TALN 2003, Batz-sur-Mer, 11–14 juin 2003

Structuration automatique de preuves mathématiques : de la logique à la rhétorique

Adil El Ghali — LATTICE – PPS
Laurent Roussarie — LATTICE – CNRS
Université Paris 7
Case 7003
2, place jussieu
75251 Paris Cedex 05
{adil,laurent}@linguist.jussieu.fr

Mots-clefs – Keywords

Génération automatique de textes, preuves mathématiques, détermination de contenu, logiques de description, structuration de document, SDRT

NLG, mathematical proof, content determination, description logics, document structuring, SDRT

Résumé

Nous présentons dans ses grandes lignes un modèle de structuration de documents pour la génération automatique de preuves mathématiques. les textes de preuve fournissent une illustration claire de l'intrication des structures sémantiques et rhétoriques des discours.

1 Introduction

En génération automatique, la phase de *structuration de document* (ou planification de discours) est d'une importance primordiale. C'est, d'abord, ce qui garantit la cohérence sémantique du texte de sortie et dans une certaine mesure, la pertinence pragmatique. Elle joue également un rôle déterminant de par son intégration dans l'architecture d'un générateur.

Il est communément admis (Reiter & Dale, 2000) que les tâches d'un système de génération de textes peuvent être divisées en deux composants principaux: un module *Quoi-dire?*, chargé de la génération profonde, et un module *Comment-le-dire?* dédié à la lexicalisation, la planification syntaxique et le traitement morphologique. De plus, il est également admis que c'est au niveau du *Quoi-dire?*, dont la tâche principale est la *détermination de contenu*, que doit intervenir la *structuration de document*. Situées ainsi en amont de l'architecture, les stratégies de *structuration de document* ont non seulement un impact durable sur l'ensemble du processus de génération, mais aussi une efficacité double : elles constituent une première étape de *mise en forme* (les choix des structures de discours se répercutent sur l'agencement final du texte), et elles incluent une phase de *raisonnement* qui n'est pas complètement indépendante de la détermination de contenu : il s'agit, par exemple, de construire une argumentation, ce qui implique de choisir, en sus de certaines connexions rhétoriques, des arguments – i.e. du contenu – convaincants.

L'objectif principal de cette communication est d'aborder ce problème de la double efficacité de la structuration de document à la lumière d'une application particulière : *la génération automatique de textes de preuves mathématiques*. La génération de preuves en langue naturelle fournit une illustration claire de l'acuité du problème en question. En effet, un objet preuve possède une structure formelle assez stable et bien définie : par exemple, dans le cas de preuves intuitionnistes, il s'agit d'une hiérarchie d'applications de règles de déduction naturelle, ce que l'on appelle l'arbre de preuve. D'un point de vue sémantique, une telle structure peut souvent se ramener à une série d'inférences (éventuellement imbriquées). On peut alors s'accorder l'intuition que l'information fournie par un arbre de preuve non seulement reflète le contenu sémantique du texte-preuve, mais donne aussi certaines indications sur sa structure rhétorique. C'est d'ailleurs ce sur quoi s'appuient un certain nombre de travaux en génération automatique de preuves (e.g. (Huang & Fiedler, 1997; Fiedler, 2001)) qui élaborent des stratégies plus ou moins sophistiquées d'exploitation ou de parcours de l'arbre pour produire, par appariements et regroupements, un plan de texte. De telles stratégies permettent de générer des textes corrects et conformes à la preuve formelle; cependant beaucoup sont stéréotypés, peu naturels, répétifs, fastidieux, et parfois même abscons. Nous pensons que ces imperfections peuvent s'expliquer par le fait qu'à partir d'une preuve formelle, il n'est pas forcément trivial de distinguer ce qui devrait relever de la détermination de contenu vs. de la structuration de document. De plus même si la structure de l'arbre de preuve contraint fortement la structure rhétorique du texte, elle ne la détermine pas complètement. Empiriquement on constate que, souvent, l'organisation apparente d'un texte-preuve réel rend compte à la fois de la structure de l'arbre mais aussi des mécanismes de raisonnement particuliers utilisés dans la démonstration.

