A complete integrated NLG system using AI and NLU tools

Laurence Danlos
Lattice
U. Paris 7, Case 7003
2, place Jussieu
75251 Paris Cedex 05
France
danlos@linguist.jussieu.fr

Adil El Ghali
Lattice
U. Paris 7, Case 7003
2, place Jussieu
75251 Paris Cedex 05
France
adil@linguist.jussieu.fr

Abstract

A standard architecture for an NLG system has been defined in (Reiter and Dale, 2000). Their work describes the modularization of an NLG system and the tasks of each module. However, they do not indicate what kind of tools can be used by each module. Nevertheless, we believe that certain tools widely used by the AI or NLU community are appropriate for NLG tasks. This paper presents a complete integrated NLG system which uses a Description logic for the content determination module, Segmented Discourse Representation Theory for the document structuring module and a lexicalized formalism for the tactical component. The NLG system, which takes into account a user model, is illustrated with a generator which produces texts explaining the steps taken by a proof assistant.

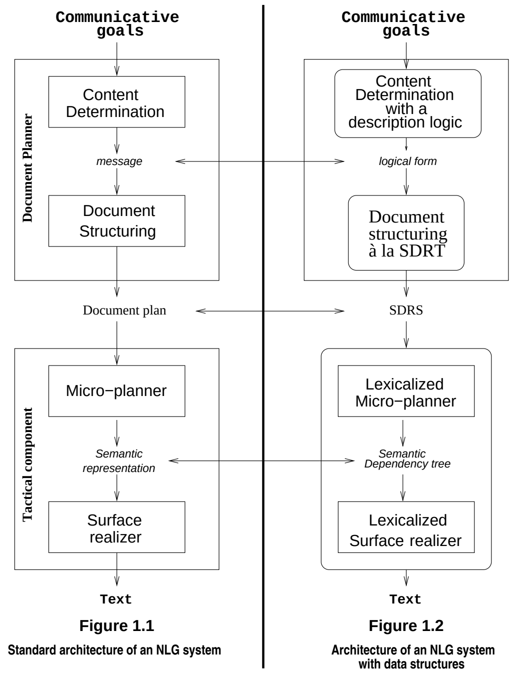


Figure 1.1 Standard architecture of an NLG system

Figure 1.2 Architecture of an NLG system with data structures

1 Introduction

The standard architecture of an NLG system proposed in (Reiter and Dale, 2000) is represented schematically in Figure 1.1. The tool used by a module and the data structure of its output are not precisely defined. According to Reiter and Dale, they vary from one author to the other. However, we believe that certain tools widely used by the AI or NLU community are appropriate for NLG tasks. Therefore, we reformulate in more specific terms Figure 1.1 as Figure 1.2.

The paper describes the modules of Figure 1.2: Section 3 justifies the use of a description logic for the content determination task and its ouput, a “message”; Section 4 specifies the use of SDRT for the document structuring task and its output, a “document plan”; Section 5 exposes briefly the use of a lexicalized formalism in the tactical component. Each section is illustrated with GePhoX, a generator that produces texts which explain the steps taken by PhoX , a proof assistant (Raffalli and Roziere, 2002). GePhoX is presented in Section 2.

As this paper intends to present a complete NLG system, there is no room for explaining each module in detail. We refer the reader to (El Ghali, 2001) for the content determination module, to (Danlos et al., 2001) for the document structuring module and to Danlos (1998; 2000) for the lexicalized tactical component. These

```
goal ∀p, d : N(d ≠ N0 → ∃q, r : N(r < d ∧ p = q ∗ d + r))
1. intros.
2. elim −4 H well_founded.N.
3. intros.
4. elim −1 d −3 a lesseq.case1.N.
5. next.
6. intros ∃ ∧.
7. next −3.
8. instance ?1 N0.
9. instance ?2 a.
10. intro.
11. trivial.
12. local a′ = a - d.
13. elim −1 a′ H3.
14. trivial.
15. elim lesseq.S_rsub.N.
16. elim −1 [case] H0.
17. trivial =H1 H5.
18. trivial.
19. lefts H5 ∧ ∃.
20. intros ∧ ∃.
21. next −3.
22. instance ?4 r.
23. instance ?3 S q.
24. rewrite mul.lS.N −r add.associative.N −r H8.
25. intro.
26. trivial.
27. save euclide_exists.
```

Table 1: *Proof script* for Euclidian division

```
Here is the goal:
goal 1/1
   |- /\p,d:N  (d != N0 ->
        \/q,r:N  (r < d & p = q * d + r))

End of goals.
%PhoX% intros.
1 goal created.

New goal is:
goal 1/1
H := N p
H0 := N d
H1 := d != N0
   |- \/q,r:N  (r < d & p = q * d + r)

End of goals.

. . .
```

Table 2: *Proof tree* for Euclidian division

modules have been built more or less independently from each other, but with the same underlying idea: adaptation of NLU/AI theories and tools to NLG. They are now integrated in a complete model, which is presented here and illustrated with GePhoX, a generator whose implementation is in progress.

