Le projet DemoNat

Patrick Thévenon
patrick.thevenon@univ-savoie.fr

LAMA, Université de Savoie
Le Bourget-du-Lac

4 Mai 2006
Laboratoire J.-A. Dieudonné
Nice

Introduction

Introduction

- But du projet :
  - Analyser et valider des preuves en langue naturelle

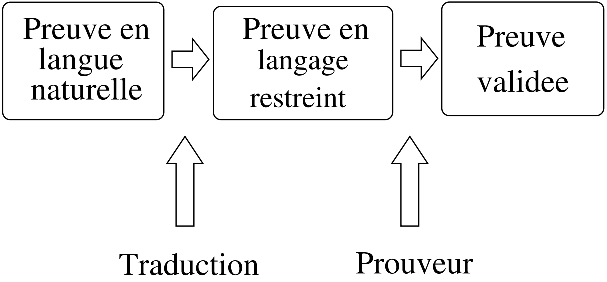
Introduction

- But du projet :
  - Analyser et valider des preuves en langue naturelle
- Interêt :
  - Enseignement
  - Simplicité

Introduction

- But du projet :
  - Analyser et valider des preuves en langue naturelle
- Interêt :
  - Enseignement
  - Simplicité
- Equipes travaillant dans le projet :
  - Lattice/Talana (Jussieu)
  - Calligramme (Nancy)
  - LAMA (Chambéry)

Le système


Preuve en langue naturelle
Preuve en langage restreint
Preuve validee
Traduction
Prouveur

Mon travail dans ce projet

Mon travail dans ce projet

- Pratique :
  - Définition d'un langage restreint
  - Implémentation d'un prouveur

Mon travail dans ce projet

- Pratique :
  - Définition d'un langage restreint
  - Implémentation d'un prouveur
- Théorique :
  - ACGs et typage principal avec deux flèches
  - Etude d'un système logique observé sur le prouveur

Le langage restreint

Le langage restreint

- But :
  - Décrire une preuve
  - Utiliser une petite grammaire
  - Permettre de donner des indications au prouveur

Le langage restreint

- But :
  - Décrire une preuve
  - Utiliser une petite grammaire
  - Permettre de donner des indications au prouveur
- Particularités :
  - Décrit un arbre de règles logiques
  - La grammaire elle-même est indépendante de la logique

Le langage restreint

- But :
  - Décrire une preuve
  - Utiliser une petite grammaire
  - Permettre de donner des indications au prouveur
- Particularités :
  - Décrit un arbre de règles logiques
  - La grammaire elle-même est indépendante de la logique
- Traitement d'une phrase :
  - Liée à un but courant
  - A chaque règle décrite est associé un but "trivial"
  - Les buts suivants sont donnés à l'utilisateur

La grammaire (simplifiée) 1

nc

- ncs
- BY ... (WITH ...) ncs
- PROVE FORM nc MYIN nc
- BYABSURD HYPNAME nc
- SET EQUAL nc
- LABEL HYPNAME

ncs

- DEDUCE FORM nc
- TRIVIAL
- meta

La grammaire (simplifiée) 2

meta

LET CST meta
SEARCH VAR meta
ASSUME FORM meta
SHOW FORM nc SHOWN
meta MYTHEN meta
PBEGIN meta PEND
PROOF nc ENDPROOF

