*TALN 2004, Fès, 19–21 avril 2004*

# Détermination de contenu dans GEPHOX

Adil El Ghali
LATTICE / PPS - Université Paris 7
2, place jussieu
case 7003 - 75251 Paris
adil@linguist.jussieu.fr

## Résumé - Abstract

Le générateur GEPHOX que nous réalisons a pour ambition de produire des textes pour des définition ou preuves mathématiques écrites à l'aide de l'assistant de preuve PHOX. Dans cet article nous nous concentrons sur le module de détermination de contenu `ContDet` de GEPHOX. Après un aperçu sur l'entrée du générateur, *i.e.* la preuve formelle et l'ensemble des règles ayant permis de l'obtenir, nous décrivons les base de connaissances du générateur et le fonctionnement de l'algorithme de détermination de contenu.

This paper deals with content determination in a text proofs generation system. Our system, GEPHOX produces a textual version of a mathematical proof formalized using the proof assistant PHOX. We start with a quick presentation of the input of the generator : the formal proof and the set of rules that the proof assistant user employs in order to find it. We describe the generator knowledge bases and define the reasoning tasks associated with the KB and show how the content determination algorithm work.

## Mots-clefs – Keywords

Génération de textes, logique de description, détermination de contenu, bases de connaissance, assistant de preuve
Natural language generation, description logic, content determination, knowledge bases, proof assistant

# Introduction

La détermination de contenu est la première tâche de tout système de génération de texte. Elle a pour but de produire une représentation abstraite des données fournies en entrée du système. C'est une des tâches les plus ardues et sans doute la plus importante du processus de génération (Reiter & Dale, 2000; Sripada *et al.*, 2001). La plupart des utilisateurs d'un système de génération préfèrent avoir un texte dont le contenu est correct mais pauvrement exprimé plutôt qu'un texte bien *écrit* mais dont le contenu n'est pas satisfaisant.

Le générateur GEPHOX que nous réalisons a pour ambition de produire des textes pour des définition ou preuves mathématiques écrites à l'aide de l'assistant de preuve PHOX (Raffalli, 2002; Raffalli & Roziere, 2002) . Dans cet article nous nous concentrons sur le module de détermination de contenu `ContDet` de GEPHOX.

Un certain nombre de travaux ont été menés dans ce domaine (Hallgren & Ranta, 2000; Coscoy, 2000). Les approches proposées par ces auteurs sont des tentatives plus ou moins réussies d'établir une correspondance directe entre objets mathématiques et représentations linguistiques. Le point de vue que nous défendons est assez différent : nous pensons qu'au moins un niveau de représentation intermédiaire est nécessaire, afin de pouvoir raisonner et manipuler non pas des preuves mathématiques mais une *vue* de ces preuves qui précède leur mise en langue. Ce point de vue se rapproche plus des travaux de (Fiedler, 2001a; Fiedler, 2001b) où une représentation intermédiaire en termes d'actes de langage est calculé pour chaque action de preuve. Cependant, la détermination de contenu et la structuration de document sont confondues dans son travail, alors que nous pensons qu'un certain nombre d'opérations de calcul de contenu sont indépendantes de la structuration et qu'il faut séparer les deux tâches.

L'approche de détermination de contenu que nous proposons est centrée sur les connaissances du système de génération. Nous voyons cette tâche comme la construction d'un univers de discours (section 3). Le module `ContDet` produit à partir de l'entrée du système une représentation (*message*) des connaissances qui vont figurer dans le texte, ce message est exprimé en logique de description *i.e.* il peut être vu comme une A-Box. Par ailleurs, ce message est construit de façon à préserver les éléments de structure de PHOX qui vont être utiles pour la suite du processus de génération, notamment pour la structuration de document.

