Vers un assistant à la preuve en langue naturelle

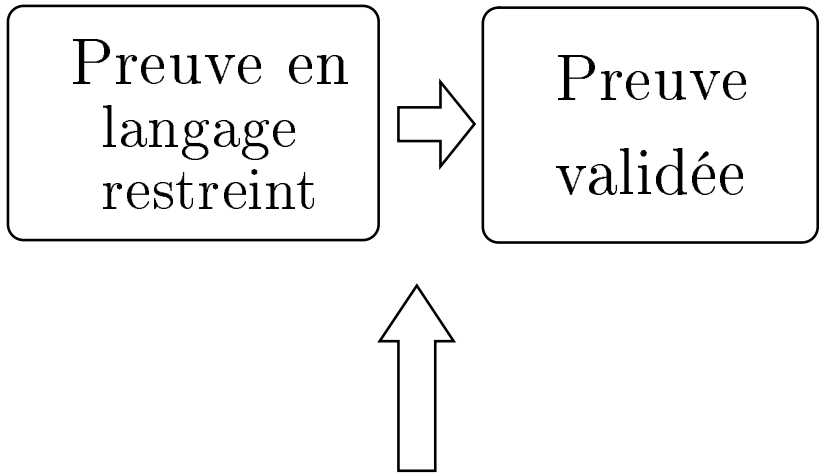
Thèse de Doctorat

Patrick Thévenon

LAMA, Université de Savoie
Le Bourget-du-Lac

5 Décembre 2006

Le contexte

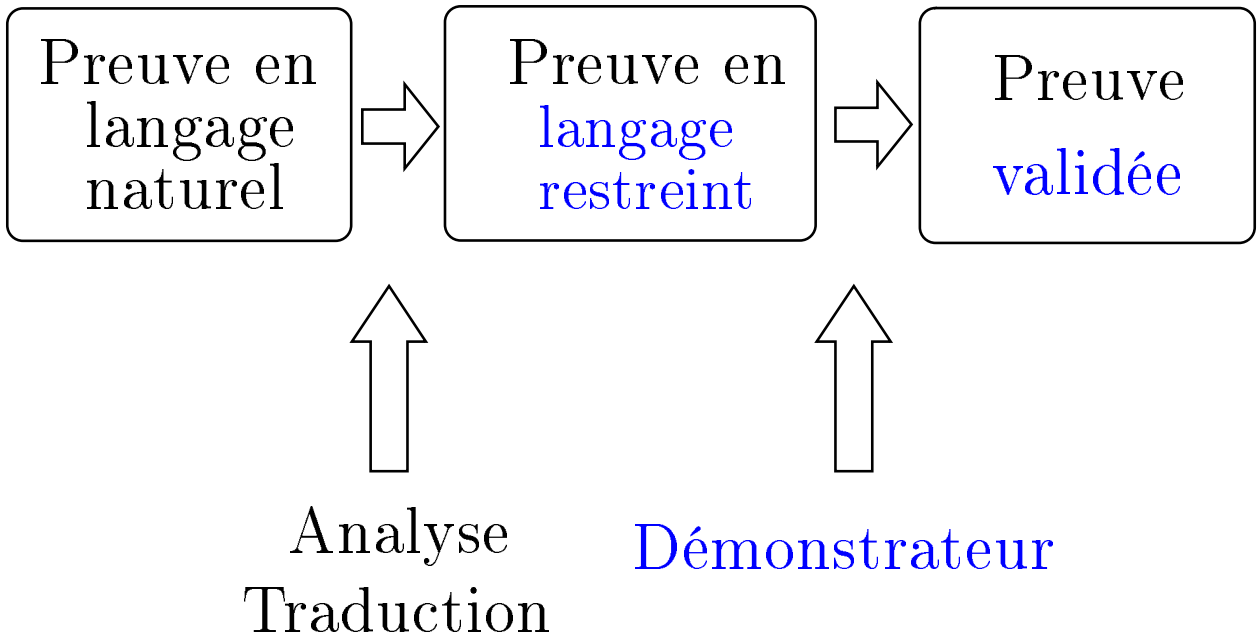


Démonstrateur

Mizar, Isar (Isabelle), MMode (Coq),
SPL (HOL), DECLARE, SAD

Le cadre

- Le projet DemoNat :
  - Analyser et valider des preuves en langue naturelle



Ma recherche

Quatre aspects assez indépendants
entrant dans le cadre du projet :

