Clôture par congruence et recherche de preuve modulo une théorie

Damien Rouhling

2013

Résumé

Ce document revient sur mon stage de recherche de licence 3, option Informatique Fondamentale. Nous présentons ici les problématiques de raisonnement automatisé qui ont été le contexte de ce stage, et les réponses apportées au problème de clôture par congruence dans le cadre d'un prouveur automatique travaillant dans un calcul des séquent polarisé et focalisé.

Table des matières

Introduction

Dans le cadre du stage de recherche de licence 3 option Informatique fondamentale, j'ai travaillé au Laboratoire d'Informatique de l'École polytechnique (LIX) sous la direction d'Assia Mahboubi et de Stéphane Lengrand. J'ai participé au développement de PSYCHE (Proof Search factorY for Collaborative HEuristics, [Psy]), un programme réalisant de la recherche de preuves.

La raisonnement assisté par ordinateur se scinde en deux domaines : la recherche et la vérification de preuve. La vérification se fait généralement via des outils interactifs (les assistants de preuve) qui permettent à l'utilisateur de donner les étapes d'une preuve et automatisent la vérification de la correction des étapes et de leur enchaînement. La recherche de preuve, automatisée, impose à l'outil d'exécuter une méthode qui construit une preuve s'il en existe une. On s'attend à ce que cette preuve, non nécessairement intuitive ni minimale, soit correcte. Diverses méthodes de recherche automatique existent, dont notamment la résolution et la méthode des tableaux ([Avr93],[Häh01]).

PSYCHE s'incrit dans la lignée des outils de raisonnement automatisé travaillant modulo une théorie. Il s'appuie sur un calcul des séquents et simule l'algorithme $DPLL(\mathcal{T})$ ([GL13],[FGLM13]). Parmi ces théories, l'une des plus élémentaires mais aussi des plus utiles est celle de l'égalité. Cette théorie, considérée seule, soulève déjà des questions d'algorithmique. On peut aussi la combiner avec d'autres théories : on parle alors de clôture par congruence modulo une théorie. Mon travail a été d'étudier un moyen d'effectuer ces combinaisons ([CCKL08], [Les11]) et de l'intégrer à PSYCHE.

PSYCHE travaille actuellement dans le calcul propositionnel. La seconde partie de mon stage s'est portée sur les questions posées par l'extension au premier ordre de PSYCHE. En logique du premier ordre, la construction d'une preuve peut nécessiter d'instancier une variable par un terme judicieux. Trouver ce terme est typiquement une chose que l'on pourrait demander à un être humain. Essayer tous les termes jusqu'à trouver celui qui convient est irréalisable en pratique. Aussi, les outils de recherche automatique retardent ce choix afin de connaître les contraintes qui pèsent sur la variable. Cet ajournement soulève quelques problèmes relatifs à tous les outils raisonnant modulo théories.

Dans ce document, nous faisons d'abord quelques rappels sur les outils de raisonnement automatisé afin de préciser la position de PSYCHE dans ce domaine. Nous détaillons ensuite le problème de clôture par congruence modulo une théorie et présentons un algorithme qui permet de le résoudre, qui provient des travaux de Conchon et al. ([CCKL08]), décrits plus en détail dans la thèse de Lescuyer ([Les11]). Nous étudions alors les problèmes soulevés par l'intégration de cet algorithme à PSYCHE. Enfin, nous envisageons quelques directions de prolongements possibles telles que certaines améliorations de l'implémentation et le passage au premier ordre.

1 Raisonnement automatisé

L'automatisation du raisonnement passe par l'implantation de procédures capables de se substituer à l'intuition humaine. Les premières réponses ont été sans cesses améliorées et dépassées à mesure que les problèmes traités sont devenus plus expressifs. Une telle évolution est décrite par Davis ([Dav01]). Nous faisons ici un point sur les méthodes actuelles de déduction automatique et observons l'intérêt de PSYCHE en ce domaine.

1.1 Déduction automatisée

La première famille d'outils de déduction automatisée est dédiée à la résolution d'instances du problème SAT. Ce problème est le suivant : étant donné des variables propositionnelles et une formule sous forme normale conjonctive (CNF) sur ses variables, existe-t-il une instanciation de ces variables qui satisfait la formule ? Une formule est en CNF si

elle est une conjonction de clauses, les clauses étant des disjonctions de littéraux (variables ou négations de variables). Les variables sont instanciées par vrai ou faux. Ce problème est NP-complet ([Coo71]).

De nombreuses recherches ont été réalisées sur les algorithmes de résolution de ce problème, menant à des heuristiques de plus en plus poussées et efficaces. L'intérêt de ce problème vient du fait que malgré la minimalité de la logique mise en jeu, des méthodes d'encodage permettent de traduire de nombreux problèmes en SAT. Ainsi, les outils efficaces de résolution SAT peuvent être utilisés plus ou moins efficacement sur d'autre problèmes en apparence plus compliqués. Cependant, l'encodage peut être trop coûteux pour être intéressant. En général, les solveurs SAT fournissent une instanciation des variables s'il en existe une.

Une extension naturelle de SAT est SAT Modulo Theories (SMT). Dans ce problème, les variables propositionnelles de SAT sont remplacées par les atomes d'une théorie. On peut résoudre une instance SMT par une communication entre un solveur SAT et une procédure de décision spécifique à la théorie. C'est le principe de l'algorithme $DPLL(\mathcal{T})$ ([NOT06]). Cette communication peut être faite à différents niveaux. Nous ne citons ici que le plus simple : chaque atome mis en jeu est nommé via une variable, on utilise le solveur SAT sur la formule ainsi traduite et, s'il trouve une instanciation, on vérifie sa cohérence du point de vue des atomes grâce à la procédure de décision. Si l'instanciation ne convient pas, on relance le solveur SAT sur la formule à laquelle on a ajouté une clause exprimant l'inconsistance d'un ensemble de littéraux, renvoyé par la procédure de décision.

Les solveurs SMT permettent alors de prouver des formules sans quantificateur. Il est en général nécessaire de calculer une forme CNF de la formule avant de la prouver. Cela peut être coûteux. Aussi, d'autres méthodes ont été développées afin d'éviter cette transformation. Les différents systèmes de calcul des séquents permettent notamment de construire des arbres de preuve sans nécessairement travailler sur une formule en forme CNF. En effet, il existe un lien fort entre la notion prouvabilité en calcul des séquents d'une part, et la notion de satisfiabilité d'une formule d'autre part. PSYCHE est un des outils travaillant en calcul des séquents.