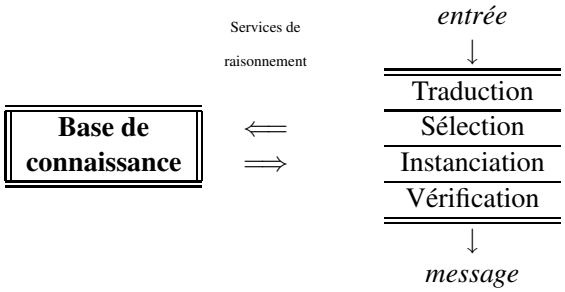


Figure 1: Architecture du module `ContDet`

Le module `ContDet` repose sur une architecture en *pipe-line* à quatre sous-modules, comme le montre la figure 1. Il utilise une base connaissances écrite en Logique de Description à laquelle on accède via une interface fournissant des services de raisonnement (subsomption, unification de concepts, ...).

L'article commence par une rapide présentation de PHOX et des éléments qui serviront comme entrée pour notre générateur (section 1). Nous présenterons ensuite les bases de connaissances du système et les services de raisonnement qui permettent d'y accéder (section 2) avant d'expliciter le fonctionnement des différents sous-modules réalisant la détermination de contenu (section 3).

# 1 Entrée du générateur

## 1.1 PHOX

PHOX est un assistant de preuve basé sur le système $AF_2$[1]. Il permet de réaliser sur ordinateur des preuves mathématiques en déduction naturelle, mais aussi de définir des théories mathématiques grâce à un système de gestion de modules. Il est par ailleurs extensible *i.e.* l'utilisateur peut définir ses propres commandes qui correspondent à des tactiques de raisonnement d'un niveau plus élevé que les règles de déduction habituelles.

Un théorème à prouver est considéré par PHOX comme un but. Le déroulement de la preuve est une application successive de commandes qui réduisent le but courant en sous-buts plus simples. Cet ensemble de commandes est appelé *script de preuve*. PHOX répond aux commandes du rédacteur[2] en donnant le *contexte* courant de la preuve, c'est-à-dire les hypothèses à disposition et le(s) but(s) courant(s) qui reste(nt) à démontrer (les hypothèses et les buts courants forment un *séquent*). Par ailleurs, le système peut fournir à chaque étape le fragment de l'*arbre de preuve* construit *i.e.* les règles et théorèmes utilisés pour passer du séquent précédent au séquent courant. La figure 2 montre un exemple d'entrée pour GEPHOX[3]. Pour produire le texte correspondant à la preuve (ou à une étape de celle-ci), le générateur prend en entrée toutes ces informations.

```
< script >goal ∀x (G x ↔ G1 x ∨ G2 x) → ∀x:G1 G2 x ∨ ∀x:G2 G1 x.< /script >
< phox >
Here is the goal:
goal 1/1
⊢ → (∀ λx (↔ (G x) (∨ (G1 x) (G2 x)))) (∨ (∀ λx (→ (G1 x) (G2 x))) (∀ λx (→ (G2 x) (G1 x))))
< /phox >
< script >intros.< /script >
< proof >
arrow_intro (∀ λx (↔ (G x) (∨ (G1 x) (G2 x)))) λH
(comment "goal 1")
< /proof >
< phox >
New goal is:
goal 1/1
H := ∀ λx (↔ (G x) (∨ (G1 x) (G2 x)))
⊢ ∨ (∀ λx (→ (G1 x) (G2 x))) (∀ λx (→ (G2 x) (G1 x)))
< /phox >
```

Figure 2: Exemple d'entrée du générateur

[1] Arithmétique fonctionnelle d'ordre 2 (Krivine, 1997)

[2] On désigne par *rédacteur* l'utilisateur de PHOX qui construit la version formelle de la preuve, et par *utilisateur* le destinataire du texte produit par GEPHOX.

[3] Les tags $< script >, < phox >$ *et* $< proof >$ délimitent respectivement au *script de preuve* rentré par le rédacteur, à la réponse de PHOX qui définit le *contexte courant* et au fragment d'*arbre de preuve* correspondant.

## 1.2 Les éléments de l'entrée

**Script de preuve** Les scripts de preuve peuvent contenir deux types de commandes. D'une part, les ***annonces*** qui regroupent, entre autres, les déclarations d'objets (types, variables, ...), les définitions ou les énoncés de théorèmes. D'autre part, les ***commandes de preuve*** qui regroupent les règles de déduction (règles d'introduction ou d'élimination de la déduction naturelle) ou tactiques de preuves (raisonnement par l'absurde, par récurrence, ...) servant à avancer dans la recherche de preuve.

