Rapport de stage

Propriétés sur les contraintes de la logique BI

Sylvain Chiron
Master 1 Informatique
Université de Bordeaux

Sous la direction de

Dominique Larchey-Wendling
Chargé de recherche CNRS/LORIA

Du 27 mai au 31 juillet 2024

Résumé

La logique Bunched Implication (BI) est une logique dite de ressources. Elle forme le noyau de la logique de séparation, utilisée par exemple dans la vérification des propriétés du tas mémoire. La recherche de preuves et contre-modèles dans BI, vue sous l'angle de la méthode des tableaux sémantiques, génère des contraintes. Ces contraintes sont des éléments d'une relation binaire sur des multi-ensembles, elles-mêmes closes pour des règles de dérivation.

L'objectif final du stage était de développer un outil pour manipuler ces contraintes sous la forme de graphes de ressources, dans l'assistant de preuve Coq. Les étapes pour en arriver à concevoir cet outil étaient cependant trop nombreuses, de sorte que le stage a consisté exclusivement à implanter dans Coq des preuves de propriétés sur les contraintes de BI, sur les indications de Dominique Larchey-Wendling.

Il a été nécessaire de prouver entre autres des propriétés sur les permutations et sur les clôtures de contraintes, et de prouver aussi des équations sur les extensions, pour en arriver à prouver des propriétés sur les contraintes de BI. Une extension est l'ajout d'une nouvelle contrainte à un ensemble clos existant, suivi de la réapplication des règles de dérivation.

Table des matières

Chapitre 1

Contexte

Le stage s'est déroulé au LORIA, Laboratoire lorrain de recherche en informatique et ses applications, sous la direction de Dominique Larchey-Wendling, chargé de recherche CNRS/LORIA.

Le LORIA est une unité mixte de recherche commune à plusieurs établissements : le CNRS, l'Université de Lorraine et l'INRIA. Un peu plus de 200 chercheurs et enseignants-chercheurs y travaillent, et environ 175 doctorants. Le LORIA est un des plus grands laboratoires de la Lorraine.

Il comporte 29 équipes structurées en 5 départements :

— Algorithmique, calcul, image et géométrie ;
— Méthodes formelles ;
— Réseaux, systèmes et services ;
— Traitement automatique des langues et des connaissances ;
— Systèmes complexes, intelligence artificielle et robotique.

C'est au sein du département Méthodes formelles que se trouve l'équipe TYPES au sein de laquelle j'ai travaillé avec Dominique Larchey-Wendling. Cette équipe est dirigée par Didier Galmiche. Elle se concentre sur la logique, la théorie de la preuve et la programmation de preuve.

Chapitre 2

Sujet

2.1 L'apparition de contraintes dans la méthode des tableaux

La méthode des tableaux sémantiques permet de prouver ou de réfuter une formule logique, en développant un arbre à l'aide de règles, deux par opérateur de la logique : une pour le cas où la formule est supposée invalide et une pour le cas où elle est supposée valide. Dans le cas de la logique Bunched Implication, ces règles font intervenir des contraintes.

De telles contraintes apparaissent également, de façon un peu plus restreinte, avec la logique intuitionniste. Ce sont elles qui distinguent la logique intuitionniste de la logique classique lors de l'utilisation de la méthode des tableaux. Elles permettent d'invalider le tiers exclu, qui est la règle admise par la logique classique et rejetée par la logique intuitionniste.

Toutes les règles pour les opérateurs de la logique BI sont présentées dans [2], tableaux 2 et 3. Voici les règles pour deux opérateurs :