L'interprétation

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

DEDUCE FORM : FORM est prouvée

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

DEDUCE FORM : FORM est prouvée

TRIVIAL : le but courant est prouvé

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

DEDUCE FORM : FORM est prouvée

TRIVIAL : le but courant est prouvé

LET CST : une nouvelle constante ajoutée

SEARCH VAR : une nouvelle variable ajoutée

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

DEDUCE FORM : FORM est prouvée

TRIVIAL : le but courant est prouvé

LET CST : une nouvelle constante ajoutée

SEARCH VAR : une nouvelle variable ajoutée

SHOW FORM : FORM implique le but courant

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

DEDUCE FORM : FORM est prouvée

TRIVIAL : le but courant est prouvé

LET CST : une nouvelle constante ajoutée

SEARCH VAR : une nouvelle variable ajoutée

SHOW FORM : FORM implique le but courant

THEN : une nouvelle prémisse pour la règle

L'interprétation

BY ... (WITH ...) : donné en indice au prouveur

PROVE FORM : la règle de coupure

DEDUCE FORM : FORM est prouvée

TRIVIAL : le but courant est prouvé

LET CST : une nouvelle constante ajoutée

SEARCH VAR : une nouvelle variable ajoutée

SHOW FORM : FORM implique le but courant

THEN : une nouvelle prémisse pour la règle

PBEGIN (...) PEND : parenthèses

PROOF (...) ENDPROOF : preuve de la prémisse courante

La justification

La justification

- Pour chaque règle une formule qui la justifie est formée
  - partager les variables le plus possible
  - Si aucun but n'a changé dans les premisses, ne pas utiliser le but courant dans la formule

La justification

- Pour chaque règle une formule qui la justifie est formée
  - partager les variables le plus possible
  - Si aucun but n'a changé dans les premisses, ne pas utiliser le but courant dans la formule
- Si les formules donnée avec BY et WITH ne sont pas des hypothèses
  - On les prouve
  - On les donne comme indices au prouveur
  - Elles sont oubliées dans le reste de la preuve

Démonstration

QUELQUES EXEMPLES SIMPLES

Le prouveur comme un foncteur

```
module Prover : functor (Logic : Logic) -> 
sig
Exception Prove_fails

val prove : ( formula * int * constraints) list
                        -> formula
                        -> unit
(* raises Prove_fails when no proof is found *)
end
```

Pour avoir un prouveur :

- donner une logique
- l'appliquer au foncteur

La logique demandée

```
module type Logic =
sig
type formula (form)

val elim_all_neg : form -> form
...
type substitution (subs)

type constraints (csts)

val unif : csts -> form -> csts -> form ->
        int * subs * csts * form * form list

val get_rules : csts -> form -> bool ->
        (string * int * subs * csts * form list ) list
end
```

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

Règles de décomposition 1

- Problème : comment déterminer l'ensemble de clauses à partir d'une formule ?

Règles de décomposition 1

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

Règles de décomposition 1

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

Règles de décomposition 2

Exple : Soit $\{\neg F, \Gamma\}$ une clause avec $F = (A \rightarrow B)$
De $F \leftrightarrow (A \rightarrow B)$ on obtient deux clauses :
$\{A, \Gamma\}$ and $\{\neg B, \Gamma\}$
Cela peut être vu comme une résolutions avec les clauses suivantes sur le littéral $F \equiv X_1 \rightarrow X_2$:
$\{X_1, X_1 \rightarrow X_2\}$ et $\{\neg X_2, X_1 \rightarrow X_2\}$

$\longrightarrow$ Décomposer c'est faire de la résolution avec des clauses de règle

$\longrightarrow$ **get_rules** demande pour chaque formule quelle règle peut être appliquée

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ...

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ...
- Suppression des clauses subsumées et tautologies

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ...
- Suppression des clauses subsumées et tautologies
- Résolution positive ou négative optionnelle

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ...
- Suppression des clauses subsumées et tautologies
- Résolution positive ou négative optionnelle
- Splitting sans splitting : ajout de variables (de splitting) propositionnelles attachées aux morceaux de clauses pour couper les clauses

Stratégies

- Un poids sur chaque clause, calculé avec des valeurs telles que la taille des clauses, des unifications, ...
- Suppression des clauses subsumées et tautologies
- Résolution positive ou négative optionnelle
- Splitting sans splitting : ajout de variables (de splitting) propositionnelles attachées aux morceaux de clauses pour couper les clauses

  $\longrightarrow$ OL-déduction pour les clauses de variables de splitting

Les ACGs

Les ACGs

- Définition
  - Deux signatures (ensemble des constantes typées)
  - Un lexique $\mathcal{L}$, morphisme entre deux signatures