1.2 PSYCHE

PSYCHE est une plate-forme modulaire permettant de faire de la recherche automatisée de preuves dans une théorie ([GL13]). Son implémentation actuelle travaille en logique propositionnelle.

Il est parfois nécessaire de faire des choix lors de la construction d'une preuve. En effet, si la forme de la formule peut imposer la démarche à suivre, il est aussi possible qu'elle laisse plusieurs possibilités qu'il faudra peut-être toutes tester afin de conserver la complétude de la recherche. Il est alors nécessaire de choisir l'ordre des différents essais. Dans ce contexte, la structure modulaire de PSYCHE est la suivante : un noyau, qui automatise toutes les étapes de la recherche ne nécessitant aucun choix, communique avec un (ou éventuellement plusieurs) plug-in. Le but de ce plug-in est de faire tous les choix et donc de définir la stratégie de recherche. L'intérêt de cette modularité est double.

En premier lieu, il n'est pas nécessaire de vérifier les preuves retournées par PSYCHE. En effet, un prouveur automatique est en général un programme de taille conséquente. Il est donc difficile d'avoir une entière confiance en un tel outil. Le noyau de PSYCHE est le garant de la correction de la preuve et de la complétude de la recherche. Il suit les consignes du plug-in mais travaille sur un type qui reste abstrait pour ce dernier. Ainsi, il fait automatiquement tout ce qui peut l'être sans faire de choix puis signale son état au plug-in afin de poursuivre la preuve si nécessaire. Le plug-in ne peut pas modifier le contenu de la preuve par lui-même donc tout ce qui est renvoyé est réellement produit par le noyau. Ce noyau, par sa petite taille, nous permet d'avoir une bonne confiance en

le résultat. On peut éventuellement renforcer cette confiance en faisant certifier le noyau à l'aide d'un assistant à la preuve.

La preuve est construite dans un calcul des séquents polarisé et focalisé (cf. Annexe A). Partant de la formule à démontrer, PSYCHE construit progressivement un arbre de preuve par des complétions partielles successives des branches. Les règles dites « asynchrones », qui peuvent être utilisées sans faire de choix, sont celles qui sont utilisées automatiquement par le noyau. C'est le plug-in qui signale qu'une règle « synchrone » ou « structurelle » peut être utilisée. Ce calcul des séquents permet de raisonner modulo théorie : la théorie $\mathcal{T}$ définit une notion de conséquence sémantique $\models_{\mathcal{T}}$. La notation $\Gamma \models_{\mathcal{T}} l$ exprime le fait que le littéral l est conséquence sémantique de l'ensemble de littéraux Γ. On note $\Gamma \models_{\mathcal{T}}$ pour exprimer que le faux est conséquence sémantique de Γ, autrement dit l'inconsistance sémantique de Γ.

La polarisation et la focalisation permettent de réduire la taille de l'espace de recherche via la simulation de méthodes de recherche connues pour leur efficacité (cf. par exemple [FGLM13]). Le second intérêt principal de la structure modulaire de PSYCHE est la possibilité de combiner ces méthodes de recherche. En effet, la construction de l'arbre de preuve se faisant par étapes, il est possible de changer de plug-in au cours de la recherche.

Ainsi, PSYCHE est un outil particulier de raisonnement automatique capable de combiner diverses méthodes afin de répondre à des problèmes relativement à une théorie. L'une d'entre elles est la théorie de l'égalité, qui nous mène à la question de la clôture par congruence modulo une théorie que nous décrivons en 2.

1.3 Contributions

Mon travail lors de ce stage a été d'adapter dans PSYCHE un algorithme de clôture par congruence modulo une théorie. La première étape a alors été de comprendre le fonctionnement de PSYCHE et l'algorithme de Lescuyer ([Les11], cf. 2.2 et 2.3). J'ai ensuite pu coder cet algorithme. Cependant, l'intégration dans PSYCHE a aussi nécessité l'ajout de mécanismes d'analyse de conflits (cf. 3.1). J'ai donc légèrement modifié l'algorithme afin d'ajouter ces mécanismes à l'aide des travaux de Nieuwenhuis et Oliveras ([NO05]), que j'ai adapté au contexte plus général du raisonnement modulo théories. J'ai alors pu tester la clôture par congruence modulo la théorie vide (cf. 2.1 pour les problèmes traités, 3.3 pour les aspects pratiques).

Je me suis enfin tourné vers l'étude plus théorique du passage au premier ordre dans PSYCHE (cf. 4.2). Des questions d'instanciation de variables, proches des problèmes d'unification ([BS01]), on été soulevées. Cette étude a conservé un aspect pratique car elle a pour but une implémentation.

2 Clôture par congruence modulo une théorie

La théorie de l'égalité traite des identités entre termes de la signature considérée. Ce problème devient plus intéressant lorsque certains symboles de fonction ont une valeur sémantique (du fait d'une combinaison avec une autre théorie, comme l'arithmétique par exemple).

2.1 Théorie de l'égalité

La théorie de l'égalité compte trois axiomes et un schéma d'axiomes et un unique symbole de prédicat. Les trois axiomes expriment le fait que l'égalité, représentée par le symbole de prédicat $=$, est une relation d'équivalence : elle est réflexive, transitive et symétrique. Ajouter le schéma d'axiomes fait de l'égalité une relation de congruence :

l'égalité se propage aux applications d'une même fonction à des arguments deux à deux égaux. Nous noterons $\mathcal{E}$ cette théorie.

Voici les axiomes de la théorie $\mathcal{E}$:

1. $\forall x,\ x = x$ (réflexivité)
2. $\forall x, y, z,\ x = y \wedge y = z \Rightarrow x = z$ (transitivité)
3. $\forall x, y\ x = y \Rightarrow y = x$ (symétrie)
4. Pour chaque symbole de fonction f d'arité n :

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n,\ (x_1 = y_1 \wedge \cdots \wedge x_n = y_n) \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

On parle alors de clôture par congruence pour désigner l'ajout de ce schéma d'axiomes à une théorie égalitaire.