Dans cette communication nous allons proposer une stratégie originale de planification de textes de preuve, visant à obtenir des discours plus naturels et plus proches des preuves rédigées en langue naturelle. Les grandes lignes de cette stratégie seront exposées à travers la présentation du système GEPHOX qui est un générateur de textes de preuves mathématiques obtenues avec le système d'aide à la preuve PHOX (Raffalli & Roziere, 2002). GEPHOX est destiné à des

mathématiciens écrivant des preuves sous PHOX ; il est également conçu pour s'intégrer à PHOX pour l'enseignement de la logique (Raffalli & David, 2001). Nous nous interessons ici au module *Quoi-dire?* du générateur dont l'organisation respecte une architecture standard en deux sous-modules (Fig. 1) : `ContDet` qui calcule le contenu à exprimer à partir de sorties de PHOX et en fonction de connaissances de l'utilisateur[1], et `DocStruct` qui calcule les plans du discours. Les plans de discours produits sont représentés dans le formalisme de la SDRT (Asher & Lascarides, 2003). Ce choix est motivé par deux raisons : d'abord nous nous fondons en grande partie sur le modèle de structuration de document de (Danlos *et al.*, 2001) qui montre l'efficacité de la SDRT pour la génération profonde ; ensuite nous adoptons les conclusions de (Zinn, 1999) selon lesquelles la DRT (et implicitement la SDRT) propose un formalisme particulièrement approprié pour l'analyse et la représentation des textes mathématiques.

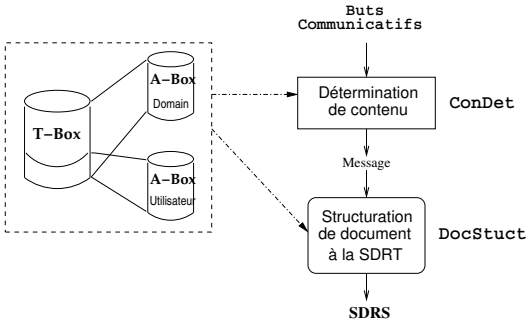


Figure 1: Architecture du *Quoi-dire-?*

2 Détermination du contenu : le module `ContDet`

Le système d'aide à la preuve PHOX permet de réaliser sur ordinateur des preuves mathématiques, en en garantissant la validité. Même si le logiciel dispose d'un algorithme de preuve automatique, son utilisation principale – par des mathématiciens ou pour l'enseignement des mathématiques – est la vérification des étapes de raisonnement. L'utilisateur guide PHOX pour démontrer des théorèmes mathématiques, en laissant le soin au logiciel de réaliser les vérifications et les opérations fastidieuses de la preuve.

2.1 L'entrée de GEPHOX

GEPHOX prend en entrée des informations de deux types: le *script de preuve* qui représente la *trace* de la démonstration (i.e. les commandes entrées par l'utilisateur) et la *sortie de PhoX* proprement dite, qui est constituée de fragments de l'*arbre de preuve* et que l'on peut donc voir comme le *contenu de la preuve*. Le module `ContDet` doit calculer à partir de cette entrée, et en tenant compte des connaissances de l'utilisateur, le message qui sera exprimé par le générateur. La figure 2 donne une illustration des types d'entrées manipulées.

[1]Dans la suite on désignera par *utilisateur* le destinataire du texte produit par le système

Proof script	*PhoX output*
def $Q\ m = m > 0 \wedge \exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$.	**Q** = $\lambda m\ (m > 0 \wedge \exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$: `nat → prop`
lem **dec** $\forall m \in \mathbb{N}(Qm \rightarrow \exists m' \in \mathbb{N}\ (Qm' \wedge m' < m))$.	**dec** = $\forall m \in \mathbb{N}\ (Qm \rightarrow \exists m' \in \mathbb{N}(Qm' \wedge m' < m))$: `theorem`
lem **sq2_irrat** $\forall m \in \mathbb{N}\ \neg Qm$.	**sq2_irrat** = $\forall m \in N\ \neg Qm$ `theorem`
theorem **square2_irrat** $\forall m, n \in \mathbb{N}\ (m^2 = 2 * n^2 \rightarrow m = 0 \wedge n = 0)$.	*goal* 1/1 $\vdash \forall m, n \in \mathbb{N}\ (m^2 = 2 * n^2 \rightarrow m = 0 \wedge n = 0)$
intros. *apply* **sq2_irrat** with H.	*goal* 1/1 H := m ∈ ℕ H0 := n ∈ ℕ H1 := $m^2 = 2 * n^2$ G := $\neg Qm$ ⊢ m = 0 ∧ n = 0
elim H with **[case]**.	*goal* 1/2 H := m ∈ ℕ H0 := n ∈ ℕ H1 := $m^2 = 2 * n^2$ G := ¬ Q m H2 := m = 0 ⊢ 0 = 0 ∧ n = 0 *goal* 2/2 H := m ∈ ℕ H0 := n ∈ ℕ H1 := $m^2 = 2 * n^2$ G := ¬ Q m H2 := y ∈ ℕ H3 := m = S y ⊢ S y = 0 ∧n = 0
⋮	⋮