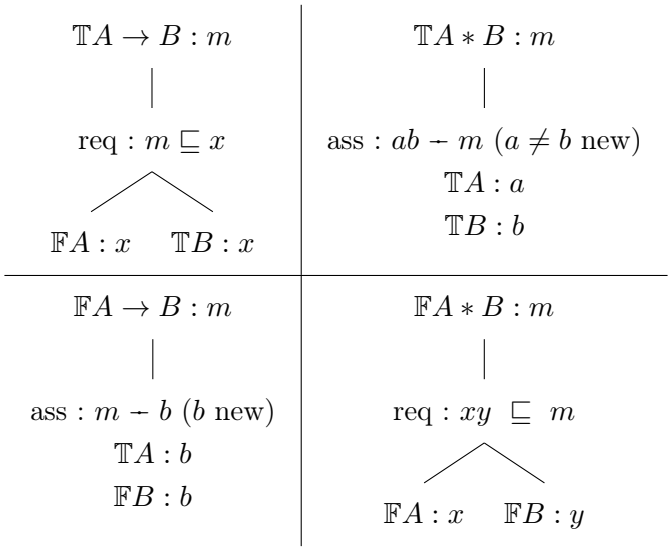


Tableau 2.1 – Les règles d'expansion de la méthode des tableaux pour les opérateurs implication et étoile de la logique BI.

— « $\mathbb{T}$ » et « $\mathbb{F}$ » signifient respectivement que la formule qui suit est supposée valide ou invalide.

— « ass » signifie qu'on ajoute une nouvelle contrainte à la relation $\sqsubseteq$, avec de nouvelles lettres a et b nouvelles pour la relation. Une telle contrainte est un couple représenté par le séparateur « $\multimap$ ».

— « req » signifie qu'on utilise une contrainte déjà existante dans la relation $\sqsubseteq$.

La règle pour l'implication est commune avec la logique intuitionniste. Dans le cas de la logique BI, les mots (multi-ensembles) opérandes des contraintes peuvent être composés de plusieurs lettres, mais dans le cas de la logique intuitionniste, ce ne peuvent être que de simples lettres.

Pour déterminer avec la méthode des tableaux si une formule est valide ou non, on procède comme suit : on commence par supposer que la formule est invalide, et on lui associe un mot composé d'une lettre. Cela donne une formule signée (par validité ou invalidité) avec un label (le mot associé), que nous appellerons FSL. Puis on développe l'arbre en appliquant successivement les règles sur l'opérateur le plus extérieur de chaque FSL. On doit chercher à clore les différentes branches de l'arbre. Si toutes les branches sont closes, alors la formule est valide. Si au contraire, une branche ne peut plus être développée alors qu'elle ne comporte pas de contradiction, alors on a trouvé un contre-modèle qui montre que la formule est invalide.

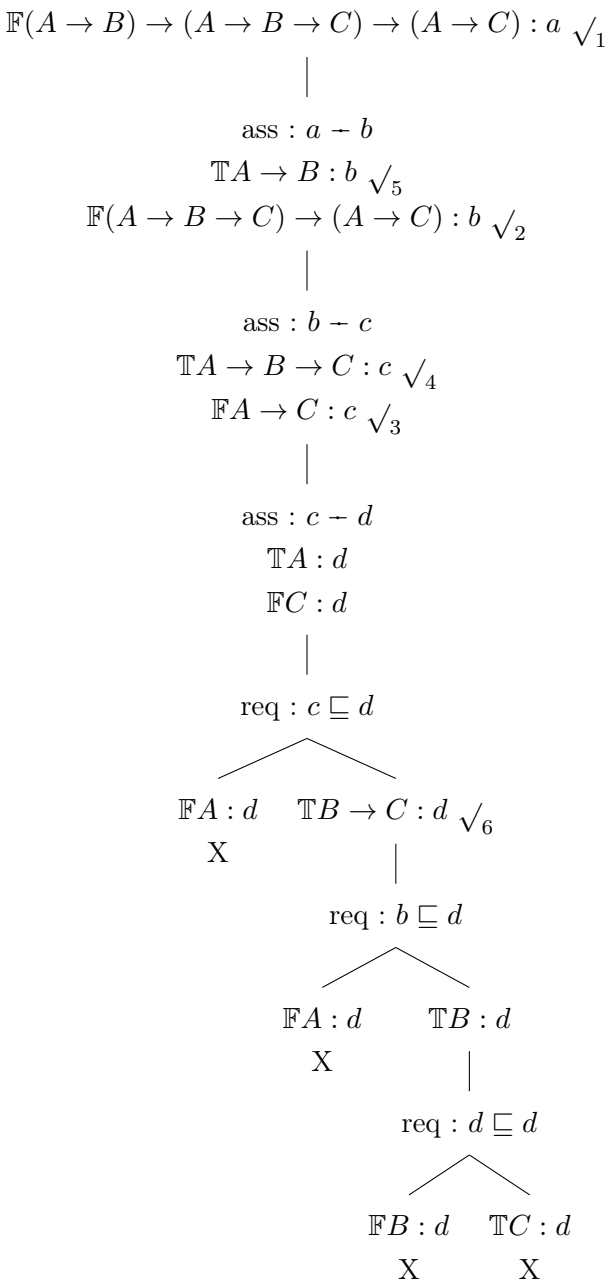
Les conditions de clôture de branche en logique BI sont données dans [2], définition 5.2. En logique intuitionniste, une condition de moins s'applique, ce qui en laisse trois, que voici :