**Contexte** À chaque étape de la preuve, le rédacteur passe d'un *séquent*(un ensemble d'hypothèses et un but à prouver) à un autre en appliquant une règle de déduction ou une tactique de preuve. Le contexte est l'ensemble des hypothèses et des buts disponibles à chaque étape, Par exemple, après l'application de la règle d'introduction de $\rightarrow$ sur l'annonce du théorème à démontrer

$$\forall x\ (\mathbf{G}\ x \leftrightarrow \mathbf{G}_1\ x\ \vee\ \mathbf{G}_2\ x) \boxed{\rightarrow} \forall x \in \mathbf{G}_1\ \mathbf{G}_2 x \vee \forall x \in \mathbf{G}_2\ \mathbf{G}_1 x.$$ [4]

le contexte résultant contient l'hypothèse **H:** $\underline{\forall \text{x } ((\mathbf{G}\ \text{x}) \leftrightarrow (\mathbf{G}_1\ \text{x} \vee \mathbf{G}_2\ \text{x}))}$ et le nouveau but qu'il faut prouver $\underline{(\forall \text{x } \mathbf{G}_1\ \text{x} \rightarrow \mathbf{G}_2\ \text{x}) \vee (\forall \text{x } \mathbf{G}_2\ \text{x} \rightarrow \mathbf{G}_1\ \text{x})}$.

**Arbre de preuve** Cette partie de l'entrée nous fournit des informations sur ce que le prouveur a utilisé comme règles élémentaires, théorèmes, axiomes ou définitions. Elle est plus informative que le script de preuve, *e.g.* l'application de la règle d'introduction de $\rightarrow$ est implicite dans le script mais elle ne l'est pas dans le fragment d'arbre de preuve correspondant à cette commande. Par ailleurs, quand le prouveur utilise la tactique automatique, les théorèmes et règles utilisées ne sont explicitées[5] que dans l'arbre de preuve.

# 2 Bases de connaissances

Le module `ContDet` utilise pour représenter les connaissances du domaine une base de connaissances écrite en Logique de Description. Ce choix est motivé par plusieurs raisons, principalement : *(i)* les bonnes propriétés mathématiques de ce formalisme nous permettent de contrôler la complexité du processus de génération, *(ii)* la séparation entre connaissances intensionnelles et extensionnelles est bien adaptée à la génération de textes, *(iii)* l'existence de raisonneurs optimisés fournissant les services standards de raisonnements facilite l'implémentation du système.

## 2.1 Logique de description

Les logiques de description (*description logics* terme abrégé en DL) (Baader *et al.*, 2003) sont un formalisme de représentation des connaissances basé sur la logique du premier ordre et

[4] Le théorème s'énonce ainsi: « si dans un groupe G, tout élément est dans l'un des deux sous-groupes $G_1$ ou $G_2$ alors l'un des sous-groupes est inclu dans l'autre (tout élément de $G_1$ appartient à $G_2$ ou tout élément de $G_2$ appartient à $G_1$). »

[5] Ces informations sont importantes parce qu'elles doivent être incluses dans le texte destiné à un utilisateur qui ne les possède pas dans sa base de connaissances personnelle (cf. section 2.2).

descendant des réseaux sémantiques et des systèmes à base de *frames*. Dans ce formalisme, la théorie est divisée en deux parties, (i) la **T-Box** qui regroupe les *connaissances intensionnelles (conceptuelles ou terminologiques) i.e.* les concepts et les relations du domaine, et (ii) la **A-Box** qui regroupe les *connaissances extensionnelles (assertionelles)* décrivant les individus du domaine et les relations entre eux. Les **concepts** peuvent être vus comme des classes d'individus du domaine et les **rôles** comme des relations binaires entre concepts/individus. Le formalisme offre un certain nombre d'opérateurs (généralement appelés constructeurs) qui permettent de définir des concepts et rôles complexes en fonction d'autres plus simples. Le langage que nous utilisons comprend les constructeurs dont la syntaxe et la sémantique sont présentés dans la table 1.

