INF551 :ÉCRITURE D'UNE PREUVE DANS LE CALCUL DES SÉQUENTS DE DIKHOFF

GETTE, G. AND LAURENS, J.

Abstract. Ce rapport synthétise les rsultats de notre travail de première période visant modifier la mthode de preuve d'un SAT-solver intuitionniste dtaill dans **??**. Il s'agit de faire en sorte de transformer l'algorithme pour produire une preuve d'une formule intuitionniste crite dans le calcul des squents de Dikhoff tel que présenté dans [1]. Le prsent rapport détaille dans une première partie le contenu des deux articles sus-cités avant de présenter les mcanismes de notre algorithme modifié et les détails de l'implémentation en OCaml.

Contents

1. Logique intuitionniste et méthode de résolution

1.1. **Détail d'un SAT-solver en logique intuitionniste.** Syntaxiquement, une formule intuitionniste est definie rcursivement, comme il suit :

$A := a|b|c|...$ (un atome)
$A_1 \vee A_2$
$A_1 \wedge A_2$
$A_1 \rightarrow A_2$
$\top|\bot$

On remarque que cette définition diffère de la logique classique par l'absence de symbole de négation. Cette absence est liée à l'absence de l'axiome du tiers exclu, à savoir : $A \vee \neg A = T$. En particulier, toute déduction faite en logique intuitionniste est en un certain sens "calculable".
L'algorithme de [2], utilise une technique équivalente celle de l'algorithme DPLL modulo une thorie en décomposant une formule intuitionniste que l'on cherche à prouver en deux ensemble de formule. Le premier se compose de clause dite "clause-implications" de la forme $(a \rightarrow b) \rightarrow c$ où a, $b$et c sont des atomes. Et le deuxième se compose de clauses dite plates, c'est-à-dire de formules de la forme :

$$(a_1 \wedge a_2 \wedge ... \wedge a_n) \rightarrow (b_1 \vee b_2 \vee ... \vee b_m)$$

C'est-à-dire, une conjonction d'atomes impliquant une disjonction d'atomes. On adopte la convention que l'un des deux cotés peut parfois être vide, auquel cas on remplace à droite par $\bot$ et à gauche par $\top$.

Il est possible de prouver, comme on le fera dans la deuxième partie que toutes les dductions faites à partir de clauses plates par la logique classique sont compatible avec la logique intuitionniste. C'est par ce principe que l'on peut dériver un SAT-solveur en logique intuitionniste à partir d'un SAT-Solveur classique. C'est-à-dire qu'en appelant le SAT-solveur sur la partie plate du problème on vérifie que la partie "plate" est satisfiable et si un modèle est vient contredire cette première partie, on utilise l'ensemble des clauses "implications" comme une théorie contre laquelle tester le modèle et extraire de l'information quant à notre problème. Ainsi, pour présenter l'algorithme il faut d'abord comprendre le prétraitement permettant de décomposer une formule en clauses "plates" et en clauses "implications", puis comprendre comment l'algorithme fait interagir ces deux ensembles.

1.2. **Calcul des séquents intuitionniste.** Une construction importante en logique et en théorie de la preuve est le calcul des séquents qui permet l'écriture et la construction formelle de preuve. L'idée principale est de manier un certain nombre de règles en jouant sur la formes des formules considérées, pour récrire ce que l'on veut démontrer et remonter ainsi successivement à des formes de plus en plus simple et par suite de plus en plus facile à manipuler. Il est naturel de vouloir étendre ce système à la logique intuitionniste. Un séquent intuitionniste s'écrit sous la forme :

$$A_1, A_2, A_3, ..., A_n \vdash B$$

et doit se comprendre comme la formule : $(A_1 \rightarrow (A_2 \rightarrow ... \rightarrow (A_n \rightarrow B)))$. Pour mémoire, l'èquivalent en logique classique, se lit différemment savoir que $A_1, A_2, ..., A_n \vdash B_1, ..., B_m$ se comprend comme $(A_1 \wedge A_2 ... \wedge A_n) \rightarrow (B_1 \vee ... \vee B_m)$.