- Pratique :
  - Définition d'un langage restreint
  - Implémentation d'un démonstrateur automatique
- Théorique :
  - Problème de typage principal
    dans un lambda calcul typé avec deux flèches
  - Etude d'un système logique observé sur le
    démonstrateur au premier ordre

L'analyse des preuves en langue naturelle

Le langage restreint

Typage principal avec deux flèches

Le démonstrateur

Un système logique

Perspectives

L'analyse des preuves en langue naturelle

- Un exemple de preuve en langue naturelle
- Des outils linguistiques

Le langage restreint

Typage principal avec deux flèches

Le démonstrateur

Un système logique

Perspectives

Un exemple de preuve en langue naturelle

Proposition

Pour toutes fonctions f et g,
si $f \circ g$ est injective ou surjective
alors g est injective ou f est surjective

Preuve : Soit f et g.

- Supposons que $f \circ g$ est injective. Soient x et y tels que $g(x) = g(y)$. Alors $f \circ g(x) = f \circ g(y)$. Par injectivité de $f \circ g$, $x = y$ et donc g est injective.
- Supposons maintenant que $f \circ g$ est surjective. Soit z, alors il existe x tel que $f \circ g(x)$ est égal à z. Par conséquent $f(g(x)) = z$. Donc f est surjective. □

Des outils linguistiques

- Les ACGs
  - Analyse la syntaxe et la sémantique de la langue naturelle
  - Génère des langages de λ-termes linéaires
  - Le cadre homogène permet la composition

Des outils linguistiques

- Les ACGs
  - Analyse la syntaxe et la sémantique de la langue naturelle
  - Génère des langages de λ-termes linéaires
  - Le cadre homogène permet la composition
- La SDRT
  - DRT : associe une sémantique à une succession de phrases
  - Permet d'étudier les relations de discours (justification, élaboration, ...)

Des outils linguistiques

- Les ACGs
  - Analyse la syntaxe et la sémantique de la langue naturelle
  - Génère des langages de λ-termes linéaires
  - Le cadre homogène permet la composition
- La SDRT
  - DRT : associe une sémantique à une succession de phrases
  - Permet d'étudier les relations de discours (justification, élaboration, ...)

On imagine alors le shéma suivant :

Analyse syntaxique $\Rightarrow$ Représentation du discours $\Rightarrow$ Langage restreint

Le langage restreint

- But :
  - Description de preuve par une petite grammaire formelle
  - Permettre de donner des indications au démonstrateur

Le langage restreint

- But :
  - Description de preuve par une petite grammaire formelle
  - Permettre de donner des indications au démonstrateur
- Particularité :
  - La grammaire elle-même est indépendante de la logique

Le langage restreint

- But :
  - Description de preuve par une petite grammaire formelle
  - Permettre de donner des indications au démonstrateur
- Particularité :
  - La grammaire elle-même est indépendante de la logique
- Interprétation d'une phrase :
  - La phrase est liée au but courant
  - On détermine un arbre de règles logiques ayant le but courant pour racine
  - A chaque règle de l'arbre est associé un but dont la preuve valide la règle
  - Les buts restants sont donnés à l'utilisateur

La grammaire par l'exemple

Traduisons la preuve de la formule vue précédemment :
$\forall f, g((f \circ g \text{ injective} \vee f \circ g \text{ surjective})$
$\rightarrow (g \text{ injective} \vee f \text{ surjective}))$

let f, g begin assume $f \circ g$ injective show g injective
then assume $f \circ g$ surjective show f surjective end

