Computing the rhetoric of text proofs

Adil El Ghali

Lattice – PPS
Université Paris 7
adil@linguist.jussieu.fr

Laurent Roussarie

Lattice – CNRS
Université Paris 7
laurent@linguist.jussieu.fr

Abstract

In this paper, we present a document structuring model for mathematical proof text generation. This model takes as input the output of a *Proof assistant* and produces document plans from which we can generate text proofs with a structure approaching as much as possible human-written proofs. In order to do this, our model has to abstract away from the purely logical proof structure. Our solution is to integrate some intentional information to enrich the rhetorical structure of the final text.

1 Introduction and motivation

In this communication, we wish to address a specific application of a theorem prover in computational linguistics, namely the natural language generation of mathematical proofs. More precisely, the issue we are concerned with is: how a structured computer-generated proof can be translated into a discourse structure approaching as much as possible human-written proofs. This raises the problem of the one-to-many correspondences between the global logical structure of a text and the linguistics (viz. semantic and rhetorical) means one can use to express this structure.

It is well known that one of the main weak points of natural language generation (nlg) is the definition of input formats. Input formats are usually tightly constrained by the domain and/or the application to which the generator is connected. And this often leads to adapting very customised strategies for the deep generation process, which seriously slows down the research in nlg, as soon as one attempts to see it as a genuine branch of computational semantics.

Plugging an NLG system to a theorem prover is scientifically interesting because the output of the prover is expected to be encoded in a non *ad hoc* language (i.e. logical formulae) which is moreover very close the language of linguistic meaning representation in the frameworks of formal model-theoretic semantics. Indeed, a mathematical proof has a stable and a well defined formal structure: a natural deduction proof is, for example, a hierarchy of elimination and introduction rules application, that we call a proof tree. From a semantic point of view, such a structure can be interpreted as a sequence of inferences (which may be embedded). Therefore, we can intuitively assume that the information given by the proof tree, gives us not only the semantic content of the proof text, but also some information about its argumentative and rhetorical structure. This point has been made and used in number of previous works in natural language proof generation ([8,7]) which elaborate proof tree exploration strategies. Those strategies attempt to build a document plan by arranging and grouping portions of the proof tree. Now, the texts produced this way are correct and close to formal proof, but they are stereotyped, unnatural, fastidious and extremely hard to read. We think that a possible explanation of those drawbacks is the difficulty to recognise what should be done by content determination module *vs.* document structuring module. Moreover, even though the proof tree structure constrains the rhetorical structure, it does not fully define it. Empirically we notice that a proof text reports not only the proof tree structure but also the reasoning mechanism used in the proof.

In this paper, we present an original strategy of text proof planning, which attempts to produce more natural and closer to hand-written text proofs. The broad outline of our strategy is detailed through the presentation of the GEPHOX system [5], which is a natural language proof generation system, the formal proofs from which we generate the text being obtained with the proof assistant PHOX [12]. GEPHOX is intended to help mathematicians using PHOX; and also to be used for computer assisted logic teaching [11].

We focus on the generator *What-to-say?* module, whose organisation follows the standard architecture [13] in two sub-modules (figure 1) :

`ContDet` in charge of computing the content to be expressed from the PHOX output, this calculus takes into account the user [1] knowledge,

`DocStruct` in charge of discourse plan calculus.

The discourse plans presented here are SDRS, following the SDRT formalism [2,1]. This choice is doubly motivated: on the one hand our model is based on the document structuring model in [6], which shows the effectiveness of SDRT for deep generation; and on the other hand we adopt the conclusions reached by [14], namely that DRT (and implicitly SDRT) offers a particularly adapted formalism for analysing and representing mathematical texts.

[1] We designate by *user* the GEPHOX generated text recipient, and by *redactor* the PHOX user