1. $\mathbb{T}X : m$, $\mathbb{F}X : n$ et $m \sqsubseteq n$
2. $\mathbb{T}\bot : m$
3. $\mathbb{F}\top : m$

À titre d'exemple, voici l'une des preuves que j'ai travaillée, montrant que la formule

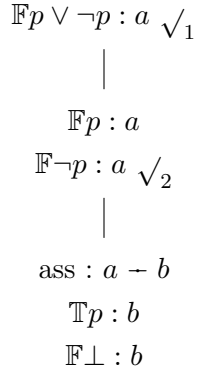
$$(A \to B) \to (A \to B \to C) \to (A \to C)$$

est vraie en logique intuitionniste :



Chaque branche voit à la fois l'une des propositions A, B ou C être vraie et fausse dans le monde d.

Voici un autre exemple, exhibant un contre-modèle pour la formule du tiers-exclu :



On rappelle que $\neg p := p \to \bot$.

Le contre-modèle est p faux dans le monde a et vrai dans le monde b.

C'est ainsi qu'apparaissent des contraintes lors de l'utilisation de la méthode des tableaux sémantiques. C'est ce type de contraintes qui ont été le cœur du sujet de mon stage.

2.2 Les différentes étapes de preuve

L'objectif final de mon stage était de fournir une représentation de graphes de ressources permettant de manipuler les contraintes de la logique BI, et de mécaniser et prouver avec Coq une correspondance avec les ensembles de contraintes. Cet objectif ambitieux n'a pas été complètement atteint ; j'ai cependant prouvé des propriétés sur le système de contraintes.

La manipulation de ces contraintes conduit à concevoir le concept de *Partial Monoidal Order* (PMO) : une relation de contraintes close pour une série de six règles de dérivation, qui sont celles utilisées pour la logique BI.

Pour en venir à prouver des propriétés sur les contraintes de BI, nous sommes passés par une série d'étapes.

1. Preuves sur les répétitions de liste et les permutations
2. Étude de propriétés des PMO
3. Preuve d'équations sur les extensions de PMO
4. Preuve de lemmes sur les alphabets par rapport aux contraintes
5. Preuve de propriétés du système de contraintes de BI

Le code source Coq est disponible à [1].

Chapitre 3

Les étapes

3.1 Répétitions de liste et permutations

Le but de cette partie était de me familiariser doucement à l'écriture de preuve en Ltac (le langage de tactiques de Coq) en prouvant de petits lemmes sur les répétitions de listes et les permutations, qui me seront utiles pour des preuves ultérieures.

Ces lemmes sont utiles, car pour représenter les multi-ensembles dits mots (les éléments des contraintes), le choix proposé par Dominique Larchey-Wendling a été d'utiliser des listes à permutation près, bien plus faciles à représenter et manipuler dans Coq. Il a donc été nécessaire d'avoir à disposition des lemmes sur les permutations pour pouvoir manipuler les mots, ainsi que des morphismes pour la réécriture. Les répétitions de liste, quant à elles, apparaissent avec les PMO du fait des règles de dérivation, à partir de certaines formes d'extensions.

Nous utiliserons de nombreuses lettres (le plus souvent l, m et k) pour représenter les listes, et les lettres a ou b pour représenter les lettres qui composent de telles listes.

3.1.1 Répétitions de liste

Il s'agit de cette définition :

$$l^0 = \epsilon$$
$$l^{1+n} = l \cdot l^n$$

J'ai dû prouver des propriétés telles que :

$$a \in l^n \to a \in l$$

$$(\forall b \in l, a = b) \to l = a^{|l|}$$

$$l \cdot r = a^n \to l = a^{|l|} \wedge r = a^{|r|} \wedge n = |l| + |r|$$

La deuxième se prouve par induction sur l, et la preuve de la troisième utilise la deuxième.