Figure 2: Extrait de l'entrée de GEPHOX pour «$\sqrt{2}$ *n'est pas rationnel* »

2.2 Les bases de connaissances

Dans GEPHOX les informations sont décrites dans les *concepts* et les *rôles* d'une logique de description (DL, cf. (Baader *et al.*, 2003)). Les connaissances sur le domaine et les connaissances propres de l'utilisateur sont respectivement repésentées dans deux bases : DKB et UKB (UKB est en fait un sous-ensemble de DKB). Les bases de connaissances sont structurées en deux parties. D'une part, la **T-Box** (pour connaissances *t*erminologiques) encode les connaissances intensionnelles, *i.e.* les *concepts* et les *relations* du domaine. Par exemple, le concept `Entier` désigne l'ensemble des entiers naturels. D'autre part, la **A-Box** (pour connaissances *a*ssertionnelles) encode les connaissances extensionnelles, *i.e.* les individus de notre univers. Par exemple $n \in$ `Entier` introduit un individu appartenant au concept `Entier`.

2.3 Calcul du contenu

Le principe général du calcul de contenu est le suivant. Il s'agit, dans un premier temps, de construire les expressions conceptuelles (**T-Box**) correspondant à l'entrée, en se rappelant des individus $\mathcal{I}$ qui leurs correspondent. C'est sur cette **T-Box** qu'opère la détermination de contenu proprement dite[2]. Enfin, les expressions conceptuelles sont instanciées en utilisant les individus $\mathcal{I}$ pour produire la **A-Box** qui représente, sous formes de faits, le message à générer.

Le module `ContDet` commence donc par *traduire* l'entrée du générateur en DL, en utilisant les concepts et rôles de la DKB. Si des définitions et des théorèmes issus de PHOX n'existent pas dans la base de connaissance, la procédure de traduction crée de nou-

[2]Deux raisons motivent ce choix: la première, d'ordre technique, est due au fait que les mécanismes de raisonnement dans une DL s'appliquent plus aisément sur la **T-Box** ; la seconde, plus théorique, vient de l'utilisation que l'on fait des expressions conceptuelles de l'entrée lors de la lexicalisation, mais ceci dépasse le cadre de cet article.

veaux concepts en fonction de ceux déjà présents. Par exemple, à partir de la définition *def* $Q\ m\ =\ m\ >\ 0\ \wedge\ \exists n\ \in\ \mathbb{N}\ (m^2\ =\ 2\ *\ n^2)$ sera créé le concept `Q` défini par $\texttt{Q} \doteq \texttt{Ensemble} \wedge \exists \texttt{sous-ensemble}.\{\mathbb{N}^*\} \wedge \exists \texttt{eq-def}.\{\exists n \in \mathbb{N}\ (m^2 = 2 * n^2)\}$[3].

Une fois construit l'ensemble des expressions conceptuelles ($\mathcal{EC}_D$) représentant l'entrée, `ContDet` doit *sélectionner* dans cet ensemble *ce qui doit être dit*. Sont alors mis en jeu des opérations de filtrage et de regroupement. Une première consiste à détecter des stratégies de raisonnement et leurs paramètres: par exemple, reconnaître que "*elim* H with **[case]**" dénote l'annonce d'un raisonnement par cas; et ensuite calculer les différents cas et les associer à cette annonce. La deuxième opération est la mise en évidence des similitudes entre portions de preuve. On utilise à cet effet l'unification de concepts (Baader & Küsters, 2001). Enfin, `ContDet` cherche à apparier des définitions de concepts complexes de la `DKB` avec des expressions de $\mathcal{EC}_D$. Cela permet de synthétiser des groupes d'informations plus ou moins simples sous le chef d'un concept prédéfini (une telle opération est généralement dénommée *agrégation* en génération automatique). Les axiomes (i.e. les définitions de `DKB`) utilisés à cet effet sont gardés en mémoire pour établir ensuite des relations de second ordre sur les éléments de $\mathcal{EC}_D$.