La question principale que l'on peut se poser est la suivante :

> *Étant donné un ensemble d'égalités* Γ*, c'est-à-dire un ensemble d'atomes de la théorie de l'égalité, et un nouvel atome* A*, peut-on déduire* A *de* Γ *(ie) a-t-on* $\Gamma \vdash_{\mathcal{E}} A$ *?*

Exemple ([NO05]) : si l'on considère $\Gamma = \{f(b) = d, b = d, f(d) = a\}$, on souhaite pouvoir déduire l'atome $a = b$. En effet, comme $b = d$, on obtient par congruence $f(b) = f(d)$. Donc, par transitivité grâce aux deux autres atomes de Γ, $d = a$ et comme $b = d$ on peut conclure.

Ce problème est typique de la question algorithmique communément nommée union-find : on cherche à construire progressivement des classes d'équivalence par union, en effectuant au cours de la construction un certain nombre de recherches de représentants de classes d'équivalence (find).

On peut aussi considérer des diségalités dans Γ qui devient alors un ensemble de littéraux. À la question de la déduction de nouvelles égalités vient alors s'ajouter celle de la consistance de notre ensemble de littéraux.

Exemple ([Les11]) : l'ensemble $\Gamma = \{a = f(f(f(a))), a = f(f(f(f(f(a))))), a \neq f(a)\}$ est inconsistant relativement à la théorie de l'égalité.

On peut aisément utiliser des structures d'union-find afin de répondre à ce problème. Pour aller plus loin, on peut combiner la théorie $\mathcal{E}$ avec des théories égalitaires plus riches vérifiant certaines propriétés.

2.2 Théories considérées pour la combinaison

Lorsque l'on combine une théorie égalitaire avec la théorie $\mathcal{E}$, on pose toujours la même question, mais les façons d'y répondre vont changer. En effet, certaines égalités entre termes vraies par combinaison des deux théories ne le seraient pas dans $\mathcal{E}$ seule.

Exemple : avec la théorie de l'arithmétique linéaire rationnelle, si f est un symbole non interprété, c'est-à-dire ici un symbole qui n'est contraint par aucun axiome de l'arithmétique, $f(1 + 3) = f(2 + 2)$ n'est pas déductible par $\mathcal{E}$ car on ne pourrait que procéder par congruence à partir de $1 = 2$ et $3 = 2$, égalités fausses dans $\mathcal{E}$. Pourtant, par combinaison, comme on sait que $1 + 3 = 2 + 2$, on peut déduire l'égalité par congruence.

Cet exemple montre que la combinaison avec une théorie donne une *valeur sémantique* à certains symboles de fonction. C'est cette valeur qui contient par exemple la commutativité de l'addition. On peut étendre le concept de valeur sémantique aux termes. C'est l'idée suivie par Lescuyer au cours de sa thèse ([Les11]).

On peut voir l'association d'une valeur sémantique à un terme contenant des symboles non interprétés de la façon suivante : on descend dans le terme (vu comme un arbre), en remplaçant chaque symbole de fonction par sa valeur sémantique, tant que l'on ne rencontre pas de symbole non interprété. Si l'on trouve un tel symbole, on considère une valeur sémantique particulière, correspondant au sous-terme commençant à la position actuelle, qui devient une feuille de l'arbre en cours de construction. Lorsque l'on a fini le parcours de l'arbre, on peut en faire une évaluation. Dans la suite, nous verrons régulièrement une valeur sémantique comme un arbre et l'égalité entre valeurs sémantiques sera notée $\equiv$.

Exemple : en arithmétique, on peut associer un polynôme à chaque terme. L'addition devient l'addition de polynômes et les constantes ont pour valeur elles-mêmes. Ainsi, $1+2$ est transformé en l'arbre $1+2$ dont la valeur sémantique est 3. En revanche, $1+4\times 2+3f(3-5)$ est transformé en $1+4\times 2+3X$ où X est une valeur sémantique particulière associée au terme $f(3-5)$ et la valeur sémantique de cet arbre est le polynôme $9+3X$.

Dans sa thèse, Lescuyer propose un algorithme répondant à notre question pour la combinaison de $\mathcal{E}$ avec une théorie $\mathcal{X}$ vérifiant certains axiomes. Nous reprenons ici les définitions essentielles à la compréhension de l'algorithme, ainsi que les notations de Lescuyer ([Les11]).

Nous travaillons sur une signature Σ. La théorie $\mathcal{X}$ interprète un ensemble $\Sigma_{\mathcal{X}} \subseteq \Sigma$ de symboles de fonction. On note $\mathcal{R}$ l'ensemble des valeurs sémantiques des termes construits sur Σ. La théorie $\mathcal{X}$ doit disposer :

- d'une fonction $t \mapsto [t]$ qui à un terme construit sur Σ associe sa valeur sémantique ;
- d'une fonction $\mathit{leaves} : \mathcal{R} \to \mathcal{P}_f^*(\mathcal{R})$ qui à une valeur sémantique associe l'ensemble fini, non vide, de ses feuilles ;
- d'une valeur sémantique spéciale $\mathbf{1}$, feuille des termes complètement interprétés ;
- d'une fonction $\mathit{subst} : \mathcal{R}\times\mathcal{R}\times\mathcal{R} \to \mathcal{R}$ qui au triplet (p,P,r) associe la valeur $r\{p \mapsto P\}$, résultat de l'évaluation de l'arbre r dans lequel toutes les feuilles étiquetées par p ont été remplacées par l'arbre P ;
- d'une fonction $\mathit{solve} : \mathcal{R}\times\mathcal{R} \to (\mathcal{R}\times\mathcal{R})^{\top,\bot}$ qui à un couple de valeurs sémantiques associe une substitution $p \mapsto P$ (au sens de *subst*) qui unifie les deux valeurs si possible, qui renvoie $\top$ si les deux valeurs sont égales et $\bot$ si l'on ne peut trouver de substitution convenable.

Exemples : toujours avec la théorie de l'arithmétique linéaire rationnelle :

- dans l'exemple précédent, si $[f(3-5)] = X$, alors $[1+4\times 2+3f(3-5)] = 9+3X$
- $\mathit{leaves}(10) = \{\mathbf{1}\}$, $\mathit{leaves}(3\times X+2\times Y) = \{\mathbf{1}, X, Y\}$ et $\mathit{leaves}(X-Y) = \{X,Y\}$
- $\mathit{subst}(X, 3Y, X+Y) \equiv 4Y$
- $\mathit{solve}(1+X,3) = (X,2)$ mais $\mathit{solve}(1+2X, 2+2X) = \bot$

L'intuition que nous avons essayé de donner de ces fonctions est représentée par des axiomes qui garantissent leur bon comportement. Nous ne les reprenons pas ici. Nous avons présenté ici tous les outils nécessaire à l'algorithme répondant à notre question.