Dans Coq, nous avons utilisé la notation `↑` (par ex. l^n s'écrit `l↑n`) pour représenter la répétition de liste.

3.1.2 Permutations

La relation de permutation $\sim$ est définie inductivement à partir de quatre règles qui m'ont été fournies :

$$\text{nil} : \frac{}{\epsilon \sim \epsilon}$$
$$\text{skip} : \frac{l \sim m}{a \cdot l \sim a \cdot m}$$
$$\text{swap} : \frac{}{a \cdot b \cdot l \sim b \cdot a \cdot l}$$
$$\text{trans} : \frac{k \sim l \quad l \sim m}{k \sim m}$$

Cette définition existe dans la bibliothèque standard, cependant pour mon apprentissage nous sommes repartis de zéro avec les permutations.

Dans Coq, nous avons utilisé la notation `~p` pour représenter les permutations. Ainsi, $l \sim m$ s'écrit `l ~p m`.

J'ai dû prouver des propriétés telles que :

— la commutativité de la concaténation :

$$l \cdot m \sim m \cdot l$$

(se prouve par induction imbriquée sur l puis sur m) ;

— le morphisme de l'appartenance :

$$l \sim m \to \forall a, a \in l \to a \in m$$

(se prouve par induction sur la preuve de permutation) ;

— le morphisme des longueurs :

$$l \sim m \to |l| = |m|$$

(se prouve par induction sur la preuve de permutation) ;

— l'inversion (c'est-à-dire l'élimination d'une permutation non arbitraire) de certaines listes simples :

$$l \sim \epsilon \leftrightarrow l = \epsilon$$

$$l \sim a \leftrightarrow l = a$$

$$l \sim a^n \leftrightarrow l = a^n$$

(utilise le lemme précédent sur les longueurs) ;

— la régularité (*cancellativity*), qui est le contraire de la règle skip :

$$a \cdot l \sim a \cdot m \to l \sim m$$

Pour prouver cette dernière propriété, il a été nécessaire de prouver d'abord quelques lemmes sur une définition inductive plus générale, l'ajout d'une lettre dans une liste par rapport à une autre :

$$\text{stop} : \frac{}{Add\ a\ l\ a{\cdot}l}$$
$$\text{next} : \frac{Add\ a\ l\ m}{Add\ a\ b{\cdot}l\ b{\cdot}m}$$

3.1.3 Éléments de Coq appris

Au fil de ces exercices, j'ai découvert et utilisé de nombreuses nouvelles tactiques de Coq :

— la fameuse `induction` ;
— le `destruct` sur d'autres constructions que le « ou » logique et `False` ;
— le `intros` avec schéma de destructions (qui permet de faire directement des `destruct` et des `apply`) ;
— `constructor` ;
— `simpl` ;
— `symmetry` ;
— `reflexivity` ;
— `f_equal` ;
— `subst` ;
— `rewrite`, qui permet la réécriture en fonction des morphismes démontrés ;
— `discriminate` ;
— `transitivity`.

Ces tactiques sont pour la plupart documentées dans [3], mais je n'ai pas trouvé cette documentation facile à comprendre.

J'ai également appris à lire des définitions inductives, et à ajouter des règles pour la tactique `auto`.

De plus, j'ai commencé à explorer la bibliothèque standard de Coq.

3.2 Propriétés des PMO

3.2.1 Définition du Partial Monoidal Order

Un PMO est une relation de contraintes close pour une série de six règles de dérivation, que voici :

$$\text{epsilon} : \frac{}{\epsilon \sqsubseteq \epsilon}$$

$$\text{left} : \frac{l \sqsubseteq m}{l \sqsubseteq l}$$

$$\text{right} : \frac{l \sqsubseteq m}{m \sqsubseteq m}$$

$$\text{subword} : \frac{l{\cdot}m \sqsubseteq l{\cdot}m}{l \sqsubseteq l}$$

$$\text{compos} : \frac{k{\cdot}m \sqsubseteq k{\cdot}m \quad l \sqsubseteq m}{k{\cdot}l \sqsubseteq k{\cdot}m}$$