| **Nom du constructeur** | **Syntaxe** | **Sémantique** |
|---|---|---|
| concept atomique | A | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| top | $\top$ | $\Delta^{\mathcal{I}}$ |
| bottom | $\bot$ | $\emptyset$ |
| conjonction | $C \wedge D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| disjonction ($\mathcal{U}$) | $C \vee D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| complement ($\mathcal{C}$) | $\neg C$ | $\Delta^{\mathcal{I}} \quad C^{\mathcal{I}}$ |
| quantification univ. | $\forall R.C$ | $\{ x \mid \forall y\ (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}} \}$ |
| quantification exist. ($\mathcal{E}$) | $\exists R.C$ | $\{ x \mid \exists y\ (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}} \}$ |
| restrictions numeriques ($\mathcal{N}$) | $>n\ R.C$ | $\{ x \mid \sharp\{ y \mid \forall y\ (x,y) \in R^{\mathcal{I}} \} > n \}$ |
| | $\leq n\ R.C$ | $\{ x \mid \sharp\{ y \mid \forall y\ (x,y) \in R^{\mathcal{I}} \} \leq n \}$ |
| collection d'individus ($\mathcal{O}$) | $\{a_1,\ldots,a_n\}$ | $\{a_1^{\mathcal{I}},\ldots,a_n^{\mathcal{I}}\}$ |
| rôle atomique | P | $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| conjonction de rôle ($\mathcal{R}$) | $Q \wedge R$ | $R^{\mathcal{I}} \cap R^{\mathcal{I}}$ |
| rôle inverse | R 1 | $\{(a_1,a_2) \mid ((a_2,a_1) \in R^{\mathcal{I}}\}$ |
| composition de rôles | Q $\circ$ R | $\{(a_1,a_2) \mid \exists a_3\ (a_1,a_3) \in Q^{\mathcal{I}} \wedge (a_3,a_2) \in R^{\mathcal{I}}\}$ |

Table 1: Syntaxe et sémantique des constructeurs de concepts et de rôles

Les DL offrent plusieurs services de raisonnement (Donini *et al.*, 1996; Horrocks, 2002). Soit une base de connaissances $\Sigma$, on désigne par les lettres majuscules C, D des concepts complexes, $R$, $Q$ des rôleset par les lettres minuscules $a, b$ des individus. Les services de raisonnements que nous utilisons sont définis comme suit.

**Subsomption :** ce qu'on écrit $\Sigma \vdash C \sqsubseteq D$. Ce service permet de tester si dans la base de connaissances $\Sigma$, C est subsumé par D, ce qui est vérifié quand $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ dans tout modèle $\mathcal{I}$ de $\Sigma$. Pour les besoins de la détermination de contenu, la subsomption permet de dire si un concept est plus spécifique qu'un autre. Un service *off-line* lié à la subsomption est la classification ou hiérarchisation des concepts d'une base de connaissances. Par ailleurs, ont dit que C et D sont équivalents (noté $C \equiv D$) si $C \sqsubseteq D$ et $D \sqsubseteq C$.

**Consistance :** ce qu'on écrit $\Sigma \not\models$. Ce service permet de vérifier si $\Sigma$ est satisfiable, *i.e.* que $\Sigma$ a un modèle. La consistance est utilisée par le module `ContDet` pour assurer que le message délivré à la sortie du module ne contient pas d'incohérence, ce qui se verrait dans le texte ou bloquerait le reste du processus de génération.

**Test d'instance :** ce qu'on écrit $\Sigma \vdash C(a)$ (resp. $\Sigma \vdash R(a, b)$). Ce service permet de vérifier si l'assertion $C(a)$ (resp. $R(a, b)$) est satisfaite dans tous les modèles de $\Sigma$, autrement dit, que $a$ est membre du concept C (resp. $\langle a,b \rangle$ est membre de la relation $R$).