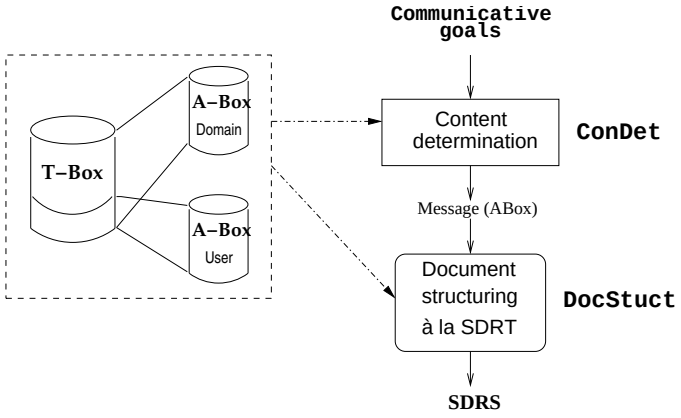


Fig. 1. *What-to-say?* module architecture

2 Content determination : the `ContDet` module

The proof assistant PhoX allows one to construct mathematical proofs on the computer, and guarantee the validity of the written proofs. Even if the software has an automated deduction capability, the main use of PhoX – by mathematicians and especially for teaching mathematics – is to check that each step of the proof is correct. To build a proof, the *redactor* guides the computer in the reasoning, and the software deals with the verifications and the fastidious steps of the proof.

2.1 GePhoX *input*

GePhoX takes as input two kinds of informations: the *proof script* representing the commands entered by *redactor*, which is also a *trace* of the proof, and the *PhoX output* formed by fragments of the *proof tree*, which can be taken as the *proof content*.

The `ContDet` module has to compute from the input, taking into account the user knowledge, the message that will be expressed by the generator. The figure 2 illustrates the treated input. One of the peculiarities of GePhoX is that the text produced has to follow as much as possible the structure encoded in the *proof script*.

2.2 Knowledge bases

In GePhoX, the input is represented with *concepts* and *roles* of a description logic (DL, cf.[4]). Domain and user specific knowledge are respectively encoded in two knowledge bases: DKB and UKB (where UKB is a sub-set of DKB). Those knowledge bases consist of two different parts. On the one hand, the **T-Box** (for *t*erminological knowledge) encodes intensional knowledge, *i.e.* domain (resp. user) *concepts* and *roles*. For example, the concept `Entier` describes the set of natural numbers, in addition, the axioms are used to represent reasonning strategies.

Proof script	*PhoX output*
def $Q\ m = m > 0 \wedge \exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$.	$\mathbf{Q} = \lambda m\ (m > 0 \wedge \exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$: 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 H2 := m = 0 ⊢ 0 = 0 ∧ n = 0 *goal* 2/2 H2 := y ∈ ℕ H3 := m = S y ⊢ S y = 0 ∧n = 0
⋮	⋮

Fig. 2. Fragment of GePhoX input for «$\sqrt{2}$ *is ir rational* »

On the other hand, the **A-Box** (for *a*ssertional knowledge) encodes extensional knowledge, *i.e.* the *individuals* of our universe. For instance, $n \in$ `Entier` introduces an individual n belonging to the concept `Entier`.

2.3 Content calculus

The general principle of content calculus is the following. We must first build the conceptual expressions (**T-Box**) representing the input. While doing this, we have to keep in mind a set $\mathcal{I}$ of individuals to which they are associated. The content determination operates on this **T-Box** [2].

Finally, the conceptual expressions are instanciated using the individuals in $\mathcal{I}$ to produce an **A-Box** (a set of facts) representing the message to be generated.

The `ContDet` module starts with the *translation* task, meaning that the input of the generator is translated into DL, using the concepts and roles of DKB. If the definitions and theorems resulting from PhoX are not present in the knowledge base, the translation procedure is able to dynamically create new concepts using the concepts and roles which are known, and axioms corresponding to definitions and theorems. For example, the definition *def* $Q\ m = m > 0 \wedge \exists n \in \mathbb{N}\ (m^2 = 2 * n^2)$ will give us the concept `Q` defined by: $\texttt{Q} \doteq \texttt{Ensemble} \wedge \exists \texttt{sous-ensemble}.\{\mathbb{N}^*\} \wedge \exists \texttt{eq-def}.\{\exists n \in \mathbb{N}\ (m^2 = 2 * n^2)\}$ [3].

Once we have built the set $\mathcal{EC}_D$ of conceptual expressions representing our input, the `ContDet` module will *select* in this set *what we must say*.

To do this we use filtering and grouping operations, to decide on the selection. The first operation consists in detecting the reasoning strategies and their

[2] Two reasons motivate this choice: the first one is technical, in DL the reasoning is easier on **T-Box**. The second, more theoretical, is due to the use of the conceptual expressions in the lexicalization process, but this does not fall within the scope of the present paper.

[3] $\{c\}$ represents the concept containing only the individual c.

parameters: for instance, recognizing that the command "*elim* H with **[case]**" corresponds to the announcement of a case reasoning, the system computes the different cases and associates them with this announcement.