$$\text{trans} : \frac{k \sqsubseteq l \quad l \sqsubseteq m}{k \sqsubseteq m}$$

J'ai dû coder ces six règles en Coq, ainsi que trois autres :

$$\text{incr} : \frac{l \leftharpoonup m}{l \sqsubseteq m}$$

$$\text{perm left} : \frac{l \sim m \quad l \sqsubseteq k}{m \sqsubseteq k}$$

$$\text{perm right} : \frac{l \sim m \quad k \sqsubseteq l}{k \sqsubseteq m}$$

Un type inductif `pmo` est défini dans Coq avec ces neuf règles. Il prend en premier paramètre une relation notée $\rightharpoonup$. Avec ce premier paramètre seulement, on obtient une autre relation, qui est $\rightharpoonup$ close, notée $\sqsubseteq$. Le type de `pmo` est :

$$(list\ X \times list\ X \mapsto Prop) \mapsto (list\ X \times list\ X \mapsto Prop)$$

Une relation prend ainsi deux $list\ X$ en paramètres, c'est-à-dire des listes de X, où X est un type arbitraire qui est celui de nos lettres. Avec ces deux paramètres, elle renvoie une proposition décrivant si les deux listes font partie de la relation ou non.

Comme défini par la règle incr, `pmo` reprend les règles de la relation qu'on lui donne en paramètre, et en ajoute d'autres en fonction des six règles et des deux règles de permutation.

3.2.2 Preuves de propriétés

J'ai prouvé quelques propriétés générales sur les PMO. Soient R et R' deux relations sur les listes, dont les PMO respectifs sont $\sqsubseteq_R$ et $\sqsubseteq_{R'}$. Les PMO sont :

— une clôture :

$$(\forall (l, m), l \sqsubseteq_R m \to l \sqsubseteq_{R'} m) \leftrightarrow (\forall (l, m), l\ R\ m \to l \sqsubseteq_{R'} m)$$

— une opération monotone :

$$(\forall (l, m), l\ R\ m \to l\ R'\ m) \to (\forall (l, m), l \sqsubseteq_R m \to l \sqsubseteq_{R'} m)$$

— une opération *idempotent* : répéter plusieurs fois l'opération sur une relation est inutile.

Ensuite, j'ai prouvé le PMO de trois relations :

— $\{a \rightharpoonup b\}$ donne $\{\epsilon \sqsubseteq \epsilon, a \sqsubseteq a, b \sqsubseteq b, a \sqsubseteq b\}$;

— $\{a \rightharpoonup \epsilon\}$ donne $\{a^n \sqsubseteq a^m \mid n \geq m\}$;

— $\{ab \rightharpoonup a\}$ donne $\{b^n \sqsubseteq b^n\} \cup \{ab^n \sqsubseteq ab^m \mid n \geq m\}$ (à permutation près).

Cela se fait en montrant la double inclusion entre le PMO et l'ensemble de contraintes. Pour montrer que le PMO est inclus, il faut faire une induction sur le type PMO. Pour montrer que l'ensemble de contraintes est inclus, il faut faire une induction sur n.

3.2.3 Éléments de Coq appris

Durant cette étape, j'ai appris comment :

— décrire une logique en Coq ;

— écrire des tactiques personnalisées (combinaisons de plusieurs tactiques) ;

— déclarer des morphismes ;

— résoudre des problèmes arithmétiques avec la tactique `lia`.

Les morphismes permettent la réécriture dans les expressions. Déclarer et prouver les morphismes m'a permis d'obtenir les règles de dérivation suivantes :

$$\frac{m \sim m' \quad l \cdot m \sim k}{l \cdot m' \sim k}$$

$$\frac{l \sim l' \quad a \in l}{a \in l'}$$

$$\frac{l \sim l' \quad m \sim m' \quad l \sqsubseteq m}{l' \sqsubseteq m'}$$

3.3 Équations

À partir de cette étape, on cherche à se restreindre aux contraintes que l'on peut rencontrer dans le cas de la preuve d'un séquent en logique BI avec la méthode des tableaux. Cela se traduit par des extensions à la relation ne contenant que $\epsilon \sqsubseteq \epsilon$, qui sont décrites dans la section 3.5. Pour simplifier les preuves sur ces extensions, on commence par prouver des équations sur des extensions plus générales.