let x, y assume $g(x) = g(y)$ show $x = y$

deduce $(f \circ g)(x) = (f \circ g)(y)$

by $f \circ g$ injective deduce $x = y$ trivial

let z show $\exists x. f(x) = z$

let x assume $(f \circ g)(x) = z$

deduce $f(g(x)) = z$ trivial

La grammaire par l'exemple

Traduisons la preuve de la formule vue précédemment :
$\forall f, g((f \circ g \text{ injective} \vee f \circ g \text{ surjective})$
$\rightarrow (g \text{ injective} \vee f \text{ surjective}))$

let f, g begin assume $f \circ g$ injective show g injective
then assume $f \circ g$ surjective show f surjective end

let x, y assume $g(x) = g(y)$ show $x = y$

deduce $(f \circ g)(x) = (f \circ g)(y)$

by $f \circ g$ injective deduce $x = y$ trivial

let z show $\exists x. f(x) = z$

let x assume $(f \circ g)(x) = z$

deduce $f(g(x)) = z$ trivial

Et Mizar ?

L'interprétation

- $H \vdash F$ le but courant, un séquent de type déduction naturelle (une seule formule à droite)
- let f, g begin assume $f \circ g$ injective show g injective then assume $f \circ g$ surjective show f surjective end

L'interprétation

- $H \vdash F$ le but courant, un séquent de type déduction naturelle (une seule formule à droite)
- let f, g begin assume $f \circ g$ injective show g injective then assume $f \circ g$ surjective show f surjective end

$\Rightarrow$

$$\frac{H, f \circ g \text{ inj} \vdash g \text{ inj} \quad H, f \circ g \text{ surj} \vdash f \text{ surj}}{H \vdash F}$$

L'interprétation

- $H \vdash F$ le but courant, un séquent de type déduction naturelle (une seule formule à droite)
- let f, g begin assume $f \circ g$ injective show g injective then assume $f \circ g$ surjective show f surjective end

$\Rightarrow$

$$\frac{H, f \circ g \text{ inj} \vdash g \text{ inj} \quad H, f \circ g \text{ surj} \vdash f \text{ surj}}{H \vdash F}$$

$\Rightarrow$

H

$\forall f, g.((f \circ g \text{ inj} \rightarrow g \text{ inj}) \wedge (f \circ g \text{ surj} \rightarrow f \text{ surj}))$

$\vdash F$

L'analyse des preuves en langue naturelle

Le langage restreint

Typage principal avec deux flèches

- Une extension des ACGs
- Le calcul
- Le typage principal
- Fragments
- Preuves

Le démonstrateur

Un système logique

Perspectives

Une extension des ACGs

- Une ACG est principalement
  - Deux structures appelées signatures $\Sigma_i = (C_i, A_i, \tau_i)$, où $\tau_i : C_i \to \mathcal{T}(A_i)$
  - Une paire de fonctions
    - $C_1 \to \Lambda(\Sigma_2)$
    - $A_1 \to \mathcal{T}(A_2)$

    formant un lexique $\mathcal{L}$ vérifiant

$$\vdash \mathcal{L}(c) : \mathcal{L}(\tau(c)) \quad \text{(R)}$$

Une extension des ACGs

- Une ACG est principalement
  - Deux structures appelées signatures $\Sigma_i = (C_i, A_i, \tau_i)$, où $\tau_i : C_i \to \mathcal{T}(A_i)$
  - Une paire de fonctions
    - $C_1 \to \Lambda(\Sigma_2)$
    - $A_1 \to \mathcal{T}(A_2)$

    formant un lexique $\mathcal{L}$ vérifiant
    $$\vdash \mathcal{L}(c) : \mathcal{L}(\tau(c)) \quad \text{(R)}$$
- L'utilisateur donne
  - Les deux signatures
  - Le lexique $\mathcal{L}$:
    1. L'image des constantes
    2. Rien de plus

Une extension des ACGs

- Une ACG est principalement
  - Deux structures appelées signatures $\Sigma_i = (C_i, A_i, \tau_i)$, où $\tau_i : C_i \to \mathcal{T}(A_i)$
  - Une paire de fonctions
    - $C_1 \to \Lambda(\Sigma_2)$
    - $A_1 \to \mathcal{T}(A_2)$

    formant un lexique $\mathcal{L}$ vérifiant
    $$\vdash \mathcal{L}(c) : \mathcal{L}(\tau(c)) \quad \text{(R)}$$
- L'utilisateur donne
  - Les deux signatures
  - Le lexique $\mathcal{L}$:
    1. L'image des constantes
    2. Rien de plus
- Un algorithme
  - Trouve le lexique complet (image des types) grâce à (R) et un algorithme de typage principal