**Unification :** ce service de raisonnement, introduit dans (Baader *et al.*, 2000; Baader & Narendran, 2001) a pour but, étant donnés C et D, de trouver une substitution $\sigma$ (appelée uni-

fieur), telle que $\sigma$(C) $\equiv \sigma$(D). Par exemple, si C = A$\vee\forall R$.A et D = (B$_1\wedge$B$_2$)$\vee\forall R$.(B$_1\wedge$B$_2$), la substitution $\sigma$ qui remplace A par B$_1\wedge$B$_2$ dans C est un unifieur pour C et D. On utilise ce service pour trouver un concept dont la définition s'unifie avec une expression conceptuelle et qui pourrait la désigner. Ce qui permet, par exemple, de détecter des tactiques de raisonnement complexes.

**Projection :** c'est un service de raisonnement non standard qui met en jeu deux bases de connaissances $\Sigma_1$ et $\Sigma_2$ telles que $\Sigma_2 \subset \Sigma_1$. Pour tout concept C dans $\Sigma_1$ et n'appartenant pas à $\Sigma_2$, ce service permet de trouver une définition de C avec uniquement des concepts de $\Sigma_2$. La projection est utilisé par le module `ContDet` pour adapter le message aux connaissances de chaque utilisateur : si au cours du processus de détermination de contenu, on manipule un concept qui n'est pas connu par l'utilisateur, le message va être enrichi par la définition de ce concept, ce qui assure une meilleure compréhension du texte final. Par exemple, dans la présentation d'une preuve manipulant des groupes à un utilisateur ne maîtrisant pas le concept de groupe, on explicitera qu'un groupe est un ensemble muni d'une loi associative, possédant un élément neutre et dont tous les éléments possèdent un inverse. Dans ce cas de figure, les concepts de Ensemble, Loi, Element, Inverse, ElementNeutre sont dans la base de connaissances de l'utilisateur, mais pas celui de Groupe.

### 2.2 Bases de connaissances de GEPHOX

GEPHOX dispose de deux bases de connaissances. Le première, DKB (*Domain Knowledge Base*), contient toute les connaissances sur le domaine mathématique dont dispose le système. Elle est structurée en deux parties. D'une part, un **noyau** correspondant à la base de PHOX : les concepts décrivant les types de base, les commandes et les objets mathématiques (variable, operateurs-logique, ...) et méta-mathématiques (theoreme, definition, commandes-de-preuves, ...) de bas niveau. D'autre part, un ensemble de **modules** (dont le découpage est calqué sur celui de PHOX) correspondant aux différentes théories mathématiques, *e.g.* un module pour les entiers, un autre pour les groupes. Lorsque le système entame un processus de génération, la base de connaissances qu'il utilise ne contiendra que le noyau et les modules qui représentent les connaissances des théories que le rédacteur de la preuve a choisi d'utiliser. Ce découpage en modules de la DKB réduit la taille de la base de connaissances effectivement utilisée et améliore l'efficacité du système.

La deuxième base de connaissances dont dispose le système, UKB (*User Knowledge Base*), est spécifique à chaque utilisateur ; il s'agit en fait d'un sous ensemble de la DKB qui reflète ses connaissances.

## 3 Algorithme de détermination de contenu

Comme on l'a vu dans la section 1, l'entrée du générateur GEPHOX est riche car elle comporte certains éléments qui peuvent être vus comme du contenu "pur" (la description des objets mathématiques, l'arbre de preuve) et d'autres qui sont plutôt des informations sur la structure de la preuve (les hypothèses et buts intermédiaires, les commandes appliquées). Dans le déroulement de la tâche de détermination de contenu, chacun de ces éléments prend un statut particulier. Les premiers vont nous donner les référents aux objets du domaine du discours, et les seconds des

actes de langage et des actions de preuve. C'est pour cette raison qu'on a besoin de manipuler dans le module `ContDet` des structures de données permettant de séparer ces deux types d'informations. Ainsi le choix du contenu à exprimer se fait en préservant la structure de la preuve, ce dont on aura besoin dans la tâche de structuration de document (El Ghali & Roussarie, 2003).