2.3 Algorithme $CC(\mathcal{X})$

L'algorithme de Lescuyer travaille sur des états $\langle \Theta \mid \Gamma \mid \Delta \mid N \mid \Phi \rangle$ où :

- Θ est l'ensemble des termes déjà rencontrés par l'algorithme ;
- Γ associe à chaque valeur sémantique les termes qui « utilisent » cette valeur, au sens où elle apparaît dans l'arbre représentant la valeur sémantique du terme, ou d'un terme de la même classe d'équivalence ;
- Δ maintient les classes d'équivalence entre valeurs sémantiques en associant à chaque valeur le représentant de sa classe ;

– N contient les diségalités rencontrées ;
– Φ est la séquence [1] des littéraux encore à traiter.

L'algorithme commence son calcul sur l'état $\langle \emptyset \mid r \mapsto \emptyset \mid r \mapsto r \mid \emptyset \mid \Phi \rangle$ où Φ est l'ensemble de littéraux à traiter et applique des règles d'inférences tant que Φ n'est pas vide. Ces règles sont données en Annexe B. L'algorithme traite aussi des requêtes : $t \overset{?}{=} u$ demande si l'ensemble des littéraux rencontrés implique l'égalité des termes t et u, et $t \overset{?}{\neq}$ fait de même avec leur différence. L'algorithme procède de cette manière : si les littéraux précédents entraînent nécessairement le littéral de la requête, l'algorithme passe à la suite, sinon il bloque.

Il y a une inconsistance si l'état final est $\langle \bot \mid \emptyset \rangle$. Sinon, si l'algorithme n'a pas bloqué, notre ensemble de littéraux est consistant. Si à partir d'un état initial $\langle \Theta \mid \Gamma \mid \Delta \mid N \mid \Phi \rangle$ on obtient une inconsistance, alors on note $\langle \Theta \mid \Gamma \mid \Delta \mid N \mid \Phi \rangle\uparrow$.

Nous illustrons ci-dessous les règles Congr et Add, qui sont les plus difficiles à comprendre.

Exemple ([Les11]) : Nous travaillons en arithmétique sur l'ensemble de littéraux $\Phi = \left\{ g\left(x+k\right) = a, x = 0, s = g\left(k\right), s \overset{?}{=} a \right\}$.

La première règle utilisée est Add qui va ajouter à Θ, progressivement, tous les sous-termes des deux termes du premier littéral. Comme on ne sait rien pour le moment, il n'y a rien à en déduire. C'est la règle Congr qui traite des égalités. Elle déduit des nouvelles égalités en appliquant le schéma de congruence. La règle Add fait de même avec les nouveaux termes découverts.

Lorsque $x = 0$ est traité par Congr, si $[x] = X$, on en déduit la substitution $X \mapsto 0$ que l'on applique à *toutes* les classes d'équivalence de Δ. Ainsi, $[x+k]$ et $[k]$ ont le même représentant. Lorsque l'on utilise plus tard la règle Add pour ajouter $g\left(k\right)$ à Θ, on déduit par congruence l'égalité $g\left(x+k\right) = g\left(k\right)$.

Ainsi, la règle Congr propage aux classes d'équivalences les substitutions provoquées par les égalités considérées : c'est l'équivalent de l'union dans les problèmes d'union-find. Elle ajoute aussi à Φ les nouvelles égalités déduites par congruence après substitution. Lorsque la règle Add ajoute un nouveau terme, on vérifie aussi si l'on peut trouver des égalités sur ce terme, obtenues par congruence.

Nous disposons ainsi d'un algorithme capable de traiter le problème de clôture par congruence modulo une théorie. Ceci est un premier pas vers l'élaboration d'une procédure de décision intégrable à PSYCHE.

3 Implémentations

L'algorithme que nous avons présenté nous permet de décider de la consistance d'un ensemble d'atomes n'impliquant que le symbole de prédicat de l'égalité. Afin de l'intégrer à PSYCHE, il a fallu construire les mécanismes d'analyse de conflit. Ceci a ensuite permis d'implanter la clôture par congruence modulo la théorie vide, c'est-à-dire lorsqu'aucun des symboles de fonction de la signature n'est interprété.

3.1 Analyse de conflit

PSYCHE fait intervenir la théorie sur laquelle on travaille par le biais des règles (*Init*) (cf. Annexe A). PSYCHE fait appel à des procédures de décision qui décident si un

1. L'algorithme est incrémental donc, en pratique, Φ est effectivement une séquence.

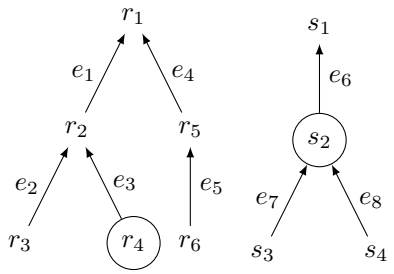
ensemble de littéraux est consistant, ou si un ensemble de littéraux a pour conséquence un autre littéral. Ces procédures doivent fournir des justifications, si possible petites : si l'ensemble est inconsistant, elles renvoient un sous-ensemble de littéraux déjà inconsistant, si le littéral objectif est bien impliqué par les autres, elles renvoient un sous-ensemble de littéraux impliquant ledit littéral. Quelques modifications ont été apportées à l'algorithme de Lescuyer afin de réaliser le calcul de ces justifications.

Nous avons tout d'abord distingué les égalités du problème initial de celles obtenues par congruence. Les deux types d'égalité sont traités identiquement par l'algorithme, mais leurs explications sont différentes. Si $f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)$ est apprise par congruence, son explication est l'union des explications de chaque $t_i = u_i$. En revanche, une égalité du problème initial est sa propre explication.

