

Implémentation du Calcul des Constructions ^{Inductive} / Presburger

Pierre-Yves Strub

FORMES, INRIA - Tsinghua University - Beijing

7 Octobre 2009

Le Calcul des Constructions Inductives/Presburger

CoqMT: une implémentation de CIC/ \mathbb{N}

Certification d'une procédure de décision (avec A. Mahboubi)

Une formalisation en Coq + Ssreflect

Génération de certificats

Calcul des Constructions ^{Inductive}/Presburger:

- ▶ un Calcul des Constructions Inductives
- ▶ une procédure de décision pour l'arithmétique linéaire dans la conversion

Avantages:

- ▶ Plus d'automatisation
 - ▶ Preuves plus courtes...
 - ▶ ...mais pas forcément plus rapide à vérifier
- ▶ Programmer plus facilement avec les types dépendants

- ▶ Modification de la règle de conversion:

Une conversion entre 2 *termes arithmétiques*
fera appel à une *procédure de décision*

- ▶ E.g., $\text{list}(x + (f\ y)) \equiv \text{list}((f\ (y + 0)) + x)$

Car $z = z' \Rightarrow x + z = z' + x$
(avec $z \rightarrow (f\ y)$, $z' \rightarrow (f\ (y + 0))$)

CoqMT:

- ▶ Une modification du noyau de **Coq** permettant l'introduction dynamiquement de procédures de décision dans la conversion.
- ▶ On décrit ici
 - ▶ Comment charger de telles boites noires
 - ▶ Comment le noyau les utilise

Roadmap de l'ajout d'une théorie

Pour une intégration effective d'une procédure de décision:

1. Chargement, au sein du noyau, d'une *boite noire*:
 - ▶ Description de la théorie
 - ▶ Code effectif de l'algorithme de décision
2. Etablissement de *liens entre les définitions Coq et la boite noire* (e.g. mapping des symboles au premier ordre sur Coq)

Chargement d'une boîte noire (1/2)

- ▶ Théorie $\mathcal{T} = (\Sigma, \mathcal{A}, \mathbf{solve}) \rightarrow$ boîte noire déclarant auprès du noyau
 1. une sorte de travail (e.g. **nat**) \rightarrow termes monosortés
 2. des symboles Σ (constructeurs ou définis) (e.g. $0_{[C,0]}$, $+_{[D,2]}$)
 3. une axiomatique \mathcal{A} (e.g. $\forall xy, x + S(y) = S(x + y), \dots$)
 - ▶ Soit via une AST interne
 - ▶ Soit via un langage pour la logique de premier ordre
 4. une fonction de décision **solve** (fonction **O'Caml**)
- ▶ Un contrat \rightarrow
solve sait décider la *validité de clauses de Horn* pour \mathcal{T} .

Chargement d'une boîte noire (2/2)

- ▶ Possibilité d'insérer plusieurs théories
 - ▶ Doivent travailler sur des sortes disjointes
 - ▶ Ne peuvent pas partager de variables

- ▶ Théories = code **O'Caml**
⇒ peuvent être insérer dynamiquement via le système de chargement de code de **Coq** (Declare ML Module)

Lier une théorie (1/3)

- ▶ Une boîte noire est *inerte* par défaut
 - ▶ Elle ne sait pas comment interagir avec les termes Coq
- ▶ Il faut
 - ▶ Lier les symboles au premier ordre sur des définitions Coq
 - ▶ Prouver que le mapping défini vérifie bien l'axiomatisation
- ▶ **Plusieurs liaisons possible sur une théorie**

Lier une théorie (2/3)

Pour Presburger, on peut e.g.

- ▶ Lier la sorte **nat** sur l'inductif

Inductive $\overline{\text{nat}} : \text{Type} := \overline{0} : \text{nat} \mid \overline{S} : \text{nat} \rightarrow \text{nat} .$

- ▶ Lier les symboles **0** et **S** sur les constructeurs $\overline{0}$ et \overline{S} de $\overline{\text{nat}}$.
- ▶ Lier le symbole **+** sur la définition **Coq** standard `Peano.plus` de l'addition.