Les ACGs

- Définition
  - Deux signatures (ensemble des constantes typées)
  - Un lexique $\mathcal{L}$, morphisme entre deux signatures
- Peut être utilisé pour des traductions entre :
  - Une syntaxe abstraite et une syntaxe concrète
  - Une syntaxe abstraite et une sémantique
  - Lors de la traduction de la langue naturelle vers le langage restreint

Les ACGs

- Définition
  - Deux signatures (ensemble des constantes typées)
  - Un lexique $\mathcal{L}$, morphisme entre deux signatures
- Peut être utilisé pour des traductions entre :
  - Une syntaxe abstraite et une syntaxe concrète
  - Une syntaxe abstraite et une sémantique
  - Lors de la traduction de la langue naturelle vers le langage restreint
- Condition sur le lexique :

$$\mathcal{L}(c) \ : \ \mathcal{L}(\tau(c))$$

Utiliser les ACGs

Utiliser les ACGs

- L'utilisateur donne
  - Les deux signatures :
    1. Les constantes
    2. Le type des constantes

Utiliser les ACGs

- L'utilisateur donne
  - Les deux signatures :
    1. Les constantes
    2. Le type des constantes
  - Le lexique $\mathcal{L}$:
    1. L'image des constantes
    2. Rien de plus

Utiliser les ACGs

- L'utilisateur donne
  - Les deux signatures :
    1. Les constantes
    2. Le type des constantes
  - Le lexique $\mathcal{L}$:
    1. L'image des constantes
    2. Rien de plus
- Un algorithme doit pouvoir :
  - trouver le lexique complet (image des types)
  - Inverser le lexique (non injectif)

Utiliser les ACGs

- L'utilisateur donne
  - Les deux signatures :
    1. Les constantes
    2. Le type des constantes
  - Le lexique $\mathcal{L}$:
    1. L'image des constantes
    2. Rien de plus
- Un algorithme doit pouvoir :
  - trouver le lexique complet (image des types)
  - Inverser le lexique (non injectif)
- Grâce aux conditions sur le lexique
  L'image des types peut être trouvée en utilisant un algorithme de typage principal

Problème

Problème

Les signatures sont toutes basées sur le même calcul

Problème

Les signatures sont toutes basées sur le même calcul

Initialement les ACGs sont basées sur le lambda calcul linéaire

Problème

Les signatures sont toutes basées sur le même calcul
Initialement les ACGs sont basées sur le lambda calcul linéaire
Le lambda calcul linéaire, utile quand on traite de la syntaxe, est limité dans son expressivité pour la sémantique où l'on a besoin d'écrire des formules utilisant plusieurs occurences de mêmes variables

Problème

Les signatures sont toutes basées sur le même calcul
Initialement les ACGs sont basées sur le lambda calcul linéaire
Le lambda calcul linéaire, utile quand on traite de la syntaxe, est limité dans son expressivité pour la sémantique où l'on a besoin d'écrire des formules utilisant plusieurs occurences de mêmes variables
Ainsi une extension du calcul, avec deux types de flèches et de variables a été défini

Problème

Les signatures sont toutes basées sur le même calcul
Initialement les ACGs sont basées sur le lambda calcul linéaire
Le lambda calcul linéaire, utile quand on traite de la syntaxe, est limité dans son expressivité pour la sémantique où l'on a besoin d'écrire des formules utilisant plusieurs occurences de mêmes variables
Ainsi une extension du calcul, avec deux types de flèches et de variables a été défini
Quand on cherche à déterminer le type principal d'un terme, des probèmes apparaissent

Le calcul 1

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

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

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

Le calcul 2

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

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

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

Le typage principal

- On a besoin d'un shéma de typage :

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

Le typage principal

- On a besoin d'un shéma de typage :

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

- Algorithme de typage usuel (Damas-Milner) avec des contraintes
  lors du typage de $(u\ v)$:
  - Si v a des variables linéaires libres u doit avoir un type $\multimap$
  - Sinon on prend une nouvelle flèche sous-spécifiée $-?$ pour typer u