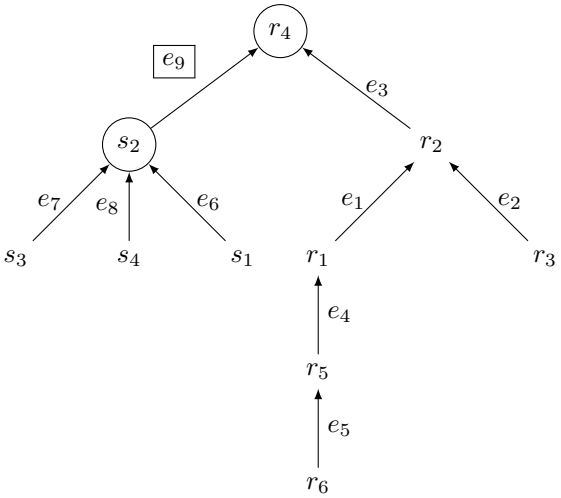
Cependant, d'autres types d'égalité existent : $1 + 1 = 2$ est assurée par la théorie et donc son explication est vide. De même, l'application d'une substitution via la règle Congr peut fusionner des classes d'équivalence qui ne contenaient pas les termes dont on traite l'égalité (cf. le dernier exemple de la section précédente). Il fallait un moyen de conserver les explications de chaque fusion de classes d'équivalence.

Au lieu de ne garder que le tableau des représentants (via Δ), nous avons alors aussi conservé les classes sous forme d'arbres, dont les arcs sont étiquetés par une égalité du problème initial, ou une congruence. L'idée vient des travaux de Nieuwenhuis et Oliveras ([NO05]). Les nœuds sont des valeurs sémantiques. Si deux classes sont fusionnées, c'est que deux valeurs sémantiques, une dans chaque arbre, ont changé de représentant et que leur nouveau représentant est identique. On inverse alors le chemin qui part de ces valeurs jusque la racine de leur arbre, et on ajoute un arc entre ces deux valeurs étiqueté par l'égalité qui a provoqué le changement de représentant (via une substitution).

Exemple : si l'égalité e_9 provoque, via les nœuds r_4 et s_2, la fusion de ces deux arbres :



on obtient l'arbre :



Ainsi, si on souhaite expliquer l'égalité de deux termes, on considère leurs valeurs sémantiques respectives, on cherche leur premier ancêtre commun dans l'arbre de leur classe d'équivalence et on fait l'union des explications des étiquettes des arcs sur les chemins de ces valeurs à cet ancêtre. On obtient alors bien, par exemple, l'explication vide pour $1 + 1 = 2$: ces termes ont même valeur sémantique et donc le premier ancêtre commun est cette valeur elle-même. Par conséquent, le chemin jusqu'à cet ancêtre est vide et l'explication aussi.

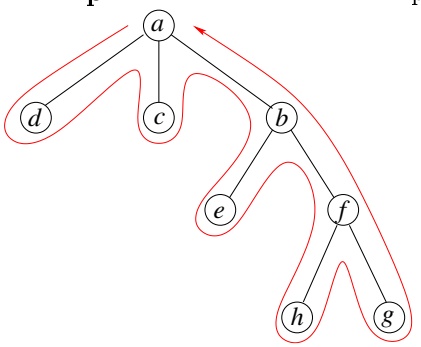
3.2 Premier ancêtre commun

Le problème de la recherche de premier ancêtre commun est le suivant : étant donné un arbre et deux nœuds, on souhaite trouver l'ancêtre commun le plus profond à ces deux nœuds. Pour des questions de simplicité, nous avons travaillé sur des arbres d'entiers.

Ce problème est résoluble par un algorithme qui effectue un prétraitement linéaire en la taille de l'arbre puis répond à chaque requête en temps constant. Les arbres étant en pratique de petite taille, nous nous sommes contentés d'un prétraitement en temps $\mathcal{O}(n \log(n))$ où n est le nombre de nœuds.

Le prétraitement consiste en une traduction vers un autre problème résoluble avec la même complexité, et en le prétraitement pour ce nouveau problème. Il s'agit de la recherche dans un tableau de l'indice de la case contenant la valeur minimale entre deux indices donnés (IMIN). En effet, si l'on effectue un tour eulérien de l'arbre, c'est-à-dire une promenade fermée partant de la racine et passant exactement deux fois par chaque arc, on peut construire un tableau de couples (nœud, profondeur) dans l'ordre des rencontres. On peut alors établir le tableau qui à chaque nœud associe l'indice de première apparition dans le tour eulérien.

Exemple : aimablement fourni par Éric Thierry :



Un Tour Eulerien avec les profondeurs

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Noeud	*a*	*d*	*a*	*c*	*a*	*b*	*e*	*b*	*f*	*h*	*f*	*g*	*f*	*b*	*a*
Profondeur	*0*	*1*	*0*	*1*	*0*	*1*	*2*	*1*	*2*	*3*	*2*	*3*	*2*	*1*	*0*

La recherche de premier ancêtre commun se ramène alors à une requête IMIN relativement aux profondeurs, entre les deux indices de première apparition des deux nœuds concernés.

Le prétraitement pour IMIN est le suivant : on précalcule les réponses à ce problème pour tous les sous-tableaux de longueur une puissance de deux. Cela se fait en temps $\mathcal{O}(n \log(n))$ si n est la taille du tableau, par programmation dynamique. Si $i < j$ est la requête à traiter, on calcule la plus grande puissance p telle que $i + 2^p \leqslant j$ et il ne reste qu'à comparer les valeurs enregistrées pour les requêtes $(i, i + 2^p - 1)$ et $(j - 2^p + 1, j)$.

3.3 Clôture par congruence

L'algorithme est présenté sous la forme d'un foncteur, qui prend en argument un module représentant la théorie $\mathcal{X}$ et fournissant toutes les fonctions citées en 2.2. Un tel module est implanté pour la théorie vide : aucun symbole n'est interprété. On utilise le

terme clôture par congruence pour désigner la clôture par congruence modulo la théorie vide.

Nous avons décidé qu'un terme serait sa propre valeur sémantique. Il s'en suit que $leaves\,(t) = \{t\}$. La fonction *solve* n'échoue jamais : il suffit de renvoyer la substitution qui transforme un des deux termes en le second. Nous avons eu accès a posteriori au code écrit en Coq ([Coq]) par Lescuyer lors de ses travaux sur Alt-Ergo ([Alt]) et il s'est avéré que nous avons fait les mêmes choix sur ce point.

