42 (52%)	1m 37s
Otter 3.3f	13	13	91	19 (21%)	40 (49%)	1m 40s
SPASS 3.0	13	13	91	19 (21%)	40 (49%)	40s
Vampire 7.44	13	13	91	19 (21%)	40 (49%)	41s
Théorème des restes chinois						
E 0.999	52	43 (83%)	114	9 (8%)	61 (54%)	3m 12s
Moses	52	26 (50%)	173	9 (5%)	140 (81%)	7m 33s
Otter 3.3f	52	22 (42%)	187	9 (5%)	158 (84%)	9m 03s
SPASS 3.0	52	52	103	9 (9%)	39 (38%)	2m 07s
Vampire 7.44	52	40 (77%)	139	9 (6%)	87 (63%)	4m 57s
Inégalité de Cauchy-Schwarz						
E 0.999	52	49 (94%)	76	30 (39%)	9 (12%)	40s
Moses	52	25 (48%)	94	29 (31%)	53 (56%)	2m 44s
Otter 3.3f	52	27 (52%)	92	29 (32%)	51 (55%)	2m 57s
SPASS 3.0	52	52	73	30 (41%)	3 (4%)	15s
Vampire 7.44	52	41 (79%)	84	30 (36%)	25 (28%)	1m 30s

[1] exposés au contrôle ontologique.
[2] vérifiés au premier passage du contrôle ontologique (la section 3.2).
[3] dans une formulation un peu différente de celle dans la section 1.6.

La racine carrée d'un nombre premier						
E 0.999	77	73 (95%)	159	15 (9%)	51 (32%)	2m 51s
Moses	27[4]	9 (33%)	58	6 (10%)	47 (81%)	2m 25s
Otter 3.3f	27[4]	13 (48%)	54	6 (11%)	38 (70%)	2m 22s
SPASS 3.0	77	77	157	15 (10%)	45 (29%)	2m 49s
Vampire 7.44	77	70 (90%)	157	15 (10%)	59 (38%)	3m 42s
Théorème de Ramsey infini						
E 0.999	99	98 (99%)	400	136 (34%)	74 (19%)	4m 31s
Moses	53[4]	28 (53%)	303	81 (27%)	129 (43%)	8m 23s
Otter 3.3f	48[4]	32 (67%)	266	74 (28%)	100 (38%)	5m 26s
SPASS 3.0	99	99	395	136 (34%)	68 (17%)	4m 16s
Vampire 7.44	99	99	393	136 (35%)	66 (17%)	4m 15s

Remarquons que SPASS est capable de démontrer tous les buts dans tous les textes partiellement grâce à ses qualités de démonstrateur (qui ont été mentionnées au chapitre 3), mais aussi grâce au fait que nos textes sont adaptés pour se vérifier avec SPASS que nous utilisons comme le démonstrateur principal (ce qui est entièrement dû à ses qualités de démonstrateur). Il est d'autant plus intéressant que les résultats de Vampire et E ne sont pas trop éloignés.

Notons aussi que le taux des sous-tâches ratées montre (si on laisse de côté les buts non-démontrés et les différences entre les démonstrateurs) à quel degré les définitions participent dans la démonstration.

[4]La vérification s'est terminée après l'échec du contrôle ontologique.