Problème

- Les signatures sont toutes basées sur le même calcul
  Initialement le lambda calcul linéaire

Problème

- Les signatures sont toutes basées sur le même calcul
  Initialement le lambda calcul linéaire
- Celui-ci, utile pour l'analyse syntaxique,
  est limité dans son expressivité pour la sémantique
  (All $\lambda x.$(AND $(A\ x)(B\ x)$))

Problème

- Les signatures sont toutes basées sur le même calcul
  Initialement le lambda calcul linéaire
- Celui-ci, utile pour l'analyse syntaxique,
  est limité dans son expressivité pour la sémantique
  (All $\lambda x.$(AND $(A\ x)(B\ x)$))
- On souhaite conserver le cadre homogène
  que permettent les ACGs
  ainsi que l'efficacité de la linéarité
  mais on a besoin également de non linéarité

Problème

- Les signatures sont toutes basées sur le même calcul
  Initialement le lambda calcul linéaire
- Celui-ci, utile pour l'analyse syntaxique,
  est limité dans son expressivité pour la sémantique
  (All $\lambda x.$(AND $(A\ x)(B\ x)$))
- On souhaite conserver le cadre homogène
  que permettent les ACGs
  ainsi que l'efficacité de la linéarité
  mais on a besoin également de non linéarité
- On définit alors une extension du calcul
  comportant deux types de flèches et de variables

Le calcul (1/2)

$$\frac{}{[\Gamma\ ;\ ] \vdash c : \tau(c)}$$

$$\frac{}{[\Gamma\ ;\ x : \gamma] \vdash x : \gamma} \qquad \frac{}{[\Gamma, x : \gamma\ ;\ ] \vdash x : \gamma}$$

$$\frac{[\Gamma\ ;\ \Delta, x : \alpha] \vdash t : \beta}{[\Gamma\ ;\ \Delta] \vdash \lambda^{\circ} x.t : \alpha \multimap \beta} \qquad \frac{[\Gamma, x : \alpha\ ;\ \Delta] \vdash t : \beta}{[\Gamma\ ;\ \Delta] \vdash \lambda x.t : \alpha \to \beta}$$

Le calcul (2/2)

$$\frac{[\Gamma\ ;\ \Delta_1] \vdash t : \alpha \multimap \beta \quad [\Gamma\ ;\ \Delta_2] \vdash u : \alpha}{[\Gamma\ ;\ \Delta_1, \Delta_2] \vdash (t\ u) : \beta}\ (*)$$

$$\frac{[\Gamma\ ;\ \Delta] \vdash t : \alpha \to \beta \quad [\Gamma\ ;\ ] \vdash u : \alpha}{[\Gamma\ ;\ \Delta] \vdash (t\ u) : \beta}$$

$(*)\ Dom(\Delta_1) \cap Dom(\Delta_2) = \emptyset$

Remarque : l'application est indifférenciée

Le typage principal

- Pour la recherche du type principal d'un terme on a besoin d'un shéma de typage :

$$\frac{[\Gamma\ ;\ \Delta] \vdash t : \alpha -?_n \beta \quad [\Gamma\ ;\ ] \vdash u : \alpha}{[\Gamma\ ;\ \Delta] \vdash (t\ u) : \beta}$$

- Les flèches sous-spécifiées $-?_n$
  sont des variables susceptibles d'être remplacées par une flèche linéaire ou une flèche intuitionniste

La propriété SNIP

- En général des flèches sous-spécifiées sont présentes dans le type principal, dans des positions quelconques

La propriété SNIP

- En général des flèches sous-spécifiées sont présentes dans le type principal, dans des positions quelconques
- On souhaite ne pas afficher ces flèches sous-spécifiées, donc trouver une notion de type principal sans elles où certaines flèches peuvent être remplacées