La structure de données élémentaire manipulée par le module `ContDet` que l'on appellera **segment** est la traduction d'une étape de preuve dans PHOX. Elle comporte deux parties : la première, *data*, qui contient les informations relatives aux objets mathématiques manipulés dans la preuve, et qui, du point de vue du système de génération, correspond à du contenu pur. La seconde, *struct*, qui contient les informations structurantes de la preuve mathématique. Ces deux parties sont exprimées en DL, ce sont des fragments d'une A-Box. Nous avons choisi de les exprimer dans la même terminologie pour rendre compte plus facilement des relations entre les individus dans la partie *data* et ceux dans la partie *struct*.

Le module `ContDet` va tout au long du traitement d'une entrée construire de manière incrémentale un **univers de discours** qui sera composé de segments. Plus précisément, on définit les structures de données comme suit :

**Définition 1** ***segment***
Un segment est un quadruplet $\langle data, struct, R_{int}, R_{ext}\rangle$ où $data$ et $struct$ sont deux ensembles de triplets $\langle individu, concept, type\rangle$ correspondant aux individus, aux concepts auxquels ils appartiennent et à leur type, pour respectivement la composante données et la composante structure. $R_{int}$ (relations internes au segment) est un ensemble de relations binaires entre des éléments de $data \cup struct$ et $R_{ext}$ (relations externes au segment) est un ensemble de relations binaires dont un participant est dans $data \cup struct$ et l'autre dans un autre segment.

**Définition 2** ***univers de discours***
Un univers de discours $\mathbb{U}$ est une suite ordonnée de segments, telle qu'il n'existe pas d'individu $a$ appartenant à deux segments différents, et telle que tout segment est relié à au moins un autre segment de $\mathbb{U}$ *i.e.* $\forall \mathbb{S} \in \mathbb{U} \quad R^{\mathbb{S}}_{ext} \neq \emptyset$. L'unicité des individus nous assure que notre message ne contiendra pas de redondance. La condition de connexité de l'univers de discours quand à elle sert à identifier les parties du textes qui ont pas de lien et qui correspondront à des unités textuelles (paragraphes, sections, ...) différents dans le texte final.

**Définition 3** ***types d'individus***
Les individus qui composent les segments sont typés pour les besoins de la détermination de contenu. Les types sont dynamiques et n'apparaissent pas dans la base de connaissances. La procédure de calcul du type d'un individu nous permet de distinguer les informations qui vont apparaître dans le message final de celles qui n'y seront pas incluses. La *visibilité* d'un individu signifie qu'il sera présent dans le message. Prenons le cas d'une formule $f$ qui contient des variables et des connecteurs logiques. Si les éléments qui composent $f$ ne sont pas utilisés pour avancer dans la preuve, les individus correspondant ne vont pas apparaître dans le message, mais ils seront présents dans l'univers de discours en étant *cachés*. L'individu correspondant à $f$ les *représentera* dans le message.

On distingue donc trois types d'individu :

**visible** c'est le type par défaut, les individus ayant ce type seront présents dans le message final ;

**représentant** pour les individus qui ont la particularité d'être le seul lien d'un ensemble d'individus avec le reste du segment ou de l'univers ;

**caché** pour ceux qui appartiennent à l'ensemble d'un individu représentant.

Le sous-module de traduction traite une à une les parties de l'entrée correspondant à une commande, le segment ainsi produit est intégré à l'univers de discours courant en effectuant des choix sur les parties à garder et celles à *ne pas dire* ; par la suite l'univers de discours produit est instancié pour l'utilisateur courant, puis vérifié (cf. section 3.3).

## 3.1 Traduction

La tâche du sous-module de traduction est de traduire l'entrée (script, réponses de PHOX et arbre de preuve) en DL. Cette opération doit produire une représentation de **toutes** les informations disponibles dans l'entrée. Elle doit, par ailleurs, préserver la structure de la preuve qui va servir à la structuration de document. La stratégie adoptée pour la traduction est de traiter un bloc composé d'une commande PHOX, de la réponse à cette commande et du fragment d'arbre de preuve correspondant, pour construire une structure composée d'une partie donnée et d'une partie structure.