La **T-Box** ainsi obtenue donne une représentation de la preuve qui fait usage de tous les concepts du domaine. Pour produire un message coopératif et personnalisé, il est nécessaire de confronter cette représentation avec les connaissances de l'utilisateur (`UKB`), afin de vérifier s'il comprend tout ce que comporte $\mathcal{EC}_D$. Cela revient en fait à *expliquer* (i.e. décomposer) les concepts de $\mathcal{EC}_D$ qui ne figurent pas dans `UKB`. On obtient ainsi une nouvel ensemble d'expressions conceptuelles $\mathcal{EC}_U$, qui est calculé par *projection*[4] de $\mathcal{EC}_D$ dans `UKB`. La dernière étape de la détermination du contenu est l'instanciation de $\mathcal{EC}_U$ par les individus de $\mathcal{I}$ et la vérification de consistance du fragment de **A-Box** obtenu.

3 Structuration de document

La figure 3 donne un exemple de sortie de `ContDet`[5]. Ce type de structure peut être vu comme un ensemble *ordonné*[6] de formes logiques qui marquent les pas significatifs de la démonstration. En regard de certaines formules figurent les axiomes qui ont permis de construire la formule. De plus, la structure est plus riche qu'un simple arbre de preuve, puisque `ContDet` fait en sorte qu'elle contienne également des élements d'information provenant du script de preuve. Un des points centraux de notre stratégie de génération profonde va consister à exploiter cette richesse, notamment en postulant que les étapes du script traduites en formes logiques s'assimilent à des *intentions communicatives*. Ces intentions vont permettre en particulier de générer des actes de langage qui « humanisent » le texte de preuve en ajoutant des éléments de rhétorique autres que les habituelles relations logiques.

Le calcul du plan du texte est pris en charge par le module `DocStruct`. Nous adoptons ici le modèle de structuration de document proposé par (Danlos *et al.*, 2001). Partant, `DocStruct` a

[3] $\{c\}$ désigne le concept individuel contenant l'indivicu c.

[4] Soit $C_1 \in KB_1$ et $C_2 \in KB_2$ tel que $C_2 \doteq projection(C_1)$,
- si C_1 est atomique alors $C_2 \doteq C_1$
- si $C_1 \in KB_2$ alors $C_2 \doteq C_1$
- si $C_1 \notin KB_2$ et $C_1 \doteq f(C_{1,i})$ alors $C_2 \doteq f(projection(C_{1,i}))$

[5] La notation $p_1 = Prop(\text{“}m^2 = ...\text{”})$ est un racourci d'écriture pour Prop(p_1) $\wedge$ constant(p_1,“$m^2 = ...$”).

[6] L'ordre provient de la sémantique du conjoncteur dynamique $\wedge$ qui n'est pas symétrique. Dans la figure 3, l'énumération des formules n'est donnée (pour l'instant) que pour la lisibilité du tableau.

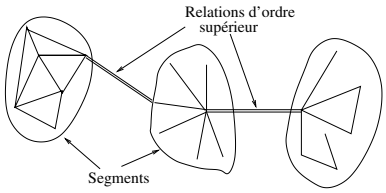
	Forme logique	*Axiomes*
A	sous-ensemble(Q,$\mathbb{N}^*$) $\wedge$ **define**(Q,"$\exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$")	Q $\doteq$ Ensemble $\wedge$ $\exists$sous-ensemble.{$\mathbb{N}^*$} $\wedge$ $\exists$eq-def.{$\exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$}
B	Entier(m) $\wedge$ neg-Q(m)	neg-Q $\doteq$ $\neg$ Q
C	t_1 = not-in($sqrt - 2$,$\mathbb{Q}$) $\wedge$ Theoreme(t_1) $\wedge$ **annonce**(t_1)	
D	1. Entier(m) $\wedge$ Entier(n) $\wedge$ **choose**(m) $\wedge$ **choose**(n) 2. p_1 = Prop("$m^2 = 2 * n^2$") $\wedge$ **suppose**(p_1) 3. l_1 = lemme("$lemme1$") 4. neg-Q(m) $\wedge$ **implies**(e_3,e_4) 5. CaseReason($current$) 6. is-case($current$,e_6) $\wedge$ Nul(m) 7. Nul(n) $\wedge$ **implies**(e_6,e_7) 8. is-case($current$,e_8) $\wedge$ Entier-non-nul(m) ...	...