Lier une théorie (3/3)

Le système demandera alors de vérifier, dans **Coq**, que:

- ▶ $\overline{\text{nat}}$ est de type `Type`, et que les symboles liés respectent l'arité des symboles de \mathcal{T} (e.g. $\overline{0} : \overline{\text{nat}}$, $\overline{S} : \overline{\text{nat}} \rightarrow \overline{\text{nat}}$).
- ▶ que les symboles **Coq** liés à des symboles constructeurs sont injectifs (ici, que \overline{S} est injectif).
- ▶ que les symboles liés à des symboles constructeurs ne sont pas confondus (ici, que $\overline{0}$ et $(\overline{S} t)$ ne sont pas égaux pour tout t).
- ▶ que les axiomes de \mathcal{T} sont bien respectés par les symboles liés. E.g., de $\forall x y, x + S(y) = S(x + y)$, il sera demandé

$$\text{forall } (x y : \text{nat}), x + \overline{+} (\overline{S} y) = \overline{S}(x + \overline{+} y)$$

Le noyau (1/2)

Initialement, le noyau décide de la conversion de 2 termes en

- ▶ Normalisant faiblement de tête
- ▶ Puis, en comparant les symboles de tête
 - ▶ **Égaux**, test de conversion sous les sous-termes appariés
 - ▶ **Sinon**, échec de la conversion

En cas d'échecs, le noyau vérifie maintenant

- ▶ qu'un des deux termes n'a pas un symbole lié en tête
 - ▶ **Si oui**, utilisation de la procédure de décision
 - ▶ **Sinon**, échec de la conversion

Le noyau (2/2)

E.g., $\overline{\text{list}}(x \overline{+} (f y)) \equiv \overline{\text{list}}((f (y \overline{+} \overline{0})) \overline{+} x)$?

- ▶ L'ancienne conversion échoue sur

$$x \overline{+} (f y) \equiv (f (y \overline{+} \overline{0})) \overline{+} x$$

- ▶ Le nouveau noyau reconnaît deux termes à chapeau algébrique
 - ▶ Il collecte les *aliens* $(f y)$ et $(f (y \overline{+} \overline{0}))$
 - ▶ Ces aliens étant convertibles, il décide de les **abstraire** par une même variable $\Rightarrow x \overline{+} z = z \overline{+} x$.
 - ▶ Il envoie le but $x + z = z + x$ à la boîte noire (qui répond positivement)
 - ▶ Les deux termes sont décrétés convertibles

Ce qui est fait

- ▶ Code de chargement et de liaison de théories
- ▶ Modification de la fonction de conversion
- ▶ Boite noire (simplexe) pour l'arithmétique linéaire
 - ▶ Liaison par défaut sur les entiers standards de Coq

- ▶ Dépôt GIT: <http://pierre-yves.strub.nu/coqmt/>

Ce qu'il reste à faire

- ▶ **Meilleure communication** entre Coq et les boîtes noires
 - ▶ Ne plus communiquer via des AST
 - ▶ Des liaisons non 1-à-1 (e.g. entiers binaires, multiplication par une constante...)
- ▶ Extraction des équations: quasiment fini
 - ▶ Le noyau essaie d'être **incrémental** le plus possible, mais...
 - ▶ ...ne donne pas cette possibilité aux boîtes noires
- ▶ Algorithmes de décision **incrémentaux** et à la Shostak (+ API entre boîtes noires et le noyau)
Ex. (Shankar 2006):
 - ▶ simplex général incrémental
 - ▶ toutes les **informations connues sur les égalités** entre termes sont **communiquées via une substitution**

Ce qu'il reste à faire

- ▶ Conversion *sous les match*
 - ▶ Un algorithme t.q. (Shankar 2006) calcule partiellement et au maximum les termes (e.g. extrait le n max. t.q. $t = S^n u$)
⇒ réduction de `match t with ...`
- ▶ Gérer plus *efficacement les aliens*
 - ▶ Actuellement, comparaison 2 à 2
 - ▶ Utilisation de contre-modèles comme les solveurs SMT
- ▶ Prospectif: transformer l'algorithme top-down actuel en un algorithme de décision général où la $\beta\iota\zeta\delta$ est une théorie comme une autre. (Nelson/Oppen, Shostak)

Le Calcul des Constructions Inductives/Presburger

CoqMT: une implémentation de CIC/ \mathbb{N}

Certification d'une procédure de décision (avec A. Mahboubi)

Une formalisation en Coq + Ssreflect

Génération de certificats

Deux approches pour la certification

- ▶ Procédure de décision certifiée (e.g. obtenue par extraction)
 - ▶ On vérifie *a priori* toutes les exécutions
- ▶ Procédure de décision générant un certificat
 - ▶ On vérifie *a posteriori* après chaque exécution

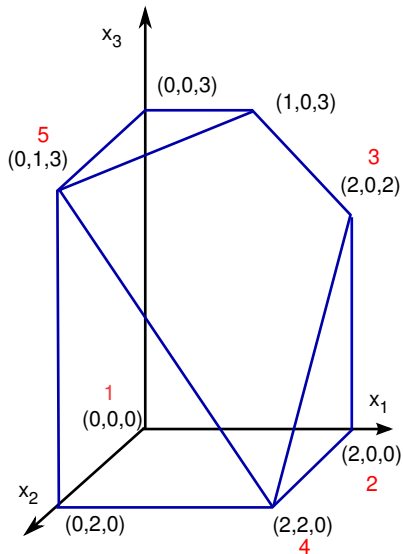
Nous nous intéressons ici à la
certification de l'algorithme du simplexe