Les règles du calcul de Dikhoff sont les suivantes :

$$\text{Axiom}\ \frac{}{\Gamma, A \vdash A}$$

$$\wedge\ \text{Left}\ \frac{A, B, \Gamma \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$$

$$\vee\ \text{Left}\ \frac{A, \Gamma \vdash \Delta \qquad B, \Gamma \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$$

$$\wedge\ \text{Right}\ \frac{A, \Gamma \vdash A \qquad B, \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$\rightarrow\ \text{Right}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash (A \rightarrow B)}$$

$$\rightarrow\ \text{LeftAtom}\ \frac{B, a, \Gamma \vdash \Delta}{a \rightarrow B, a, \Gamma \vdash \Delta}$$

$$\rightarrow\ \text{LeftAnd}\ \frac{C \rightarrow (A \rightarrow B), \Gamma \vdash \Delta}{(C \wedge A) \rightarrow B \vdash \Delta}$$

$$\rightarrow\ \text{LeftOr}\ \frac{(C \rightarrow A), (C \rightarrow B), \Gamma \vdash \Delta}{(C \wedge A) \rightarrow B \vdash \Delta}$$

$$\rightarrow\ \text{LeftImplies}\ \frac{C \rightarrow A, \Gamma \vdash C \rightarrow A \qquad B, \Gamma \vdash \Delta}{(C \wedge A) \rightarrow B \vdash \Delta}$$

Pour démontrer une formule automatiquement en utilisant ces règles, il suffit de les appliquer "de bas en haut", et d'essayer de remonter à une forme simple et facilement vérifiable, comme un sequent ne comprenant qu'un ensemble d'atomes. On s'assure de la terminaison d'un tel algorithme utilisant ces règles, en utilisant un poids w et en montrant qu'il s'agit d'une fonction décroissante "en remontant" dans un arbre de preuve; c'est-à-dire que le poids de l'ensemble des formules au dessus de la barre de déduction est strictement infrieur celui placé au dessous de la barre. La fonction w est définie récursivement comme il suit :
$w(a) = 1$ quand a est un atome
$w(A \wedge B) = w(A) + w(B) + 2$
$w(A \rightarrow B) = w(A \vee B) = w(A) + w(B) + 1$
$w(A \vee B) = w(A) + w(B) + 1$

On peut constater que ces règles sont très similaire à celle utilisèe dans le calcul des séquents classique, avec une subtilitè suplémentaire apportée par Dykhoff [1], la règle "$\rightarrow$ Left" est décomposée en 4 nouvelles règlesselon la forme de la formule servant de prémisse l'implication laquelle la règles est appliquée. Cette modification de Dykhoff permet d'éviter la création de copies de formules, et ainsi d'assurer la terminaison (grâce à l'argument du poids décroissant.)

Le but de ce projet est de comprendre les liens qui unissent l'article [2] et le calcul d'une preuve dans ce calcul. Pour ce faire nous modifions l'algorithme original et tentons de formaliser les différentes etapes du calcul pour les récrire dans les règles explicitées ci-dessus. Ce calcul permet deux choses :

- Fournir une méthode efficace de preuve en logique intuitionniste, car l'algorithme décrit dans [2] est de loin l'un des plus efficace de l'état de l'art
- Comprendre ce qui fait précisément l'efficacité de cet algorithme en le formalisant

2. Construction d'une preuve en logique intuitionniste

2.1. **Modification de l'algorithme.** Notre algorithme débute par un prétraitement des données d'entré. Tout d'abord, l'ensemble des atomes sont renommés pour permettre l'introduction de nouveaux atomes durant l'établissement des ensembles S et X. La formule est ensuite mise sous forme canonique selon les règles explicitées en page 625 de [2] de la manière suivante : la formule est poussée dans une liste vide, puis pour chaque lment de cette liste, l'algorithme essaye d'appliquer une des règles. En cas de russite, l'élement est dépilé et les formules générées sont ajoutées en début de liste, en cas d'échec l'algorithme met cet élement de côté et passe aux suivants. Ce traitement génère de nouveaux atomes mais réduit la taille des clauses, donc termine. Enfin la liste est parcourue pour séparer les clauses plates (ensemble S) des clauses de type implication (ensemble X), et pour chaque clause $(a \to b) \to c$ de X, la clause $b \to c$ est ajoutée à S.