Figure 3: Fragment du message pour «$\sqrt{2}$ *n'est pas rationnel* » (sortie de ContDet)

deux tâches principales à mener pour produire une structure de document : 1) choisir les unités minimales de la structures, 2) choisir les relations rhétoriques qui lient les unités entre elles pour garantir la cohésion et la cohérence du discours.

3.1 Segmentation de l'ensemble de formes logiques

La structure de la **A-Box** calculée par ContDet est celle d'un graphe connexe[7] (les sommets correspondant aux individus et les arcs à des rôles ou relations conceptuels). Dans ce graphe, certains sous-graphes sont fortement connexes (pseudo-cliques). Notre stratégie est de considérer que ce sont ces sous-graphes qui donneront lieu à des unités de discours, i.e. des segments minimaux de contenu. Par ailleurs, nos sous-graphes sont reliés entre eux par des dépendances que nous assimilerons à des relations d'ordre supérieur portant sur les unités de discours. Une telle relation correspond en fait soit à une commande du script, soit au résultat d'un axiome (e.g. *déduction, conclusion...*). Le mécanisme est illustré dans le schéma ci-dessous.



Un résultat de segmentation apparaît dans la disposition de la figure 3 (colonne de gauche), où chaque ligne correspond à un segment de discours. Comme annoncé *supra*, notre futur plan de discours sera formalisé dans le cadre de la SDRT, i.e. sous forme d'une SDRS (*Segmented Discourse Representation Structure*) (Asher & Lascarides, 2003). Une SDRS est une structure dans laquelle des constituants de discours sont connectés par des relations rhétoriques. Les constituants sont des représentations sémantiques dynamiques héritées de la DRT, à savoir des DRS. Les segments de la figure 3 sont donc destinés à être traduits en DRS, et en accord avec (Danlos *et al.*, 2001), cette opération se fait conjointement à la sélection des relations rhétoriques.

[7]Nous n'allons nous intéresser ici qu'au cas d'un seul graphe connexe; si nous avons un graphe composé de plusieurs parties non connexes, le module DocStruct les traitera séparement. Dans notre message (Fig. 3) les parties A, B, C et D sont les représentants de sous graphes non connexes de notre entrée.

3.2 Calcul des relations rhétoriques

Le modèle proposé par (Danlos *et al.*, 2001) est *déclaratif*. L'idée est que les relations rhétoriques sont associées à des postulats de sens et que ceux-ci sont considérés comme des conditions (des déclencheurs) de sélection d'une relation valide. Les conditions sont formulées dans le même langage que celui des formes logiques d'entrée pour permettre des appariemments directs. La figure 4(a) illustre un tel appariemment avec une règle pour la relation *Resultat*, qui en SDRT permet d'exprimer la causalité entre deux phrases. La règle a la forme suivante: *conditions* $\rightarrow$ SDRS. La SDRS de la partie droite est une portion de discours dans laquelle est instanciée la relation rhétorique déclenchée. La fonction `Calculer_SDRS` sera explicitée en § 3.3; elle fait partie de la procédure de structuration et permet de construire le discours récursivement. La SDRS obtenue en Fig. 4(a) pourra être générée en « Phrase$_1$. *Donc* Phrase$_2$. » Notons que comme dans (Danlos *et al.*, 2001), une condition comme *implies*(e_1, e_2) peut donner lieu à autre chose qu'une relation rhétorique, par exemple un prédicat (« verbal ») qui fonde une DRS (« X_1 implique X_2. », « de X_1 on obtient X_2. » ou « X_2 se déduit de X_1 »).