Le simplexe (1/2)

Problème d'optimisation d'une fonction linéaire
sous des contraintes linéaires de satisfiabilité

- ▶ Peut servir de base à l'élaboration d'une boîte noire intégrable à notre nouvelle version de Coq.
- ▶ Nous avons étudié
 - ▶ La version certifiée
 - ▶ La version avec génération de certificat

Le simplexe (2/2)



Le Calcul des Constructions Inductives/Presburger

CoqMT: une implémentation de CIC/ \mathbb{N}

Certification d'une procédure de décision (avec A. Mahboubi)

Une formalisation en Coq + Ssreflect

Génération de certificats

Une formalisation en Coq + Ssreflect (1/2)

- ▶ Ce que nous utilisons:
 - ▶ Ssreflect (équipe Mathematical Components)
 - ▶ Bibliothèque pour l'algèbre linéaire (Sidi Ould Biha, équipe M.C.)

- ▶ Ce que nous **avons formalisé**:
 - ▶ Anneaux et corps ordonnés
 - ▶ Un peu de convexité (sur les polytopes)
 - ▶ Correction des pas de l'algorithme d'optimisation
 - ▶ Correction des conditions d'arrêt

Une formalisation en Coq + Ssreflect (2/2)

- ▶ Ce que nous n'avons pas formalisé:
 1. Dualité
 2. Théorie générale de la convexité (Krein-Milman, ...)
 3. Terminaison (Bland's rule, ...)
 4. Initialisation (méthode des 2 phases, ...)
- ▶ Les points (3) et (4) sont nécessaires pour l'obtention d'un algorithme certifié.

Le Calcul des Constructions Inductives/Presburger

CoqMT: une implémentation de CIC/ \mathbb{N}

Certification d'une procédure de décision (avec A. Mahboubi)

Une formalisation en Coq + Ssreflect

Génération de certificats

Notre approche

Adapté l'approche des **solvers SMT** pour la décision de l'arithmétique linéaire, dans le cas où l'on optimise la **fonction constante**.

“A fast Linear-Arithmetic Solver for DPLL(T)”
B. Dutertre-L. de Moura (CAV'06)

Roadmap

- ▶ Effectuer les calculs en dehors du système de preuves
- ▶ Fournir un témoin ou certificat
- ▶ Vérifier ce certificat dans le système de preuves

Comment ça marche ?

- ▶ S'il existe une solution
 - ▶ On calcule cette solution en dehors de Coq
 - ▶ Coq vérifie que cette solution vérifie les contraintes initiales
 - ▶ Cette vérification est donc faite par du calcul certifié
- ▶ S'il n'existe pas de solution
 - ▶ On calcule, en dehors de Coq, une combinaison linéaire inconsistante de l'ensemble initial de contraintes
 - ▶ Coq vérifie l'inconsistance
 - ▶ Cette vérification est simple et utilise seulement quelques lemmes d'arithmétiques

Un exemple

Du système:

$$\begin{cases} x + y \geq 0 & (e_1) \\ x \leq -1 & (e_2) \\ y \leq -1 & (e_3) \end{cases}$$

on obtient:

$$(e_1) \quad 0 \leq x + y \leq -2 = (-1) + (-1) \quad (e_2) + (e_3)$$

ce qui est absurde:

$$0 \leq -2$$

Le certificat (cas insatisfiable)

Le certificat (pour le cas insatisfiable) est de la forme:

- ▶ Une liste de coefficients (combinaison linéaire)
- ▶ Un énoncé absurde de la forme $a \leq b \wedge b < a$

Pour vérifier ce certificat, nous avons besoin de:

- ▶ Calcul: l'énoncé absurde n'utilise que des termes clos
- ▶ Vérifier une combinaison linéaire d'inéquations
- ▶ Utiliser la transitivité de l'ordre

Nombre linéaire de pas de déduction + Calcul au sein de Coq

Ce qui est fait

- ▶ Un algorithme du simplexe en O'Caml
 - ▶ Non totalement optimisé ?
 - ▶ Calcul en arithmétique exacte (rationnelle et entière)
 - ▶ Utilisation des coupures de Gomory (arithmétique entière)
 - ▶ Gestion des inégalités larges et strictes
 - ▶ Sait générer soit le témoin, soit le certificat de non-satisfiabilité
- ▶ Génération de la preuve Coq, à partir des certificats, en cours
- ▶ Benchmark avec les traces de Z3
(Chantal Keller, TypiCal, INRIA Saclay)