Les classes d'équivalence représentées sous forme d'arbres ayant des valeurs sémantiques pour nœuds, nous avons aussi ajouté un module pour une telle structure en argument du foncteur pour cet algorithme. Comme précisé en 3.2, les recherches d'ancêtres communs se font sur des arbres d'entiers, donc nous avions des mécanismes de traduction : chaque valeur sémantique a un identifiant qui la représente dans l'arbre. Comme dans une même classe d'équivalence les identifiants ne sont pas nécéssairement consécutifs, nous avons un second mécanisme de traduction afin de « normaliser » les entiers contenus dans un arbre. Un système de mémoisation est mis en place pour éviter de recalculer les traductions à chaque appel.

Le module implémentant les fonctions nécessaires à la gestion des classes d'équivalence est écrit à l'aide du module Map d'OCaml ([LDF+12]). Une seconde version, adaptant les travaux de Conchon et Filliâtre ([CF07]), se servant de tableaux dynamiques, a été partiellement écrite. Nous l'avons abandonnée car l'usage des Maps suffisait.

Nous avons aussi modifié le parser utilisé par PSYCHE afin de traiter les fichiers au format SMT-LIB 2 ([BST10]) correspondant au problème de clôture par congruence. En effet, si la transformation du problème d'entrée en arbre syntaxique abstrait est commune pour toutes les théories, chaque théorie doit fournir un mécanisme d'interprétation d'un tel arbre. La gestion des symboles (de prédicat ou de fonction) non interprétés a alors nécessité quelques modifications.

PSYCHE dispose alors d'une procédure de décision capable de réaliser la clôture par congruence de toute théorie respectant les axiomes de la thèse de Lescuyer [Les11], modulo l'implémentation des fonctions requises par l'algorithme. Cependant, seule la clôture par congruence modulo la théorie vide est complètement implantée et il reste des perspectives d'amélioration des implémentations actuelles.

4 Perspectives

Outre l'ajout de nouvelles théories $\mathcal{X}$ dont on veut réaliser la clôture par congruence, l'implémentation de l'algorithme de clôture par congruence modulo une théorie peut être otpimisée via la modification de certaines méthodes. De plus, le passage de la logique propositionnelle à la logique du premier ordre dans PSYCHE soulève des questions qui ont constitué la fin de mon stage.

4.1 Clôture par congruence

Parmi les nouvelles théories à ajouter à notre implémentation, on peut citer par exemple l'arithmétique linéaire rationnelle. Toutes les fonctions nécessaires étant précisées dans la thèse de Lescuyer ([Les11]), nous avons jugé que pour la fin d'un stage il serait plus intéressant d'aborder les questions évoquées en 4.2.

Les procédés d'analyse de conflit peuvent être améliorés. Une idée tirée des travaux de Nieuwenhuis et Oliveras ([NO05]) permettrait d'éviter de réaliser les unions d'explications. En effet, lors de la récupération des étiquettes sur les chemins de nos deux nœuds à leur ancêtre commun, on peut « compresser » l'arbre en considérant que tous les nœuds rencontrés sont en fait un seul et même nœud. Ainsi, si l'on doit parcourir à nouveau une

partie de ces chemins, on ne récupérera pas une seconde fois les explications et les unions ne sont donc plus nécessaires.

Cependant, l'intérêt de l'incrémentalité de l'algorithme est de pouvoir repartir de notre état final lorsque l'on n'ajoute par exemple qu'un seul littéral à l'ensemble en entrée. Il est alors nécessaire, lors de la mémoisation, de garder les arbres non compressés, si l'on veut pouvoir obtenir des explications complètes par la suite.

La recherche d'ancêtre commun peut aussi être adaptée. Comme nous l'avons dit, il existe un algorithme avec un prétraitement linéaire, que nous n'avons pas implanté car les classes d'équivalence sont petites en pratique. Il conviendrait de même de réaliser une étude empirique afin de réaliser si ce prétraitement est vraiment efficace et si un algorithme plus « naïf » asymptotiquement ne serait pas plus rapide sur les instances pratiques rencontrées.

Il est toujours possible de réaliser des tests avec d'autres structures de données. Par exemple, l'implémentation des classes d'équivalences sous forme de tableaux persistants ([CF07]), que nous avons modifiée à l'aide de tableaux dynamiques, pourrait être achevée puis testée.

4.2 Premier ordre

Le passage au premier ordre pose la question de l'instanciation de variables en termes. En effet, par exemple, en calcul des séquents, la règle suivante est en général utilisée :

$$\frac{\Gamma \vdash A\left[x := t\right], \Delta}{\Gamma \vdash \exists x A, \Delta}$$

Mais comment trouver un t convenable ? Sur des problèmes de petite taille, c'est l'intuition humaine qui fournit en général le terme. Si l'on souhaite traiter des problèmes de taille plus conséquente, une automatisation est nécessaire et il faut donc trouver un moyen efficace de remplacer l'appel à l'intuition.

Une solution est de différer l'instanciation en introduisant des variables dites « existentielles ». Les contraintes pesant sur ces variables sont obtenues aux feuilles de l'arbre de recherche. Cependant, la structure de PSYCHE impose au plug-in de n'avoir qu'une connaissance locale de l'arbre. Ainsi, le plug-in ne connaît en général pas l'ensemble des contraintes pesant sur chaque variable. Il est alors nécessaire d'introduire un mécanisme de propagation de contraintes afin de parvenir à trouver une instanciation convenable.

Nous avons écrit deux systèmes travaillant avec une propagation en trois étapes. La première est celle de la construction progressive de l'arbre depuis la racine. Nous propageons via les séquents les dépendances entre variables. En effet, l'instanciation d'une variable « existentielle » peut dépendre des variables introduites précédemment, qu'elles soient « existentielles » ou non, mais pas de celles introduites après.

C'est la deuxième étape qui nous a menés à construire deux systèmes de règles d'inférences en calcul des séquents : celle de la propagation des contraintes obtenues aux feuilles vers la racine. Les branches étant construites progressivement, on peut vouloir partager d'une branche à l'autre les contraintes récupérées. Ainsi, soit l'on récupère les contraintes aux feuilles et on les propage vers la racine en les combinant aux nœuds, soit, lorsque d'un nœud partent deux branches, on en traite une, puis on propage les contraintes obtenues dans la seconde et il en résulte de nouvelles contraintes à propager vers la racine.