La propriété SNIP

- En général des flèches sous-spécifiées sont présentes dans le type principal, dans des positions quelconques
- On souhaite ne pas afficher ces flèches sous-spécifiées, donc trouver une notion de type principal sans elles où certaines flèches peuvent être remplacées
- Un terme typé a la propriété SNIP si
  - Les flèches sous-spécifiées sont négatives
  - Les flèches intuitionnistes sont positives

La propriété SNIP

- En général des flèches sous-spécifiées sont présentes dans le type principal, dans des positions quelconques
- On souhaite ne pas afficher ces flèches sous-spécifiées, donc trouver une notion de type principal sans elles où certaines flèches peuvent être remplacées
- Un terme typé a la propriété SNIP si
  - Les flèches sous-spécifiées sont négatives
  - Les flèches intuitionnistes sont positives
- Les termes linéaires ont la propriété SNIP

La propriété SNIP

- En général des flèches sous-spécifiées sont présentes dans le type principal, dans des positions quelconques
- On souhaite ne pas afficher ces flèches sous-spécifiées, donc trouver une notion de type principal sans elles où certaines flèches peuvent être remplacées
- Un terme typé a la propriété SNIP si
  - Les flèches sous-spécifiées sont négatives
  - Les flèches intuitionnistes sont positives
- Les termes linéaires ont la propriété SNIP
- Les termes η-longs ont la propriété SNIP

La propriété SNIP

- En général des flèches sous-spécifiées sont présentes dans le type principal, dans des positions quelconques
- On souhaite ne pas afficher ces flèches sous-spécifiées, donc trouver une notion de type principal sans elles où certaines flèches peuvent être remplacées
- Un terme typé a la propriété SNIP si
  - Les flèches sous-spécifiées sont négatives
  - Les flèches intuitionnistes sont positives
- Les termes linéaires ont la propriété SNIP
- Les termes η-longs ont la propriété SNIP
- On peut alors remplacer les flèches sous-spécifiées par des flèches intuitionnistes

Idée de la preuve (1/2)

- Termes linéaires :
  - On généralise la propriété :
    1. Chaque variable de type qui apparaît apparaît deux fois avec une occurrence positive et une occurrence négative
    2. Le type a la propriété SNIP
    3. Les flèches sous-spécifiées sont toutes distinctes

Idée de la preuve (1/2)

- Termes linéaires :
  - On généralise la propriété :
    1. Chaque variable de type qui apparaît apparaît deux fois avec une occurrence positive et une occurrence négative
    2. Le type a la propriété SNIP
    3. Les flèches sous-spécifiées sont toutes distinctes
  - Vrai pour les termes sous forme β-normale

Idée de la preuve (1/2)

- Termes linéaires :
  - On généralise la propriété :
    1. Chaque variable de type qui apparaît apparaît deux fois avec une occurrence positive et une occurrence négative
    2. Le type a la propriété SNIP
    3. Les flèches sous-spécifiées sont toutes distinctes
  - Vrai pour les termes sous forme β-normale
  - Stable par β-expansion

Idée de la preuve (2/2)

- Termes η-longs :
  - Une notion de justification de chaque atome, chaque flèche dans un type principal est définie : à chaque atome / flèche, on associe un ensemble de sous-termes qui ont cet atome / flèche en tête de type.

Idée de la preuve (2/2)

- Termes η-longs :
  - Une notion de justification de chaque atome, chaque flèche dans un type principal est définie : à chaque atome / flèche, on associe un ensemble de sous-termes qui ont cet atome / flèche en tête de type.
  - La propriété SNIP est généralisée :
    1. Tout est justifié (par des sous-termes justifiants)
    2. Si les sous-termes justifiants sont des variables x de termes $\lambda x.u$ tels que $x \notin u$ alors le type est un atome a et a est unique
    3. Les flèches sous-spécifiées sont uniques et négatives
    4. Les $\longrightarrow$ sont positives

Idée de la preuve (2/2)