- En général des flèches sous-spécifiée restent à la fin
- Si nous souhaitons les éliminer, il peut y avoir des problèmes
- Exemple :
  Soit

$$t = \lambda g \lambda f \lambda^{\circ} x \lambda u.(g \quad (f\ x) \quad (f\ \lambda t.(t\ u)))$$

son type principal est

$$\begin{aligned} \vdash t\ :\ (b \multimap b \mathbin{-?_1} n) & \quad \rightarrow \\ (((a \mathbin{-?_2} e) \rightarrow e) \multimap b) & \quad \rightarrow \\ ((a \mathbin{-?_2} e) \rightarrow e) & \quad \multimap \\ a & \quad \rightarrow \quad n \end{aligned}$$

t n'est ni linéaire ni η-long

La propriété des flèches

- un terme typé a la propriété des flèches si
  - les flèches sous-spécifiées sont négatives
  - Les flèches intuitionistes sont positive

La propriété des flèches

- un terme typé a la propriété des flèches si
  - les flèches sous-spécifiées sont négatives
  - Les flèches intuitionistes sont positive
- Les termes linéaires ont la propriété des flèches

La propriété des flèches

- un terme typé a la propriété des flèches si
  - les flèches sous-spécifiées sont négatives
  - Les flèches intuitionistes sont positive
- Les termes linéaires ont la propriété des flèches
- Les termes η-longs ont la propriété des flèches

Un système logique

Les idées développées pour le prouveur ont permis d'aboutir à un système de preuve entre

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

Les règles 1

Dans le cas propositionnel :

$$\frac{S = S'; \Gamma, A \vee B}{S; \Gamma, A, B} \qquad \frac{S = S'; \Gamma, (A \vee B)^{\perp}}{S; \Gamma, A^{\perp}} \qquad \frac{S = S'; \Gamma, (A \vee B)^{\perp}}{S; \Gamma, B^{\perp}}$$

Idem avec les flèches ($A \rightarrow B = \neg A \vee B$)

$$\frac{S = S'; \Gamma, (A \wedge B)^{\perp}}{S; \Gamma, A^{\perp}, B^{\perp}} \qquad \frac{S = S'; \Gamma, A \wedge B}{S; \Gamma, A} \qquad \frac{S = S'; \Gamma, A \wedge B}{S; \Gamma, B}$$

$$\frac{S = S'; \Gamma, (\neg A)^{\perp}}{S; \Gamma, A} \qquad \frac{S = S'; \Gamma, \neg A}{S; \Gamma, A^{\perp}}$$

Les règles 2

$$\frac{S = S'; \Gamma, A; \Gamma', A^{\perp}}{S; \Gamma, \Gamma'}\, Res \qquad \frac{S = S'; \Gamma, A, A}{S; \Gamma, A}\, Contr$$

Au premier ordre on ajoute

$$\frac{S = S'; \Gamma, \forall x.A(x)}{S; \Gamma, A(t)} \qquad \frac{S = S'; \Gamma, (\forall x.A(x))^{\perp}}{S; \Gamma, A(y)^{\perp}}\, (*)$$

$$\frac{S = S'; \Gamma, \exists x.A(x)}{S; \Gamma, A(y)}\, (*) \qquad \frac{S = S'; \Gamma, (\exists x.A(x))^{\perp}}{S; \Gamma, A(t)^{\perp}}$$

(*) y est une variable fraîche

Résultats

Proposition

Le sytème est sûr et complet : Soit F une formule
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

Conjoncture

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

Conclusion / Projets

- Pratique pour the prouveur :
  - Besoin de beaucoup d'améliorations
    fonctions de poids, structures de données, stratégies,...
  - A été utilisé par deux logiques classiques
    propositionnelle et premier ordre
  - Sera utilisé dans PhoX, assistant de preuve
    developpé par C. Raffalli
- Théorique :
  - Dans les ACGs :
    - Travailler sur le matching
      I. Cervesato a défini un calcul similaire
    - Autre preuve pour le typage principal avec des sous-types
    - D'autres extensions du calcul, avec des traits
  - Pour le système :
    - Trouver des stratégies de preuve