La propagation de la deuxième étape ne fait pas encore d'instanciation, car il est naturel d'instancier les variables sur lesquelles pèsent le moins de dépendances et d'en tirer des conclusions sur celles qui en dépendent. La preuve que ces systèmes sont équivalents à un système avec instanciation directe, modulo quelques hypothèses sur la théorie mise en jeu, est en cours d'écriture. La troisième étape est alors la transformation qui passe de ces systèmes à un système avec instanciation directe. Les choix d'instanciation sont faits lors d'un parcours depuis la racine, en les propageant afin de tenir compte des dépendances.

Une discussion a aussi été ouverte à ce sujet avec David Delahaye, auteur de Zenon, un prouveur travaillant avec la méthode des tableaux ([BDD07]).

Conclusion

Le sujet principal de ce stage était l'intégration à PSYCHE, un prouveur automatique, d'une procédure de décision déterminant la consistance ou l'inconsistance d'un ensemble de littéraux relativement à la clôture par congruence d'une théorie égalitaire. En second lieu, nous nous sommes intéressés aux questions posées par le premier ordre, dans le cadre d'une adaptation des règles du calcul des séquents utilisé par PSYCHE.

Il a fallu tout d'abord comprendre les besoins de PSYCHE en matière de procédure de décision, afin d'adapter et de compléter l'algorithme de Lescuyer ([Les11]). Des questions d'union-find et d'algorithmique des graphes ont alors été soulevées. Une fois celles-ci résolues, l'algorithme a pu être implanté en OCaml et testé sur des benchmarks au format SMT-Lib 2.

Les questions d'instanciation de variables, proches de l'unification du premier ordre lorsque l'on souhaite les différer, ont été traitées d'un point de vue théorique, dans l'optique d'une implémentation pratique. Cela a mené à deux systèmes de règles d'inférences simplifiés qu'il faudra développer et certifier afin de les intégrer à PSYCHE.

Des améliorations des implémentations sont encore possibles. Certaines structures de données et certains algorithmes n'ont pas été testés et une étude comparative serait intéressante. Un problème que nous n'avons pas soulevé est celui des symboles de prédicat interprétés. En effet, l'algorithme de Lescuyer traite du symbole de l'égalité et nous avons pu l'intégrer à une procédure de décision traitant aussi des symboles de prédicat non interprétés. Des combinaisons du type Nelson-Oppen, évoquées rapidement par Lescuyer dans sa thèse ([Les11]), seraient peut-être à considérer.

A Règles du calcul des séquents utilisé dans PSYCHE

Règles synchrones

$$(\wedge^+)\ \frac{\Gamma \vdash^{\mathcal{P}} [A] \qquad \Gamma \vdash^{\mathcal{P}} [B]}{\Gamma \vdash^{\mathcal{P}} [A \wedge^+ B]} \qquad (\vee^+)\ \frac{\Gamma \vdash^{\mathcal{P}} [A_i]}{\Gamma \vdash^{\mathcal{P}} [A_1 \vee^+ A_2]}$$

$$(\top^+)\ \frac{}{\Gamma \vdash^{\mathcal{P}} [\top^+]} \qquad (Release)\ \frac{\Gamma \vdash^{\mathcal{P}} N}{\Gamma \vdash^{\mathcal{P}} [N]}\ N \text{ n'est pas } \mathcal{P}\text{-positive} \qquad (Init_1)\ \frac{}{\Gamma \vdash^{\mathcal{P},p} [p]}\ \Gamma_{lit} \models_{\mathcal{T}} p$$

Règles asynchrones

$$(\wedge^-)\ \frac{\Gamma \vdash^{\mathcal{P}} A, \Gamma' \qquad \Gamma \vdash^{\mathcal{P}} B, \Gamma'}{\Gamma \vdash^{\mathcal{P}} A \wedge^- B, \Gamma'} \qquad (\vee^-)\ \frac{\Gamma \vdash^{\mathcal{P}} A, B, \Gamma'}{\Gamma \vdash^{\mathcal{P}} A \vee^- B, \Gamma'}$$

$$(\bot^-)\ \frac{\Gamma \vdash^{\mathcal{P}} \Gamma'}{\Gamma \vdash^{\mathcal{P}} \Gamma', \bot^-} \qquad (\top^-)\ \frac{}{\Gamma \vdash^{\mathcal{P}} \Gamma', \top^-}$$

$$(Pol)\ \frac{\Gamma \vdash^{\mathcal{P},l} \Gamma'}{\Gamma \vdash^{\mathcal{P}} \Gamma'}\ \begin{array}{l} l^\perp \notin \mathcal{P} \\ l \in \Gamma, \Gamma' \end{array} \qquad (Store)\ \frac{\Gamma, A^\perp \vdash^{\mathcal{P}} \Gamma'}{\Gamma \vdash^{\mathcal{P}} A, \Gamma'}\ \begin{array}{l} A \text{ est } \mathcal{P}\text{-positive} \\ \text{ou un littéral } \mathcal{P}\text{-négatif} \end{array}$$

Règles structurelles

$$(Select)\ \frac{\Gamma, P^\perp \vdash^{\mathcal{P}} [P]}{\Gamma, P^\perp \vdash^{\mathcal{P}}}\ P \text{ est } \mathcal{P}\text{-positive} \qquad (Init_2)\ \frac{}{\Gamma \vdash^{\mathcal{P}}}\ \Gamma_{lit} \models_{\mathcal{T}}$$

B Règles d'inférence de $CC(\mathcal{X})$

$$\textsc{Congr}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a = b; \Phi \rangle}{\langle \Theta \mid \Gamma \uplus \Gamma' \mid \Delta' \mid N \mid \Phi'; \Phi \rangle}\ a, b \in \Theta, \Delta[a] \not\equiv \Delta[b]$$

où

$$(p, P) = solve(\Delta[a], \Delta[b])$$

$$\Gamma' = \bigcup_{l \in leaves(P)} l \mapsto \Gamma(l) \cup \Gamma(p)$$

$$\forall r \in \mathcal{R}, \Delta'(r) = \Delta(r)\{p \mapsto P\}$$

$$\Phi' = \left\{ f(\vec{u}) = f(\vec{v}) \,\middle|\, \begin{array}{l} \Delta'[\vec{u}] \equiv \Delta'[\vec{v}], f(\vec{u}) \in \Gamma(p) \\ f(\vec{v}) \in \Gamma(p) \cup \bigcup_{t \in \Theta \mid p \in leaves(\Delta[t])} \bigcap_{l \in leaves(\Delta'[t])} \Gamma(l) \end{array} \right\}$$