The second operation is to highlight the similarities between different steps in the proof. For this we use concept unification [3].

Finally, `ContDet` attempts to match the complex concept definitions in the DKB with the expressions in $\mathcal{EC}_D$. This allows one to synthesize sets of more or less simple information to be associated with a predefined concept in the knowledge base (this operation is generally named *aggregation* in natural language generation).

The axioms (i.e. definitions in the DKB) used for this task are subsequently kept in mind to calculate second order relations on the elements of $\mathcal{EC}_D$.

The obtained **T-Box** gives a proof representation using all the concepts of the domain. To produce cooperative and personalized texts, we need to compare this representaion with the user knowledge (UKB), in order to make sure that he/she can understand the content of $\mathcal{EC}_D$. This comes down to *explaining* (i.e. decomposing) the $\mathcal{EC}_D$ concepts which are not present in the UKB. This gives us a new $\mathcal{EC}_U$ conceptual expression set, which is calculated by *projection* of $\mathcal{EC}_D$ in the UKB.

The last step of content determination is to instanciate the expressions in $\mathcal{EC}_U$ with the individuals in $\mathcal{I}$ and to verify the consistency of the obtained **A-Box** fragment.

3 Document structuring

The figure 3 gives an example of the `ContDet` output [4]. This kind of structure can be seen as an *ordered* [5] set of logical forms marking the significant steps of the proof.

The axioms that allow us to build certain **A-Box**s are given in the second column. Moreover, the given structure is richer than a simple proof tree, since the `ContDet` module ensures that they contain some elements of information from the proof script.

A central point in our deep generation strategy consists in exploiting this richness, considering that the proof script steps translated into logical forms can be assimilated to *communicative intentions*. Those intentions allow us to generate speech acts which " humanise " the proof text, by including in it some rhetorical elements other than the usual logical relations.

The document plan calculus is taken into account by the `DocStruct` module. We adopt here the document structuring model proposed by [6]. The `DocStruct` module is in charge of two main tasks in order to produce a doc-

[4] The notation $p_1 = Prop(\text{“}m^2 = ...\text{”})$ is a shortcut for $\text{Prop}(p_1) \wedge \text{constant}(p_1, \text{“}m^2 = ...\text{”})$.
[5] The order is imposed by the semantics of the dynamic conjunction $\wedge$ which is not symmetrical. In the figure 3, the enumeration of logical forms is given to improve readability.

	*Logical forms (**A Box**)*	*Axioms*
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) ...	// *the redactor start a case reasoning* ...

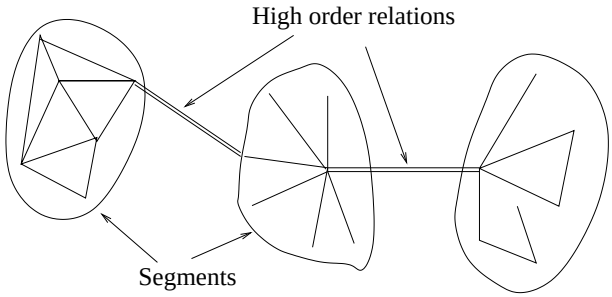
Fig. 3. Fragment of the message for «$\sqrt{2}$ *is irrational* » (ContDet Output)

ument structure: 1) choosing the minimal structural units, 2) choosing the rhetorical relations linking the units among them, to ensure discourse cohesion and coherence.

3.1 Segmentation of the logical forms set

The **A-Box**[6] structure calculated by the ContDet module can be viewed as a connected[7] graph (the nodes describe individuals and the edges the conceptual roles or relations). In this graph, certain subgraphs are strongly connected (pseudo-cliques). Our strategy is to consider them as the source of minimal discourse units, i.e. minimal content segments.

Furthermore, the subgraphs are connected with dependency relations that we assimilate to high order relations, bearing on the discourse units. Such a relation corresponds either to a script command or to the result of an axiom application (e.g. *deduction, conclusion...*). This mechanism is illustrated in the schema:



[6] Our logical forms are in fact a fragment of the **A-Box**

[7] We only consider here the case of a single connected graph; if we have a graph composed of several non-connected subgraphs, the DocStruct module treats them separately. In our message (figure 3) the lines A, B, C and D represent the unconnected subgraphs of our input.

The segmentation result is given in the left column of the figure 3, in which each line corresponds to a discourse segment.