2 GePhoX

PhoX is an extensible proof assistant based on higher order logic, which has been developped to help mathematicians building proofs and teaching mathematics. Like other proof assistants, PhoX works interactively. The user (a mathematician) first gives the theorem to be proven (a goal). PhoX returns a list of subgoals which should be easier to prove than the initial goal. The user enters a command to guide PhoX in choosing or achieving a subgoal. The proof is thus computed top-down from *goals* to *evidences*. The user's commands form a *Proof script*. PhoX's output is a list of successive goals equivalent to a *Proof tree*.

Both the *Proof script* and PhoX's output are difficult to read (even for a mathematician), as the reader can see for himself in Table 1 and Table 2. Hence, the need of an NLG system in order to obtain an easy-to-read version of the proof.

GePhoX is given as input both the *Proof script* and the *Proof tree*. It is one of the main original proposals in our generator (similar generators, such as PROVERB (Huang and Fiedler, 1997), take as input only the *Proof tree*). It makes it possible for GePhoX to start from an incomplete proof and produce texts during the interactive session. These texts help the mathematician user: before entering a new command in the *Proof script*, he can read a text reminding himself what he has been doing so far. The *Proof script* is also useful for identifying the reasoning strategies that have been used (reasoning by contradiction or induction), while it is very hard (if not impossible) to retrieve this information from a *Proof tree* with its numerous deduction steps.

Another originality of GePhoX is that it takes into account the knowledge of the user who can be either a mathematician using PhoX or a person more or less novice in mathematics. For the same proof, GePhoX can generate several texts according to the user model.

3 Using a descrition logic (DL)

The knowledge representation system Kl-One (Branchman et al., 1979) was the first DL. It was created to formalize semantic networks and frames (Minsky, 1974). It introduces the notions of TBoxes and ABoxes respectively for terminological and assertional knowledge. Kl-One has been widely used in the NLG community to formalize the domain model. On the other hand, this is not the case for the more recent DLs. Nevertheless, they present at least two advantages compared to Kl-One : 1) for a large variety of DLs, sound and complete algorithms have been developped for main inference problems such as *subsumption*, *concepts satisfiability* and *consistency* (Donini et al., 1996); 2) the relations between instances and classes are well defined for all the constructors, and their mathematical and computational properties have been studied in detail (Horrocks et al., 2000). So we believe that DLs are appropriate for the content determination task as shown in 3.3. Let us first briefly present DLs.

3.1 A brief introduction to DL

The three fundamental notions of DLs are *individuals* (representing objects in the domain), *concepts* (describing sets of individuals), and *roles* (representing binary relations between individuals or concepts). A DL is characterized by a set of *constructors* that allow us to build complex concepts/roles from atomic ones. The set of constructors which seem useful for GePhoX and their syntax are shown in Table 3; examples of concepts and roles with their semantic are shown underneath Table 3.

Constructor (abbreviation)	**Syntax**
atomic concept	A
top	$\top$
bottom	$\bot$
conjonction	$C \wedge D$
disjonction ($\mathcal{U}$)	$C \vee D$
complement ($\mathcal{C}$)	$\neg C$
univ. quant.	$\forall R.C$
exist. quant. ($\mathcal{E}$)	$\exists R.C$
numeral restrictions ($\mathcal{N}$)	$>n\ R.C$
	$\leq n\ R.C$
collection of individuals ($\mathcal{O}$)	$\{a_1,\ldots,a_n\}$
atomic role	P
roles conjonction ($\mathcal{R}$)	$Q \wedge R$
inverse role	R^{-1}
role composition	Q $\circ$ R

Table 3: Syntax of standard constructors

Examples of concepts with their semantic
`Theorem`, `Variable`, $\{H_1\}$, $\exists$`CHOOSE.User`
{ x / `Theorem`(x) } : Theorem concept
{ x / `Variable`(x) } : Variable concept
{ H_1} : concept constructed by the $\mathcal{O}$ constructor on individual H_1
{ x / $\exists$ u : `User`, `CHOOSE`(u,x) }

Examples of roles with their semantic
`IMPLIES`, `PROVES`
{ x,y / `IMPLIES`(x,y) } : x implies y
{ x,y / `PROVES`(x,y) } : x proves y

The choice of constructors is domain dependent. Constructors other than those used in GePhoX (e.g. temporal extension) can be used for other domains (e.g. domains with non trivial temporal information), without altering the mathematical and computational properties.