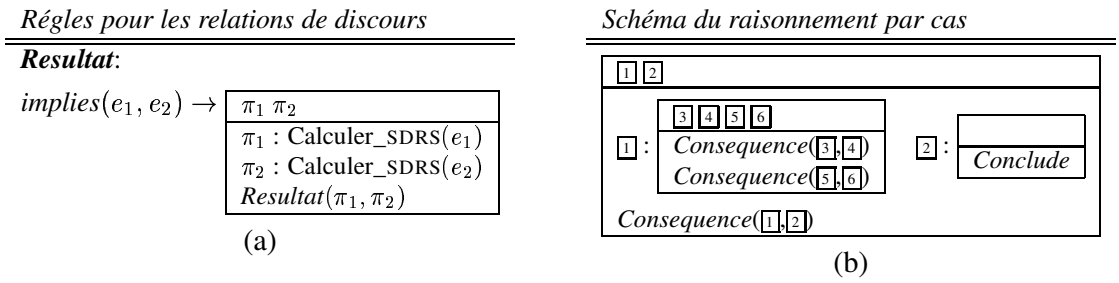


Figure 4: Règles et Schémas de Structuration Rhétorique

Cependant notre stratégie se distingue de (Danlos *et al.*, 2001) sur plusieurs points. D'abord les formes logiques de notre entrée comportent des informations qui peuvent refléter des intentions communicatives. Cela permet de sélectionner dans un plus large éventail de relations rhétoriques, notamment en faisant varier les forces illocutoires. Nous reprenons en effet l'hypothèse de (Asher & Lascarides, 2003) selon laquelle les arguments des relations rhétoriques sont des actes de langage et que les relations induisent des typages illocutoires. Il s'agit ici pour nous de traduire des « actes de démonstration » (les commandes saisies dans PHOX) en actes de langage. Les règles sont similaires à celles de Fig. 4(a) mais les relations déclenchées pourront spécifier une force illocutoire particulière. Par exemple, une condition comme *choose*(n) pourra ainsi donner lieu à une assertion (« on choisit n... ») ou un impératif (« soit n... »). Nous postulons aussi qu'un impératif déclenche une relation qui a portée sur tout le reste du discours[8]; en l'occurence, pour les textes de preuve, nous posons la relation *Background*$_i$ pour « impératif d'arrière-plan » (cf. § 3.4).

Une seconde particularité de notre approche tient au traitement des annonces de stratégies de raisonnement mentionnées dans la forme logique et issues du script de preuve. Une telle annonce ne déclenche pas directement une relation rhétorique, mais un *schéma* rhétorique propre au raisonnement en question. Un schéma est en fait une structure contenant plusieurs relations. Par exemple, le schéma du raisonnement par cas est illustré en Fig. 4(b). Ce type de traitement s'explique par le fait que ces stratégies de raisonnement s'appuient sur des théorèmes fondamentaux (e.g. le tiers exclu pour le raisonnement par cas) qui en soi ne sont jamais explicités

[8]Ce que nous ne démontrerons pas ici pour des raisons de place.

dans la démonstration, mais qui en revanche impliquent des agencements discursifs bien précis (e.g. une suite de « *si*..., *alors*... » dans le raisonnement par cas). Un schéma ainsi enclenché ne contient qu'un squelette rhétorique, qui sera paramétré ensuite en fonction du contenu de la forme logique (cf. § 3.3).

3.3 Algorithme

Nous présentons ici l'algorithme par lequel GEPHOX met en œuvre les règles de déclenchement des relations rhétoriques. L'algorithme prend en entrée une liste de *formes logiques* pour produire un plan de document : une SDRS. Il est incrémental, et réalise deux traitements différents : l'un pour l'activation des schémas rhétoriques, c'est-à-dire les SDRS complexes déclenchées par des stratégies de raisonnement ou des conditions intentionnelles, le second implémentant un traitement à la (Danlos *et al.*, 2001) pour les règles simples.