- Termes η-longs :
  - Une notion de justification de chaque atome, chaque flèche dans un type principal est définie : à chaque atome / flèche, on associe un ensemble de sous-termes qui ont cet atome / flèche en tête de type.
  - La propriété SNIP est généralisée :
    1. Tout est justifié (par des sous-termes justifiants)
    2. Si les sous-termes justifiants sont des variables x de termes $\lambda x.u$ tels que $x \notin u$ alors le type est un atome a et a est unique
    3. Les flèches sous-spécifiées sont uniques et négatives
    4. Les $\longrightarrow$ sont positives
  - La propriété est stable lors des unifications nécessaires pour obtenir le type principal

L'analyse des preuves en langue naturelle

Le langage restreint

Typage principal avec deux flèches

Le démonstrateur

- Un démonstrateur universel
- Résolution
- Règles de décomposition
- Stratégies

Un système logique

Perspectives

Démonstration

TEST SUR L'EXEMPLE AVEC PHOX

Le démonstrateur comme foncteur

```
module Prover : functor (Logic : Logic) →
sig
Exception Prove_fails
val prove : ( formula * int ) list → formula → unit
end
```

Pour avoir un démonstrateur :

- Définir une logique (formules, unification, ...)
- Appliquer le foncteur à la logique

Logiques utilisées : Prop, FOL (+ langage restreint), PhoX

Résolution

- Principe : trouver une contradiction dans un ensemble de clauses (ensemble de formules disjonctives)
- Deux règles
  - Règle de résolution

$$\frac{C_1, L_1 \quad C_2, L_2 \quad \sigma = mgu(L_1, \overline{L_2})}{C_1\sigma, C_2\sigma}\ res$$

  - Règle de contraction

$$\frac{C_1, L_1, L_2 \quad \sigma = mgu(L_1, L_2)}{C_1\sigma, L_1\sigma}\ contr$$

Règles de décomposition (1/2)

- Problème : comment déterminer l'ensemble de clauses à partir de la formule à prouver ?

Règles de décomposition (1/2)

- Problème : comment déterminer l'ensemble de clauses à partir de la formule à prouver ?
- On ne veut pas décomposer tout quand on a $F \longrightarrow F$ à prouver

Règles de décomposition (1/2)

- Problème : comment déterminer l'ensemble de clauses à partir de la formule à prouver ?
- On ne veut pas décomposer tout quand on a $F \longrightarrow F$ à prouver
- L'idée :
  - Les clauses sont des ensembles de formules (non nécessairement des formules atomiques)
  - Utiliser des règles de décomposition pendant la preuve et non au préalable

Règles de décomposition (2/2)

Exple : Soit $\{\neg F, \Gamma\}$ une clause avec $F = (A \rightarrow B)$
On décompose F, on obtient les deux clauses :
$\{A, \Gamma\}$ and $\{\neg B, \Gamma\}$

$\rightarrow$ Décomposer F ajoute de nouvelles clauses provenant de l'équivalence $F \leftrightarrow (A \rightarrow B)$

$\rightarrow$ C'est la logique qui décompose les formules connaissant leur polarité dans la clause

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ... pour suivre au mieux les indications

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ... pour suivre au mieux les indications
- Une décomposition paresseuse des formules permettant, à l'aide des poids, d'être peu sensible à l'ajout d'hypothèses inutiles

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ... pour suivre au mieux les indications
- Une décomposition paresseuse des formules permettant, à l'aide des poids, d'être peu sensible à l'ajout d'hypothèses inutiles
- Suppression des clauses subsumées et tautologies

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ... pour suivre au mieux les indications
- Une décomposition paresseuse des formules permettant, à l'aide des poids, d'être peu sensible à l'ajout d'hypothèses inutiles
- Suppression des clauses subsumées et tautologies
- Résolution positive ou négative

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ... pour suivre au mieux les indications
- Une décomposition paresseuse des formules permettant, à l'aide des poids, d'être peu sensible à l'ajout d'hypothèses inutiles
- Suppression des clauses subsumées et tautologies
- Résolution positive ou négative
- Séparation sans séparation : ajout de variables (de séparation) propositionnelles pour couper les clauses