3.2 Domain and user models in DL

The *Domain model* is the set of concepts and roles necessary to express the input of the generator. More formally, let $\mathcal{T_D}$ be a TBox, such that each input `I` can be described by means of an ABox $\mathcal{A_D}$ corresponding to $\mathcal{T_D}$. The knowledge base $\Sigma_D = (\mathcal{T_D}, \mathcal{A_D})$ is called knowledge base for the domain and noted DKB. The *User model* is a knowledge base $\Sigma_U = (\mathcal{T_U}, \mathcal{A_U})$ such that $\mathcal{T_U}$ and $\mathcal{A_U}$ are respectivly subsets of $\mathcal{T_D}$ and $\mathcal{A_D}$. Σ_U is noted UKB. Table 4 shows a part of the DKB for GePhoX.

```
Goal                MathObj
  Subgoal               Axiom
  Hypothese             Theorem
Rules                     well_founded
   Intro                  lesseq.case1
   Elim                   add.associative
   Rewrite              Operator
   Trivial                 LogicalOper
   Left                       Exist
ReasonningStrategy            Forall
   ByInduction                LAnd
   ByContradiction         ArithOper
   . . .                      Add
   . . .                      Multi
```

Table 4: GePhoX Domain model

3.3 Content determination tasks

The content determination module performs four tasks, as shown in Figure 2.

Translation: The input of the generator (assertional information) is first translated into concepts of the TBox. For that purpose, a correspondence is established between the elements of the input and concepts and roles in the DKB. The $\mathcal{O}$ constructor is used to keep information about the individuals occurring in the input. For example, command 2 in Table 1 with individual H is translated into the concept $C_0 \doteq \exists EliminationWell\_founded.Hypothese$

$\{H\}$, and commands 8 to 11 are translated into $C_1 \doteq \exists ByInduction\ \{p\}$.

Selection: The selection task consists of choosing the relevant concepts among those constructed in the translation phase with regard to the UKB. For example, if C_0 is an unknown concept for the user, a concept C must be looked up in the UKB such as C *approximates*[1] C_0.

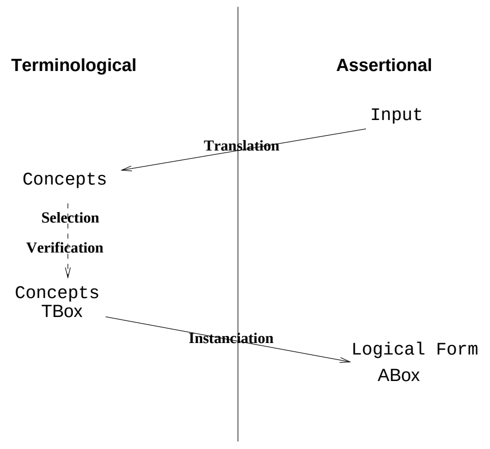


Figure 2: Content Determination Tasks

Verification: At this point, the *coherence* of all the concepts of the selection is verified. For example, if the user tries to reason by induction on a real number, GePhoX tells him that it is not possible.

Instanciation: With the information about individuals, which have been kept in the translation phase (with the use of the $\mathcal{O}$ constructor), the instanciation task is straightforward. Table 5 shows some instanciated concepts for the Euclidian division.

As is well known, designing knowledge bases (DKB and UKB) and translating the input of the generator into concepts and roles of the DL is a difficult task which has to be fulfilled for every generator. However, with a DL, the selection, verification and instanciation tasks are domain independent: algorithms and their implementation are reusable. Moreover, when using a DL for the content determination task, the "message" is a first order logic formula (a standard representation shared by a large community) which takes into account the user knowledge and whose coherence has been checked.