Une commande PHOX est composé de deux parties : l'instruction de preuve ($intro, elim, goal$, ...) et une partie arguments qui peut être soit l'énoncé (une formule) d'un théorème à prouver ou d'une définition dans le cas des commandes $def, goal, lem$, ..., soit une hypothèse, une variable, un opérateur logique, ... pour les autres commandes, soit vide quand le rédacteur de la preuve laisse à la tactique automatique le soin de choisir les arguments pour sa commande. Les instructions de preuve vont être traduites par des individus appartenant au concept correspondant et cet individu sera rangé dans la case **struct** du segment courant. Pour la partie arguments nous avons deux cas de figures. D'une part, un argument peut introduire un seul individu, c'est le cas pour les variables, les noms d'hypothèses et les opérateurs ; le module de traduction créé alors un individu pour ce référent et doit calculer le concept auquel il appartient. D'autre part, un argument peut introduire plusieurs référents de discours; c'est le cas pour une formule, les variables et connecteurs logiques composant une formule donneront autant d'individus dans l'univers de discours. Par exemple, une commande $goal\ f$ va produire un individu $g \subseteq$ `Goal` qui sera relié par le rôle `has-formula` à la traduction de $f$.

## 3.2 Sélection

Le sous-module de sélection est le coeur de `ContDet` : c'est durant cette phase que les choix les plus importants s'effectuent : décider de ce qui va être inclus dans le message et de ce qu'on *ne va pas dire*, garantir que le message est assez informatif, mais pas trop et ne pas perdre la structure générale de la preuve qui nous servira par la suite.

Le module de sélection construit un univers de discours $\mathbb{U}$ de manière incrémentale, le processus commence par un univers de discours vide, on y ajoute au fur et à mesure les segments fournis par le module de traduction. Quand un segment est ajouté à $\mathbb{U}$, plusieurs procédures sont exécutées :

- on identifie dans le segment courant $\mathbb{S}_C$, les individus déjà présents dans $\mathbb{U}$. Pour un tel individu $a$, on efface l'occurrence de $a$ dans $\mathbb{S}_C$ et on reporte les relations auxquelles il participait dans $\mathbb{S}_C$ vers l'occurrence de $a$ dans $\mathbb{U}$, les dites relations passeront de $R_{int}$ à $R_{ext}$ dans $\mathbb{S}_C$.
- pour les individus correspondant aux commandes dans *struct*, vérifier s'ils peuvent se combiner avec un ou plusieurs autres individus de même nature dans les segments précédents pour produire une tactique plus complexe. On identifie, par exemple, un raisonnement par cas en unifiant avec l'expression conceptuelle qui lui correspond dans la base de connaissances (qui contient des concepts décrivant les différents cas et leurs preuves) avec les concepts correspondant à un ensemble d'individus dans le volet *struct* réalisant un raisonnement par cas. On utilise pour cela l'unification entre les concepts correspondant aux tactiques de haut niveau avec les concepts des individus correspondants aux commandes. Lorsque l'unification est possible, les segments en question et $\mathbb{S}_C$ seront fusionnés en un seul segment.
- pour tous les individus modifiés dans l'univers (les individus de $\mathbb{S}_C$ et ceux qui en ont été effacés), on recalcule les concepts aux quels ils appartiennent et qui, dans certains cas, sont plus spécifiques du fait des nouvelles relations. Cela revient à calculer le concept le plus spécifique de l'individu (on utilise test d'instance et subsomption).
- on identifie les nouveaux individus *représentants* et on marque comme **caché** les éléments de leur ensemble, les individus cachés ne seront pas présents dans le message final. Quand un individu représentant perd son statut, à cause de la présence dans le segment courant d'un élément de son ensemble, ce dernier devient **visible**.