`Calculer_SDRS``(listeFL)`
- **Pour** C_i dans *listeFL*
  - **Si** C_i contient une stratégie ou des conditions intentionnelles S_i
  - **Alors**
    - SCH_i = **`Declencher_schema`**(S_i)
    - $L_{i,j}$ = **`ListeElements`**(SCH_i)
    - **Pour** tout j
      - $SDRS_{i,j}$ = **`Calculer_SDRS`**$(L_{i,j})$
      - $SDRS_i$ = **`Saturer_Schema`**$(SCH_i, \{SDRS_{i,j}\}_j)$
  - **Sinon**
    - $SDRS_i$ = **`Calculer_DRS`**(C_i)
  - **`Augmenter_Context`**$(SDRS_i)$
  - **Retourner** $SDRS_i$

`Calculer_DRS``(C)`
- – choisir les réferents de discours
- – resoudre les équations anaph

`Augmenter_Context``(D)`
- **`Ajouter_Context`**(D)
- **`Calculer_DiscRel`**(D)

`Calculer_DiscRel``(D)`
- – calcul des relations réthoriques candidates et choisir la plus adéquates

`Saturer_schema``(SCH`$_i$`,{SDRSi,j}`$_j$`)`
- – placer les $SDRS_{i,j}$ dans SCH_i
- – vérifier les relations de discours

Figure 5: Algorithme de structuration de document

3.4 Exemple

L'algorithme de structuration de document appliqué au message[9] à la figure 3, donne, parmi les solutions possibles, le plan de document représenté à la figure 6.

Le plan de document est donné comme entrée au module *Comment-le-dire ?*, dans GEPHOX ce module est réalisé par un instantiation de la platforme CLEF (Meunier & Reyes, 1999) avec

[9]Nous détaillons uniquement la partie (D) du message qui correspond à la preuve du théorème proprement dite.

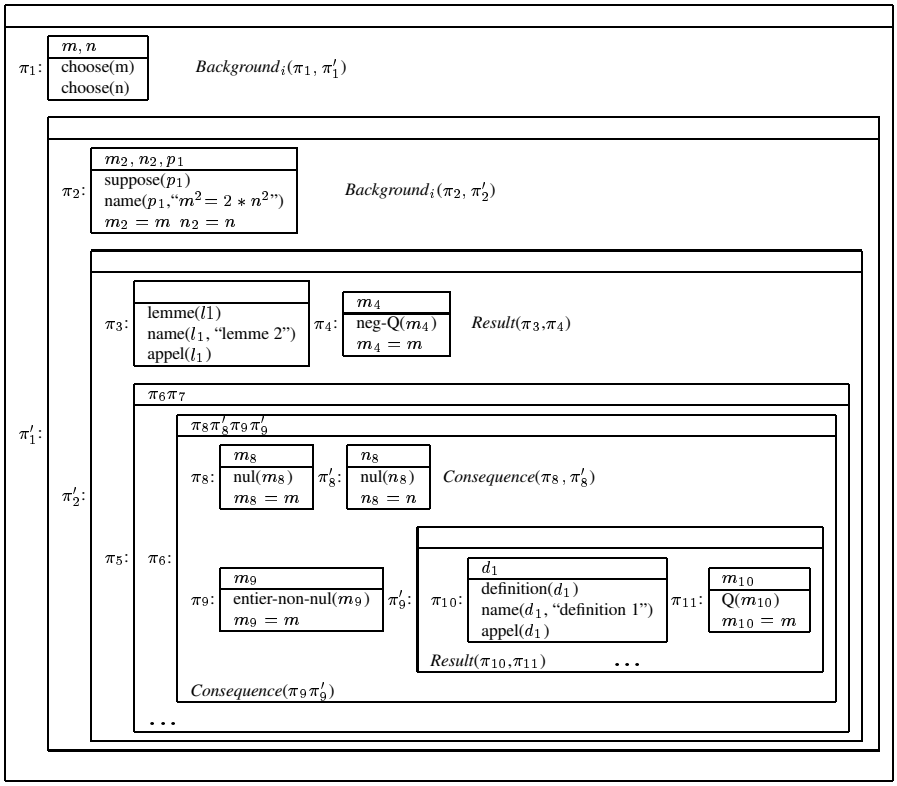


Figure 6: Plan de document possible pour «$\sqrt{2}$ *n'est pas rationnel* »

un dictionnaire associé aux concepts et relations du domain, et une grammaire G-TAG (Danlos, 1998) du français légèrement modifiée pour prendre en compte les spécificités des textes mathématiques. Le texte ci-dessous illustre le type de sortie prévu.