[1]Given two TBoxes $\mathcal{T}$ and $\mathcal{T}'$ with $\mathcal{T} \subset \mathcal{T}'$ and a concept $C \in \mathcal{T} - \mathcal{T}'$, $C' \in \mathcal{T}'$ *approximates* C if C minimally subsumes C' or C' minimally subsumes C.

```
.  $\exists p_1 \in \mathtt{Entier}$
   $named(p_1, \mathbf{p})$
   $choose(user, p_1)$
.  $\exists d_1 \in \mathtt{EntierNonNul}$
   $named(d_1, \mathbf{d})$
   $choose(user, d_1)$
.  $\exists f_1 \in \mathtt{Formula}$
   $constant(f_1, \exists \mathbf{q,r:\ N\ (r < d \wedge p = q.d + r)})$
.  $prove(user, f_1)$
   $induction(f_1, p_1)$
   ...
```

Table 5: DL-Message for Euclidian division

4 Using SDRT for document structuring

In (Danlos et al., 2001) we advocate using SDRT (Segmented Discourse Representation Theory (Asher, 1993; Asher and Lascarides, 1998)) as a discourse framework, since SDRT and DRT (Discourse Representation Theory, (Kamp and Reyle, 1993)) are the most popular frameworks for formal and computational semantics. Let us briefly present SDRT.

4.1 A brief introduction to SDRT

SDRT, designed first for text understanding, was introduced as an extension of DRT in order to account for specific properties of discourse structure. SDRT can be viewed as a super-layer on DRT whose expressiveness is enhanced by the use of discourse relations. Thus the DRT structures (Discourse Representation Structures or DRS) are handled as basic discourse units in SDRT.

DRSs are "boxed" first order logic formulae. Formally, a DRS is a couple of sets $\langle \mathtt{U,Con} \rangle$. `U` (the universe) is the set of discourse referents. `Con` contains the truth conditions representing the meaning of the discourse.

A SDRS is a pair $\langle \mathbf{U},\mathbf{Con} \rangle$, see Figure 3. **U** is a set of labels of DRS or SDRS which can be viewed as "speech act discourse referents" (Asher and Lascarides, 1998). **Con** is a set of conditions on labels of the form:

- $\pi : K$, where π is a label from **U** and K is a (S)DRS

- $R(\pi_i, \pi_j)$, where π_i and π_j are labels and R a discourse relation. Discourse relations are inferred non-monotonically by means of a defeasible glue logic exploiting lexical and world knowledge.

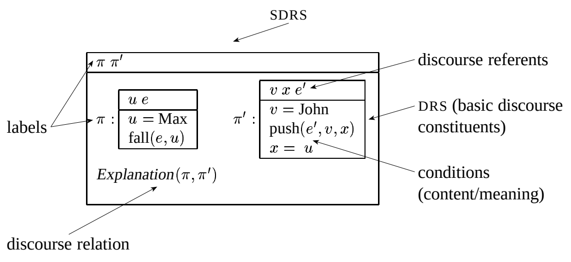
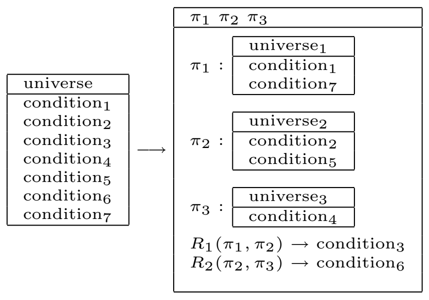


Figure 3: SDRS for *Max fell. John pushed him.*

4.2 Building a SDRS

Starting from a “message” encoded into a logical form, the document structuring module builds a SDRS. On a first step, the logical form is translated into a DRS. In the case of a purely existential formula[2], this amounts to putting all the variables into the universe of the DRS and splitting the formula into elementary conjoined conditions.

After this first step, the document structuring task amounts to building a SDRS from a DRS and to go on recursively on each embedded (S)DRS. This process is schematized below.



Let us first examine the principles governing the splitting of the conditions. All the conditions in the DRS have to be expressed in the SDRS. Two cases arise:

- either a condition in the DRS appears as a condition in one of the sub-DRS; that is the case for *condition*$_1$ which appears in the sub-DRS labelled π_1;
- or it is expressed through a discourse relation; that is the case for *condition*$_3$ with $R_1(\pi_1, \pi_2) \rightarrow$ condition$_3$, which means that $R_1(\pi_1, \pi_2)$ must have condition$_3$ among its consequences: no other element is in charge of expressing condition$_3$.

To establish discourse relations, the SDRT conditions are reversed. As an illustration, in SDRT for text understanding, there is the Axiom (1) for *Narration*. This axiom states that if *Narration* holds between two SDRSs π_1 and π_2, then the main event (me) of π_1 happens before the main event of π_2.

(1) $\Box(\mathit{Narration}(\pi_1, \pi_2) \rightarrow me(\pi_1) < me(\pi_2))$

For text generation, this axiom is reversed as shown below (Roussarie, 2000, p. 154):

- If k_1 and k_2 are DRSs whose main eventualities are not states,
- and if the main event of k_1 occurs before the main event of k_2,
- then *Narration*(π_1, π_2) is valid when π_1 and π_2 respectively label k_1 and k_2.