As previously stated, our future document plan will be formalised in SDRT, i.e. it will be a SDRS (*Segmented Discourse Representation Structure*) [2]. An SDRS is a structure in which discourse constituents are connected by rhetorical relations. The constituents are dynamic semantic representations inherited from the DRT, namely DRSs. The segments in figure 3 will be translated into DRSs, and according to [6], this operation will take place at the same time with the rhetorical relations selection.

3.2 Rhetorical relations calculus

The model proposed by [6] is *declarative*. The main idea is that rhetorical relations are associated to meaning postulates which in turn are considered as conditions (triggers) of selecting a valid relation. The conditions are formulated in the same language as input logical forms in order to allow direct associations.

The figure 4(a) illustrates such an association with a *Resultat* rule, which in SDRT expresses a causality relation between two sentences. The rule has the following form: *conditions* $\rightarrow$ SDRS. The SDRS on the right-hand side is a discourse component in which the rhetorical relation triggered is instantiated. The function `Calculate_SDRS` will be developed in § 3.3; it is part of the structuring procedure and allows recursive construction of discourse. The SDRS obtained in figure 4(a) can produce the text [8] : " Sentence$_1$. *Donc* Sentence$_2$. " [9]. Let us mention that, following [6], a condition such as *implies*(e_1, e_2) is not necessarily associated with a rhetorical relation; it may for instance be associated with a predicate (" verbal ") which gives rise to an DRS (" X_1 *implique* X_2. " [10], " *de* X_1 *on obtient* X_2. " [11] or " X_2 *se déduit de* X_1 "). [12]

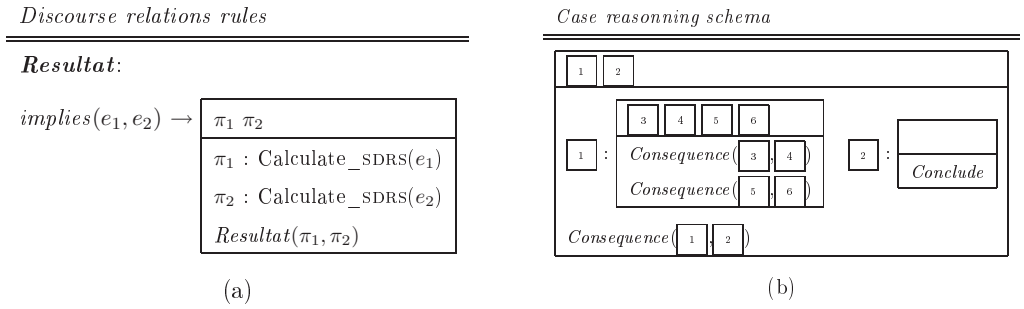


Fig. 4. Rhetorical structuring rules and schema

[8] Our system produce french texts, so
[9] " Sentence$_1$. *therefore* Sentence$_2$. "
[10] " X_1 *implies* X_2. "
[11] " *from* X_1 *we obtain* X_2. "
[12] " X_2 *is deduced from* X_1 "

However our strategy differs from [6] in many points. On the one hand, the logical forms of our document structuring input contain informations which can reflect communicative intentions. This allows us to select from a wide range of rhetorical relations, mainly by varying the illocutionary forces. We follow in this the [2] hypothesis assuming that the arguments of rhetorical relations should be speech acts tokens, and that the relations induce an illocutionary typing. In our application, we have to translate "proving acts" (the PhoX commands typed by the redactor) into speech acts. The rules are similar to those in figure 4(a) except that when a triggering condition has been computed from an operation from the proof script, it can yield an imperative clause. For instance, a condition such as $choose(n)$ can produce an indicative ("on choisit n..." [13]) or an imperative ("soit n..." [14]). Indeed, following [9], we assume that imperatives denote actions whose semantic contributions is to change the context like a (deontic) modal operator; now in text proofs, it happens that the actions denoted by imperatives usually correspond to some steps of the demonstration process.

Another particularity of our approach comes from the treatment of reasoning strategy announcements mentioned in the logical form and calculated from the proof script. Such an announcement does not directly trigger a rhetorical relation, but a rhetorical *schema* associated with the reasoning strategy. A schema is in fact a complex structure containing a certain number of relations. For example, the case reasoning schema is given in figure 4(b). Using the proof script information allows us to skip some proof portions along the line of the chosen schema to produce a coarse grained text.