### 3.3 Instanciation et vérification

La phase de sélection produit un univers de discours qui est exprimé dans la DKB. Pour que ce message donne un texte compréhensible par l'utilisateur, on va le projeter dans la UKB, autrement dit, pour tout concept présent dans l'univers et n'appartenant pas à sa UKB, on en calcule la projection dans la UKB. L'instanciation est en quelque sorte une mise au niveau de l'utilisateur du message. La raison principale pour laquelle la sélection n'est pas faite en utilisant directement la UKB est qu'il est parfois nécessaire d'expliquer certaines parties de la preuve à un utilisateur qui n'aurait pas assez de connaissances pour les comprendre, alors que le système en dispose. Ce choix a été fait dans l'optique d'utiliser PHOX couplé à GEPHOX pour l'enseignement des mathématiques (Raffalli & David, 2002).

Pour finir, on vérifie la consistance de la base de connaissances composé de la UKB et la A-BOX correspondant à l'univers de discours. Ceci garantit que le message donnera un texte cohérent par rapport aux connaissances de l'utilisateur.

## Conclusion

Dans cet article nous avons décrit une méthode de détermination de contenu, basée sur l'exploitation des propriétés des bases de connaissances décrivant le domaine mathématique et les connaissances qu'en a le destinataire du texte. Cet algorithme sépare deux types d'inférences mises en oeuvre dans la détermination de contenu : la *première* correspond au déroulement

de cette tâche *i.e.* comment on combine les données en entrée du système pour construire un message cohérent, la *deuxième*, qui exploite les bases de connaissances du système et les mécanismes de raisonnement offerts par celles-ci, garantit à notre message les propriétés de minimalité, de qualité et de fidélité à l'entrée tout en étant au niveau de l'utilisateur

## Remerciements

Merci à Laurence Danlos, Michel Parigot et Paul Rozière pour leur patience et leurs remarques.

## Références

BAADER F., KUSTERS R. & MOLITOR R. (2000). *Rewriting Concepts Using Terminologies – Revisited.* Rapport interne 00-04, Germany.

F. BAADER, D. L. MCGUINNESS, D. NARDI & P. F. PATEL-SCHNEIDER, Eds. (2003). *Description Logics Handbook: Theory, Implementation and Applications.* Cambridge University Press.

BAADER F. & NARENDRAN P. (2001). Unification of concept terms in description logics. *Journal of Symbolic Computation 31(3)*, p. 277–305.

COSCOY Y. (2000). *Explication textuelles de preuves pour le calcul des constructions inductives.* Thèse d'université, Université de Nice-Sophia-Antipolis.

DONINI F., LENZERINI M., NARDI D. & SCHAERF A. (1996). Reasoning in description logics. In G. BREWKA, Ed., *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, p. 193–238. CLSI Publications.

EL GHALI A. & ROUSSARIE L. (2003). Computing the rhetoric of text proofs. In *Proceeding of ICoS-4*, Nancy, France.

FIEDLER A. (2001a). P.rex: An interactive proof explainer. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01).*

FIEDLER A. (2001b). *User-adaptive proof explanation.* PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, Saarbrücken, Germany.

HALLGREN T. & RANTA A. (2000). An extensible proof text editor. In *In Logic for Programming and Automated Reasoning (LPAR'2000),*, volume LNCS/LNAI 1955, p. 70–84: Springer Verlag,.

HORROCKS I. (2002). Reasoning with expressive description logics: Theory and practice. In A. VORONKOV, Ed., *Proceedings of the 18th International Conference on Automated Deduction.*

KRIVINE J.-L. (1997). *Lambda-calcul, types et modèles.* Dunod.

RAFFALLI C. (2002). *User's manual of the PhoX library.* LAMA, Université de Savoie.

RAFFALLI C. & DAVID R. (2002). Apprentissage du raisonnement assité par ordinateur. *Quadrature 45*, p. 25–36.

RAFFALLI C. & ROZIERE P. (2002). *The PhoX Proof checker documentation.* LAMA, Université de Savoie / Université Paris 7.

REITER E. & DALE R. (2000). *Buiding Natural Language Generation Systems.* Studies in Natural Language Processing. Cambridge University Press.

SRIPADA S., REITER E., HUNTER J. & YU J. (2001). A two-stage model for content determination. In *Proceedings of ENLGW-2001*, p. 3–10.