Une fois la mise sous forme canonique achevée, nous passons la phase de recherche de preuve, que nous avons rendu rcursive. Cette phase se compose de deux fonctions, *intuitCheck* et *intuitProve* :

- flat clauses S - implication clauses X - assumptions A - proof goal q - SAT-proof p - Intuitionistic proofs p', p2 and p3

```
switch satProve (s, A, q)
    case YesSAT (A', p) :
        return YesI (A', p');
    case NoSAT (M ) :
        switch intuitCheck (s, X , M )
            case True :
                return NoI (M );
            case False (M*, c, i, X*, p2):
                switch intuitProve(($M* \rightarrow c$)::s,X,A,q)
                    case YesI(A', p1) :
                        return YesI(A', p3);
                    case NoI(M'):
                        return NoI(M');
```

avec p' :

$$\text{SAT}\ \frac{\text{p}}{S, X, A \vdash q}$$

et p3 :

$$\text{MP}\ \frac{\text{p1} \qquad \text{IMPL}\ \dfrac{\text{M*, S, X-i, i} \qquad \text{p2}}{S, X - i, \vdash M* \to c}}{S, X, A \vdash q}$$

– Ensemble S – implication clauses X – model M

```
for i      X :
    let (a      b)      c = i
    if a, b, c $\notin$ M then
        switch intuitProve (s, X      {i}, M      {a}, b)
            case YesI (M*,p) :
                return False(M*, c, i, X–{i}, p)
            case NoI(M')
            return True;
```

Cet algorithme tente donc de trouver un modèle vérifiant l'ensemble des clauses plates de S grâce à un SAT-solver : en effet, l'existence d'une preuve classique d'un tel ensemble de clause est équivalente à celle d'une preuve intuitionniste. Ainsi, l'efficacit algorithmique du SAT-solver est utilisée pour accélérer l'exécution en vue de statuer sur l'existence ou non d'une telle preuve. Le modèle éventuel est ensuite confronté aux clauses de X : s'il est compatible, l'algorithme renvoie la preuve associée, sinon, l'ensemble S est augmenté de la clause $M* \to c$ et un nouveau modèle est alors cherché.

2.2. **Récriture d'une preuve classique en intuitionniste.** Le SAT-solver n'étudiant que des clauses plates, donc un sous ensemble très restreint de clauses, seules 3 règles seront utilisées par ce dernier, et il est alors aisé de transformer une preuve classique en preuve intuitionniste.

Ainsi, les règles classiques UNSAT, resolve et currify :

$$\text{UNSAT}\ \frac{A \vdash q}{A, \neq q \vdash \bot}$$

$$\text{resolveSAT}\ \frac{H \vdash L \qquad H', L \vdash A}{H \cup H', \neq q \vdash \bot}$$

$$\text{currifySAT}\ \frac{A \vdash q}{A, \neq q \vdash \bot}$$

deviennent en intuitioniste :

$$\text{doubleNegIntuit}\ \frac{A \vdash q}{A, q \to \bot \vdash \bot}$$

$$\text{MP}\ \frac{\dfrac{H \vdash L}{H, H' \vdash L \to H'} \qquad H', L \vdash A}{H, H' \vdash A}$$

$$\text{currifyIntuit}\ \frac{A, l \vdash \bot}{A \vdash l \to \bot}$$

3. Implémentation et résultat

3.1. **Structure du projet.** Le projet se compose de plusieurs modules :

Generics contient des fonctionnelles usuelles de gestion de liste

Syntax explicite la construction des formules à prouver

Clausification transcrit les formules à prouver sous forme canonique

IKernel implémente les règles de calcul intuitioniste utilisées

Inference fournit quelques règles d'inférence usuelles, et permet de transcrire des règles classiques en intuitionniste.

Forest gére la construction des arbres de preuve

enfin Printer génère une sortie user-friendly en latex de la preuve éventuellement crée

References

[1] R. Dyckhoff and L. Pinto, "A permutation-free sequent calculus for intuitionistic logic.," 1996.
[2] K. Claessen and D. Rosén, "Sat modulo intuitionistic implications," in *Proceedings of the 20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning - Volume 9450*, LPAR-20 2015, (New York, NY, USA), pp. 622–637, Springer-Verlag New York, Inc., 2015.