A	**Définition 1** Nous définissons Q comme un sous-ensemble de $\mathbb{N}^*$ tel que $\forall m \in Q$ on a $\exists n \in \mathbb{N} \quad m^2 = 2 * n^2$
B	**Lemme 2** *Pour tout* $m \in \mathbb{N}$ *on a* $\neg(Qm)$
C	**Lemme 3** *si* $\forall m, n \in \mathbb{N}\ (m^2 = 2 * n^2 \rightarrow m = 0 \wedge n = 0)$ *alors* $\sqrt{2} \notin \mathbb{Q}$
D	**Théorème** $\sqrt{2} \notin \mathbb{Q}$ *Proof* Soit $n, m \in \mathbb{N}$, supposons que $m^2 = 2 * n^2$ par le *lemme 2* nous avons $\neg(Qm)$, si $m = 0$ nous obtenons facilement $n = 0$; si $m > 0$ alors nous avons (Qm) par définition de Q et donc une contradiction. donc $\forall m, n \in \mathbb{N}\ (m^2 = 2 * n^2 \rightarrow m = 0 \wedge n = 0)$, d'aprés le *lemme 3* nous avons donc $\sqrt{2} \notin \mathbb{Q}$

4 Conclusion

Le modèle de génération profonde de GEPHOX propose une stratégie de planification originale sur au moins deux aspects. D'abord, par rapport aux générateurs automatiques de textes de preuves déjà existants, les (plans de) discours ici produits s'annoncent plus naturels et plus proches des preuves rédigées manuellement. Cela est dû à la spécificité du module `ContDet` qui exploite avantageusement les sorties du prouveur, notamment en dégageant des informations de nature intentionnelle (e.g. les commandes du script). Par ailleurs, si l'utilisation d'intentions

n'est pas neuve en génération (e.g. cf. (Moore & Paris, 1993)), la mise en œuvre que nous présentons se distingue par la simplicité du traitement. En effet l'approche est hybride en ce sens que les conditions intentionnelles et les condition informationnelles (sémantiques) de l'entrée donnent toutes des structures de discours de même types: des (portions de) SDRS. Celles-ci peuvent ensuite s'assembler selon un procédé unique (qui dépend seulement des contraintes de bonne formation stipulées en SDRT). Cela permet, entre autres, de s'affranchir de la complexité, souvent mentionnée, d'une gestion séparée des buts communicatifs et de leurs interactions parfois discordantes avec les structures rhétoriques.

Références

ASHER N. & LASCARIDES A. (2003). *Logics of Conversation*. Cambridge: CUP. (à paraître).

BAADER F. & KÜSTERS R. (2001). Unification in a Description Logic with Transitive Closure of Roles. In R. NIEUWENHUIS & A. VORONKOV, Eds., *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*, volume 2250 of *Lecture Notes in Artificial Intelligence*, Vienna, Austria: Springer–Verlag.

F. BAADER, D. L. MCGUINNESS, D. NARDI & P. F. P.-S. HNEIDER, Eds. (2003). *Description Logics Handbook: Theory, Implementation a nd Applications*. Cambridge University Press.

DANLOS L. (1998). G-TAG : un formalisme lexicalisé pour la génération de textes inspiré de TAG. *Revue T.A.L.*, **39**(2), 7–33.

DANLOS L., GAIFFE B. & ROUSSARIE L. (2001). Document structuring à la SDRT. In *Proceedings of the 8th European Workshop on Natural Language Generation (EWNLG'2001)*, p. 11–20, Toulouse.

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

HUANG X. & FIEDLER A. (1997). Proof verbalization as an application of NLG. In *IJCAI'97 Proceeding (2)*, p. 965–972.

MEUNIER F. & REYES R. (1999). La plate forme de développement de générateurs de textes CLEF. In *Actes du 2è Colloque Francophone sur la GAT, GAT'99*, Grenoble.

MOORE J. D. & PARIS C. L. (1993). Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics*, **19**(4), 651–694.

RAFFALLI C. & DAVID R. (2001). Apprentissage du raisonnement assité par ordinateur,. *Á paraître dans quadrature*.

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. CUP.

ZINN C. (1999). Understanding mathematical discourse. In *Proceedings of Amstelogue'99, 3rd Workshop on the Semantics and Pragmatics of Dialogue*, Amsterdam.