N et Δ' sont cohérents

$$\textsc{IncohEq}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a = b; \Phi \rangle}{\langle \bot \mid \Phi \rangle}\ a, b \in \Theta, \Delta[a] \not\equiv \Delta[b]$$

où Γ', Δ' et Φ' sont les mêmes que pour la règle Congr et N et Δ' sont incohérents

$$\textsc{Diff}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a \neq b; \Phi \rangle}{\langle \Theta \mid \Gamma \mid \Delta \mid N \cup \{(a, b); (b, a)\} \mid \Phi \rangle}\ a, b \in \Theta, \Delta[a] \not\equiv \Delta[b]$$

$$\textsc{IncohDiff}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a \neq b; \Phi \rangle}{\langle \bot \mid \Phi \rangle}\ a, b \in \Theta, \Delta[a] \equiv \Delta[b]$$

$$\textsc{Unsolv}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a = b; \Phi \rangle}{\langle \bot \mid \Phi \rangle}\ a, b \in \Theta, \Delta[a] \not\equiv \Delta[b]$$

où $\bot = solve(\Delta[a], \Delta[b])$

$$\textsc{Remove}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a = b; \Phi \rangle}{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid \Phi \rangle}\ a, b \in \Theta, \Delta[a] \equiv \Delta[b]$$

$$\textsc{Add}\ \frac{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid C[f(\vec{a})]; \Phi \rangle}{\langle \Theta \cup \{f(\vec{a})\} \mid \Gamma \uplus \Gamma' \mid \Delta \mid N \mid \Phi'; C[f(\vec{a})]; \Phi \rangle}\ \begin{cases} f(\vec{a}) \notin \Theta \\ \forall v \in \vec{a}, v \in \Theta \end{cases}$$

où $C[f(\vec{a})]$ est une équation ou requête faisant apparaître le sous-terme $f(\vec{a})$ et

$$\begin{cases} \Gamma' = \bigcup\limits_{l \in \mathcal{L}_\Delta(\vec{a})} l \mapsto \Gamma(l) \cup \{f(\vec{a})\} \\ \Phi' = \left\{ f(\vec{a}) = f\left(\vec{b}\right) \middle| \Delta[\vec{a}] \equiv \Delta\left[\vec{b}\right], f\left(\vec{b}\right) \in \bigcap\limits_{l \in \mathcal{L}_\Delta(\vec{a})} \Gamma(l) \right\} \end{cases}$$

avec $\mathcal{L}_\Delta(\vec{a}) = \bigcup\limits_{v \in \vec{a}} leaves(\Delta[v])$

$$\textsc{Query}\ \frac{\left\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a \stackrel{?}{=} b; \Phi \right\rangle}{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid \Phi \rangle}\ a, b \in \Theta, \Delta[a] \equiv \Delta[b]$$

$$\textsc{QueryDiff}\ \frac{\left\langle \Theta \mid \Gamma \mid \Delta \mid N \mid a \stackrel{?}{\neq} b; \Phi \right\rangle}{\langle \Theta \mid \Gamma \mid \Delta \mid N \mid \Phi \rangle}\ \begin{array}{l} a, b \in \Theta, \Delta[a] \not\equiv \Delta[b] \\ \langle \Theta \mid \Gamma \mid \Delta \mid N \mid a = b \rangle \uparrow \end{array}$$

$$\textsc{Incons}\ \frac{\langle \bot \mid e; \Phi \rangle}{\langle \bot \mid \Phi \rangle}$$

Références

[Alt] Alt-ergo. http://alt-ergo.lri.fr/.

[Avr93] Arnon Avron. Gentzen-type systems, resolution and tableaux. *J. Autom. Reasoning*, 10(2) :265–281, 1993.

[BDD07] Richard Bonichon, David Delahaye, and Damien Doligez. Zenon : An Extensible Automated Theorem Prover Producing Checkable Proofs. In Nachum Dershowitz and Andrei Voronkov, editors, *Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 4790 of *Lecture Notes in Computer Science (LNCS)/Lecture Notes in Artificial Intelligence (LNAI)*, pages 151–165, Yerevan (Armenia), October 2007. Springer.

[BS01] Franz Baader and Wayne Snyder. Unification theory. In Robinson and Voronkov [RV01], pages 445–532.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB), 2010. www.SMT-LIB.org.

[CCKL08] Sylvain Conchon, Evelyne Contejean, Johannes Kanig, and Stéphane Lescuyer. Cc(x) : Semantic combination of congruence closure with solvable theories. *Electr. Notes Theor. Comput. Sci.*, 198(2) :51–69, 2008.

[CF07] Sylvain Conchon and Jean-Christophe Filliâtre. A persistent union-find data structure. In Claudio V. Russo and Derek Dreyer, editors, *ML*, pages 37–46. ACM, 2007.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *STOC*, pages 151–158. ACM, 1971.

[Coq] The Coq Proof Assistant. http://coq.inria.fr/.

[Dav01] Martin Davis. The early history of automated deduction. In Robinson and Voronkov [RV01], pages 3–15.

[FGLM13] Mahfuza Farooque, Stéphane Graham-Lengrand, and Assia Mahboubi. A bisimulation between DPLL(T) and a proof-search strategy for the focused sequent calculus. In *2013on Logical Frameworks and Meta-Languages : Theory and Practice (LFMTP 2013)*, June 2013.

[GL13] Stéphane Graham-Lengrand. Psyche : a proof-search engine based on sequent calculus with an LCF-style architecture. 2013.

[Häh01] Reiner Hähnle. Tableaux and related methods. In Robinson and Voronkov [RV01], pages 100–178.

[LDF+12] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. *The OCaml system release 4.00 Documentation and user's manual*, 2012.

[Les11] Stéphane Lescuyer. *Formalisation et développement d'une tactique réflexive pour la démonstration automatique en Coq.* Thèse de doctorat, Université Paris-Sud, January 2011.

[NO05] Robert Nieuwenhuis and Albert Oliveras. Proof-producing congruence closure. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 2005.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories : From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, 53(6) :937–977, 2006.

[Psy] Psyche : the Proof-Search factorY for Collaborative HEuristics. http://www.lix.polytechnique.fr/~lengrand/Psyche/.

[RV01] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.