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ... pour suivre au mieux les indications
- Une décomposition paresseuse des formules permettant, à l'aide des poids, d'être peu sensible à l'ajout d'hypothèses inutiles
- Suppression des clauses subsumées et tautologies
- Résolution positive ou négative
- Séparation sans séparation : ajout de variables (de séparation) propositionnelles pour couper les clauses

  $\longrightarrow$ OL-déduction pour les clauses de séparation

Démonstration

TEST SUR L'EXEMPLE

L'analyse des preuves en langue naturelle

Le langage restreint

Typage principal avec deux flèches

Le démonstrateur

Un système logique
- Les règles
- Résultats

Perspectives

Un système logique

Les idées développées pour le démonstrateur ont permis d'aboutir à un système pour la logique du premier ordre avec deux variantes :

1. Un système dit 'logique'
2. Un système dit 'implémenté'

Un système logique

Les idées développées pour le démonstrateur ont permis d'aboutir à un système pour la logique du premier ordre avec deux variantes :

1. Un système dit 'logique'
2. Un système dit 'implémenté'

Il est possible de voir une 'similitude' avec

- La déduction libre de M. Parigot
  - Dual du calcul des séquents (règles d'élimination)
- Le calcul des structures de A. Guglielmi
  - Un ensemble de clauses est vu comme une conjonction de formules disjonctives
  - Les règles ne branchent pas

Comparaison

Système logique	Système implémenté
$\dfrac{\Gamma, A; \Gamma', A^{\perp}}{\Gamma, \Gamma'}\ Res$	$\dfrac{\Gamma, A; \Gamma', A'^{\perp}}{\Gamma\sigma, \Gamma'\sigma}\ Res$
$\dfrac{\Gamma, \forall x.A(x)}{\Gamma, A(t)}$	$\dfrac{\Gamma, \forall x.A(x)}{\Gamma, A(y)}\ (*)$
$\dfrac{\Gamma, (\forall x.A(x))^{\perp}}{\Gamma, A(y)^{\perp}}\ (*)$	$\dfrac{\Gamma, (\forall x.A(x, \overline{z}))^{\perp}}{\Gamma, A(f(\overline{z}), \overline{z})^{\perp}}$

(*) y est une variable fraîche

Résultats / Conjecture

Proposition (Les sytèmes sont sûrs et complets)

Il existe une déduction aboutissant à la clause vide à partir de $F^{\perp}$ si et seulement si Il existe une preuve de $\vdash F$ en calcul des séquents

Proposition (Stratégie complète)

La stratégie d'élimination des clauses subsumées et des tautologies est complète

Conjecture

S'il existe une déduction de la clause vide utilisant une résolution sur une formule A telle que A et $A^{\perp}$ proviennent d'une formule F et $F^{\perp}$ alors il existe une déduction de la clause vide faisant directement la résolution sur F et pas sur A

L'analyse des preuves en langue naturelle

Le langage restreint

Typage principal avec deux flèches

Le démonstrateur

Un système logique

Perspectives

Perspectives

- Pratique pour le démonstrateur :
  - calcul des poids, structures de données, stratégies, ...
  - Ajouter une méthode pour l'égalité (paramodulation)
  - Travailler sur les messages d'erreurs
- Théorique :
  - Dans les ACGs :
    - Travailler sur le matching
      I. Cervesato a défini un calcul similaire
    - Autres extensions du calcul
  - Pour le système :
    - Trouver des stratégies de preuve

Bibliographie

- C.-L. Chang, R. C.-T. Lee, *Symbolic Logic and mechanical Theorem proving*, Academic Press, 1973
- P. de Groote, *Towards Abstract Categorial Grammars*, in Association for Computational Linguistics, 2001
- M. Giero, F. Wiedijk, *MMode, a Mizar Mode for the proof assistant Coq*, 2003
- A. Leitsch, *The Resolution Calculus*, Springer, 1997
- *Mizar*, `http ://www.mizar.org/project/`
- M. Roger, *Raffinements de la résolution et vérification de protocoles cryptographiques*, Thèse, 2003
- M. Wenzel, *Isabelle/Isar - a versatile environment for human-readable formal proof documents*, Thèse, 2002