This treatment can be motivated by a common property of reasoning strategies, that is their dependency on fundamental theorems (e.g. the elimination of $\vee$ for case reasoning) that are never explicited in the text proofs, but which induce precise discourse configurations (e.g. a sequence of " *si...,alors...* " [15] for case reasoning). Such a triggered schema contains just a rhetorical skeleton which will be later on shaped to fill the logical form content.

3.3 Algorithm

The algorithm takes as input a list of logical forms in order to produce the document plan: an SDRS. It works in an incremental way and achieves two different treatments: one is activates rhetorical schema, i.e. complex SDRS trigged by reasoning strategies or intentional conditions, we choose a rhetorical schema *SCH*, construct recursively the SDRSs corresponding to the elements of *SCH* and then saturate it; the second implements a declarative treatment *à la* [6] for simple rules like *Resultat* (figure 4a).

[13] "We choose n ..."
[14] "Let n ..."
[15] " *if...,then...* "

```
Calculate_SDRS(listeFL)
 for C_i in listeFL
  if C_i contains a strategy or intentional conditions S_i
  then
   choose a schema SCH_i

   for each L_{i,j} in SCH_i
    // where Li,j is an element of SCH_i
    SDRS_{i,j} = Calculate_SDRS(FL_{i,j})
     // where FL_{i,j} is the formula associated to L_{i,j}
    SDRS_i = result of filling SCH_i with {SDRS_{i,j}}_j
  else
   SDRS_i = Calculate_DRS( C_i)   // following the algorithm of [6]
  return SDRS_i
```

Applying the document structuring algorithm to the message in figure 3, produce, as a possible solution, the document plan represented in figure 5.

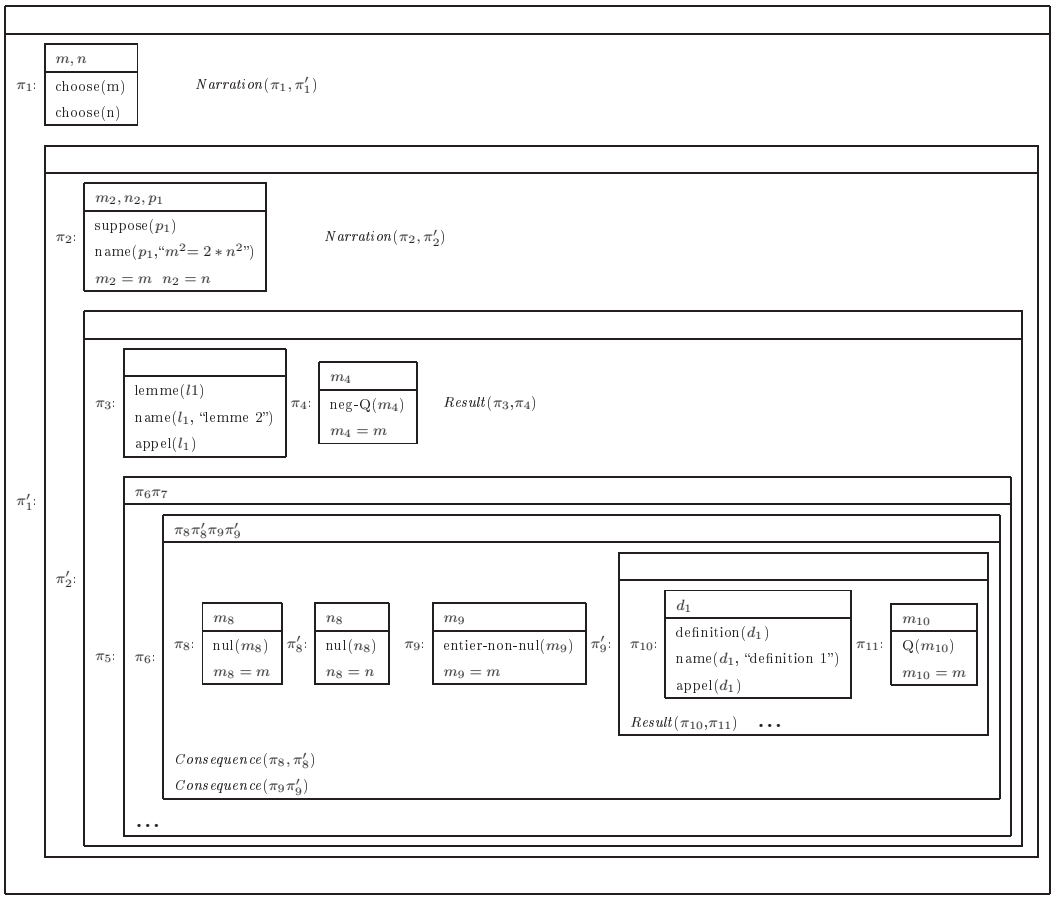


Fig. 5. A fragment of possible document plan for «$\sqrt{2}$ *is irrational* »