Dès cette étape, les démonstrations sont devenues bien plus difficiles, car les hypothèses sont devenues bien plus nombreuses et les constructeurs des types inductifs bien plus complexes.

Cette étape est liée à la suivante. Chronologiquement, les deux ont été réalisées simultanément, car les lemmes sur les alphabets ont été nécessaires à la preuve des équations.

Il y a d'abord trois variables et trois hypothèses disponibles pour les trois équations. Les trois variables sont une relation $\sqsubseteq$, un mot (liste) m qui est dans la relation et un mot α qui n'y est pas. Les trois hypothèses sont :

— la relation $\sqsubseteq$ est un PMO ;
— on dispose de la contrainte $m \sqsubseteq m$;
— aucune lettre de α n'est dans $\sqsubseteq$, autrement dit l'alphabet de α est complètement distinct de celui de $\sqsubseteq$, ce que l'on écrit : $A_\alpha \cap A_\sqsubseteq = \varnothing$.

Les trois équations consistent à déterminer quel est le PMO d'une extension. Les voici.

$$pmo(\sqsubseteq \cup \{\alpha{\cdot}m \rightharpoonup \alpha{\cdot}m\}) = \sqsubseteq \cup \{\delta{\cdot}x \rightharpoonup \delta{\cdot}y \mid \epsilon \neq \delta \preccurlyeq \alpha \wedge x \sqsubseteq y(\preccurlyeq \bullet \sqsubseteq)m\}$$

$$\begin{aligned} pmo(\sqsubseteq \cup \{m \rightharpoonup \alpha\}) = \sqsubseteq \cup \{&\delta{\cdot}e \rightharpoonup \delta{\cdot}f \mid \epsilon \neq \delta \preccurlyeq \alpha \wedge e \sqsubseteq f(\preccurlyeq \bullet \sqsubseteq)\epsilon\} \\ &\cup \{x \rightharpoonup \alpha{\cdot}e \mid x \sqsubseteq m{\cdot}e \wedge e(\preccurlyeq \bullet \sqsubseteq)\epsilon\} \end{aligned}$$

$$pmo(\sqsubseteq \cup \{\alpha \rightharpoonup m\}) = \sqsubseteq \cup \{\delta{\cdot}\alpha^i{\cdot}x \rightharpoonup \delta{\cdot}y \mid \exists j, \delta \preccurlyeq \alpha^j \wedge m^i{\cdot}x \sqsubseteq y \wedge m^j{\cdot}y \sqsubseteq m^j{\cdot}y\}$$

Il y a deux nouveaux opérateurs dans ces équations.

— L'opérateur $\preccurlyeq$ est l'opérateur « sous-mot ». Par exemple, $\delta \preccurlyeq \alpha$ signifie que l'on peut obtenir δ en retirant des lettres à α (sans considération d'un quelconque ordre des lettres).
— L'opérateur $\bullet$ est l'opérateur de composition de relations. Par exemple, $y(\preccurlyeq \bullet \sqsubseteq)m$ signifie :

$$\exists y', y \preccurlyeq y' \wedge y' \sqsubseteq m$$

Ces preuves ont bien entendu été faites en montrant l'inclusion dans chaque sens.

À chaque fois, ce qui a été de loin la grande étape a été de montrer que la partie droite de l'équation est un PMO. Cela a nécessité de faire de grandes inductions à neuf cas, parfois subdivisés en encore plus de cas.

Voici quelques lemmes intermédiaires (ayant un PMO $\sqsubseteq$ comme hypothèse) que j'ai dû prouver pour prouver ces équations :

$$\frac{e(\preccurlyeq \bullet \sqsubseteq)\epsilon \quad l \sqsubseteq m}{l{\cdot}e \sqsubseteq m{\cdot}e}$$

$$\frac{l \sqsubseteq m \quad m^i \sqsubseteq m^i}{l^i \sqsubseteq m^i}$$

$$\frac{l((\preccurlyeq \bullet \sqsubseteq) \bullet (\preccurlyeq \bullet \sqsubseteq))m}{l(\preccurlyeq \bullet \sqsubseteq)m}$$

3.4 Lemmes sur les alphabets

Les lemmes de cette section ont été requis par l'étape précédente et je les ai prouvés dans le même temps. En pratique, j'ai écrit un long fichier contenant des lemmes sur les alphabets, certains étant assez complexes, c'est pourquoi je distingue cette section de la précédente.

Ces lemmes reposent sur plusieurs nouvelles définitions.

— La relation « est un sous-mot » est représentée par l'opérateur $\preccurlyeq$. Définition :

$$l \preccurlyeq m \leftrightarrow \exists k, m \sim l{\cdot}k$$

— Une lettre peut être dans l'alphabet d'un mot :

$$a \in A_l \leftrightarrow a \in l$$

— Une lettre peut être dans l'alphabet d'une relation :

$$a \in A_{\sqsubseteq} \leftrightarrow \exists (l, m), l \sqsubseteq m \wedge (a \in A_l \vee a \in A_m)$$

Notons qu'en Coq, l'appartenance à un alphabet est représenté par un prédicat. Par exemple, $a \in A$ est codé par `P a` (`P` est de type $X \mapsto Prop$), tandis que $a \in l$ s'écrit `In a l` (`In` est de type $X \mapsto list\ X \mapsto Prop$). Lors de l'utilisation du lemme, Coq associe `In a l` à `P a`.

Avec ces nouvelles définitions, j'ai prouvé des lemmes tels que :

— $|l| = |m| \to l \preccurlyeq m \to l \sim m$;

— $l^n \preccurlyeq l \to n \leq 1 \wedge l = \epsilon$;

— deux morphismes (le deuxième à sens unique) :

$$\frac{l \sim l' \quad m \sim m' \quad l \preccurlyeq m}{l' \preccurlyeq m'}$$

$$\frac{l \preccurlyeq m \quad a \in l}{a \in m}$$

— le découpage de deux mots concaténés permutables avec deux mots concaténés (ne requiert pas les nouvelles définitions) :

$$u{\cdot}v \sim l{\cdot}r \to \exists (l_u, l_v, r_u, r_v),\ l \sim l_u{\cdot}l_v \wedge u \sim l_u{\cdot}r_u \\ \wedge\ r \sim r_u{\cdot}r_v \wedge v \sim l_v{\cdot}r_v$$

— le découpage d'un mot dont toutes les lettres se trouvent dans deux alphabets :

$$\forall a \in w, a \in P \vee a \in Q \to \exists (w_0, w_1), w \sim w_0{\cdot}w_1 \wedge (\forall a \in w_0, a \in P) \\ \wedge\ (\forall a \in w_1, a \in Q)$$

— le découpage de deux mots concaténés permutables avec deux mots concaténés, sachant que ces mots ont toutes leurs lettres dans deux alphabets (ce que l'on représente avec $\subset$) :

$$l_0 \subset P \to m_0 \subset P \to l_1 \subset Q \to m_1 \subset Q \ \to P \cap Q = \varnothing \\ \to l_0 \sim m_0 \wedge l_1 \sim m_1$$

J'ai plusieurs autres lemmes de découpage de mots qui ressemblent à (ou sont des combinaisons de) ceux que j'ai donnés ci-dessus. Je n'ai malheureusement pas su comment factoriser.

3.5 Propriétés des contraintes de BI

Dans cette étape, on définit un prédicat inductif sur les relations qui détermine si elles ont pu ou non être générées par la logique Bunched Implication. Puis on établit que ces relations sont également acceptées par un prédicat plus général, qui accepte les extensions de nos équations de l'étape 3.3. Grâce à cette généralisation, les preuves (dans lesquelles on peut utiliser les équations) de propriétés sur nos extensions plus générales (dites « extensions libres ») permettent de dériver facilement les mêmes propriétés sur les relations générées par BI.

Il y a quatre variables et quatre hypothèses qui se retrouvent dans ces cas inductifs. Les variables sont : une relation $\sqsubseteq$, un mot m et deux lettres a et b. Les hypothèses sont :

— m est dans la relation : $m \sqsubseteq m$;
— $a \neq b$;
— $a, b \notin A_{\sqsubseteq}$;
— $\sqsubseteq$ est acceptée par le prédicat.