4 Conclusion

The deep generation model in GePhoX proposes a planning strategy which is original for at least two reasons. First, in comparison to the existing proof text automatic generators, the discourse (plans) produced by GePhoX look more natural and closer to hand-written proofs. This is due to the specificity of the content determination module `ContDet` which takes advantage of the proof assistant output, essentially by extracting informations of intentional nature (e.g. the proof script commands).

Secondly, even if use of intentions in natural language generation is not new (e.g. cf. [10]), the solution presented here, which consist in taking into account the redactor intentions, has the advantage of being simple and elegant. Actually, our approach is hybrid in that both intentional conditions and informational (semantical) conditions of the input produce the same discourse structures: fragments of SDRS. These fragments are then assembled on the basis of a unique procedure (which is only constrained by well-formedness rules in SDRT). This way, we can moreover avoid the well-known complexity of a separate treatment of communicative goals and rhetorical structures.

The text in figure 6 illustrates the kind of output produced.

Definition 1. Nous définissons Q comme un sous-ensemble de $\mathbb{N}^*$ tel que $\forall m \in Q$ on a $\exists n \in \mathbb{N} \quad m^2 = 2 * n^2$

Lemma 2. *Pour tout* $m \in \mathbb{N}$ *on a* $\neg(Qm)$

Lemma 3. *si* $\forall m, n \in \mathbb{N}\ (m^2 = 2 * n^2 \rightarrow m = 0 \wedge n = 0)$ *alors* $\sqrt{2} \notin \mathbb{Q}$

Theorem. $\sqrt{2} \notin \mathbb{Q}$

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'après le *lemme 3* nous avons donc $\sqrt{2} \notin \mathbb{Q}$

Fig. 6. A possible text for «$\sqrt{2}$ *is irrational* »

References

[1] Asher, N., "Reference to Abstract Objects in Discourse," Kluwer, Dordrecht, 1993.

[2] Asher, N. and A. Lascarides, "Logics of Conversation," Cambridge University Press, Cambridge, 2003.

[3] Baader, F. and R. Küsters, *Unification in a description logic with transitive closure of roles*, in: R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of LPAR 2001*, Lecture Notes in Artificial Intelligence **2250** (2001).

[4] Baader, F., D. L. McGuinness, D. Nardi and P. F. Patel-Schneider, editors, "Description Logics Handbook: Theory, Implementation and Applications," Cambridge University Press, 2003.

[5] Danlos, L. and A. El Ghali, *A complete integrated NLG system using AI and NLU tools*, in: *Proceeding of COLING'2002, Taipei, Taiwan*, 2002.

[6] Danlos, L., B. Gaiffe and L. Roussarie, *Document structuring à la SDRT*, in: *Proceedings EWNLG'2001*, Toulouse, 2001, pp. 11–20.

[7] Fiedler, A., "User adaptive proof explanation," Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany (2001).

[8] Huang, X. and A. Fiedler, *Proof verbalization as an application of NLG*, in: *IJCAI'97 Proceeding (2)*, 1997, pp. 965–972.

[9] Lascarides, A., *Imperatives in dialogue*, in: *Proceedings of the 5th International Workshop on Formal Semantics and Pragmatics of Dialogue (BI-DIALOG 2001)*, Bielefeld Germany, 2001, pp. 1–16.

[10] Moore, J. D. and C. L. Paris, *Planning text for advisory dialogues: Capturing intentional and rhetorical information*, Computational Linguistics **19** (1993), pp. 651–694.

[11] Raffalli, C. and R. David, *Computer assisted teaching in mathematics*, in: *to appear in the proceedings of the Workshop on 35 years of Automath*, 2002.

[12] Raffalli, C. and P. Roziere, "The PhoX Proof checker documentation," LAMA, Université de Savoie / Université Paris 7 (2002).

[13] Reiter, E. and R. Dale, "Building Natural Language Generation Systems," Studies in Natural Language Processing, Cambridge University Press, 2000.

[14] Zinn, C., *Understanding mathematical discourse*, in: *Proceedings of Amstelogue'99, 3rd Workshop on the Semantics and Pragmatics of Dialogue*, Amsterdam, 1999.