Dans le cas des extensions libres, il n'y a pas a et b mais α (comme dans les équations), et il y a une hypothèse de plus : α est sans carré (définition plus bas).

Les deux prédicats acceptent tout d'abord la relation ne contenant que $\epsilon \sqsubseteq \epsilon$, ainsi que les relations équivalentes à une relation déjà acceptée par le prédicat. Les autres relations générées par BI sont les suivantes, et sont généralisées par les extensions libres indiquées à côté :

BI	Extensions libres
$pmo(\sqsubseteq \cup \{ab \rightharpoonup m\})$ $(m \neq \epsilon)$	$pmo(\sqsubseteq \cup \{\alpha \rightharpoonup m\})$
$pmo(\sqsubseteq \cup \{a{\cdot}m \rightharpoonup b\})$	$pmo(pmo(\sqsubseteq \cup \{\alpha_1{\cdot}m_1 \rightharpoonup \alpha_1{\cdot}m_1\}) \cup \{m_2 \rightharpoonup \alpha_2\})$ avec $m_2 = \alpha_1{\cdot}m_1$
$pmo(\sqsubseteq \cup \{\epsilon \rightharpoonup m\})$ $(m \neq \epsilon)$	$pmo(\sqsubseteq \cup \{\alpha \rightharpoonup m\})$
$pmo(\sqsubseteq \cup \{m \rightharpoonup b\}$	$pmo(\sqsubseteq \cup \{m \rightharpoonup \alpha\})$

TABLEAU 3.1 – Les extensions générées par BI et les extensions libres correspondantes.

J'ai prouvé les propriétés suivantes sur toute relation $\sqsubseteq$ générée par ces extensions :

— elle est sans carré, c'est-à-dire que pour tout mot l de $\sqsubseteq$, on a :

$$\forall a, \neg(aa \preccurlyeq l)$$

— elle a un alphabet fini :

$$\exists l, \forall a, a \in l \leftrightarrow a \in A_{\sqsubseteq}$$

— elle est régulière (*cancellative*) :

$$\forall (u, v, k), \frac{k{\cdot}u \sqsubseteq k{\cdot}v}{u \sqsubseteq v}$$

— elle est antisymétrique :

$$\forall (l, k), \frac{l \sqsubseteq k \quad k \sqsubseteq l}{k \sim l}$$

Chapitre 4

Bilan

Durant ce stage, je me suis formé à la recherche de preuves et à leur formalisation avec Coq. De ce fait, le stage a répondu à mes attentes. J'ai découvert Coq plus en profondeur que lors de mes cours à l'université, je l'ai utilisé avec des objets plus complexes, et j'ai pu appréhender la difficulté d'un travail de recherche.

Le code produit fonctionne très bien, il peut être parcouru interactivement avec CoqIDE (l'environnement de développement que j'ai utilisé pendant le stage), et Coq est le garant de sa correction. Étant donné qu'il ne contient aucune clause `Admitted` et que Coq le compile très bien, toutes les propriétés qu'il contient sont prouvées. Le code est accessible à [1].

Je suis heureux que mon maitre de stage, Dominique Larchey-Wendling, m'ait grandement accompagné tout le long du stage. J'ai été très fréquemment bloqué lorsqu'il était nécessaire de concevoir une preuve, et mon maitre de stage m'a ainsi souvent donné les solutions à implémenter dans Coq, sans quoi je n'aurais pas du tout été aussi rapide.

Je suis parti un peu frustré de ne pas être allé au bout de la tâche qui m'était confiée, mais satisfait d'avoir compris et codé une série de définitions et de nombreuses preuves.

Références

[1] Dépôt du projet bi-constraints-graph. https://gitub.u-bordeaux.fr/sylchiron/bi-constraints-graph

[2] Larchey-Wendling, D., & Galmiche, D. (2009). Exploring the relation between Intuitionistic BI and Boolean BI: an unexpected embedding. *Mathematical Structures in Computer Science*, 19(3), 435-500.

[3] Tactics — Coq 8.19.2 documentation. https://coq.inria.fr/doc/V8.19.2/refman/proof-engine/tactics.html