As another example, the condition *cause*(e_1, e_2) can be expressed through *Result*(π_1, π_2) or *Explanation*(π_2, π_1) when π_1 and π_2 label the sub-DRSs that contain the descriptions of e_1 and e_2 respectively.

Let us now examine how we determine the universes of sub-DRSs, i.e. discourse referents, while observing two technical constraints, namely:

- the arguments of any condition in a sub-DRS must appear in the universe of this DRS;
- the universes of all the sub-DRSs have to be disjoint. This constraint is the counterpart of the following constraint in understanding: “partial DRSs introduce new discourse referents” (Asher, 1993, p. 71).

These two constraints are not independent. Assuming that the first constraint is respected, the second one can be respected with the following mechanism: if a variable x already appears in a preceding sub-DRS labelled π_x, then a new variable y is created in the universe of

[2]More complex formulas are not considered here.

the current sub-DRS labelled π_y and the condition $y = x$ is added to the conditions of π_y. The discourse referent y will be generated as an anaphora if π_x is *available* to π_y (Asher, 1993), otherwise it will be generated as a definite or demonstrative NP.

A document structuring module la SDRT based on the principles we have just exposed can be used for any generator (whose "message" is first order logic formula). The algorithm and the rules establish discourse relations (obtained by reversing the rules in NLU) are generic. See below an example of SDRS in GEPHOX, the SDRS built from Table 5.

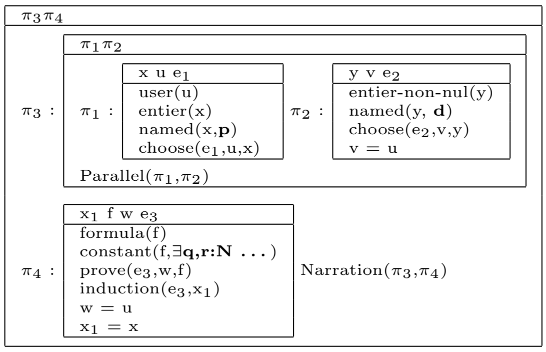


Table 6: SDRS for Euclidian division

5 Using a lexicalized grammar for the tactical component

Lexicalized grammars are commonly used in NLU and also in NLG (Stede, 1996). In Danlos (1998; 2000) we propose a lexicalized formalism, called G-TAG, for the tactical component of an NLG system. It is modularized into a microplanner which produces a semantic dependency tree and a surface realizer which produces the text (see Figure 1.2).

The surface realizer is designed to use the syntactic and lexical information of a lexicalized TAG grammar. The TAG grammar is extended to handle multi-sentential texts and not only isolated sentences.

The microplanner is based on a lexicalized conceptual-semantic interface. This interface is made up of *concepts*; each concept is associated with a lexical database. In our model, a concept is either a term in the TBox or a discourse relation. A lexical database for a given concept records the lexemes lexicalizing it with their argument structure, and the mappings between the conceptual and semantic arguments. The process of generating a semantic dependency tree from a SDRS $\langle \mathbf{U},\mathbf{Con} \rangle$ is recursive:

- An element π_i in **U** is generated as a clause if π_i labels a DRS and recursively as a text (possibly a complex sentence) if π_i labels a SDRS.
- A condition $R(\pi_i, \pi_j)$ in **Con** is generated as a text "S_i. *Cue* S_j." or as a complex sentence "S_i *Cue* S_j.", where S_i generates π_i, S_j π_j, and *Cue* is a cue phrase which is encoded in the lexical database associated with R (*Cue* may be empty).
- A condition $\pi : K$ in **Con** where K is a DRS $\langle$U,Con$\rangle$ is generated as a clause according to the following constraints (which are the counterparts of constraints in understanding):

  • A discourse referent in U is generated as an NP or a tensed verb.

  • Conditions guide lexical choices. Conditions such as x = John correspond to proper nouns. Equality conditions between discourse referents (e.g. $x = y$) give rise to (pronominal or nominal) anaphora. The other conditions, e.g. $prove(e_1, x, y)$, are lexicalized through the lexical data base associated with the concept (*prove*).

The surface realizer, based on a TAG grammar, is a set of lexical data bases. A data base for a given lexical entry encodes the syntactic structures realizing it with their syntactic arguments. With such a TAG grammar and a morphological module, the text is computed in a deterministic way from the semantic dependency tree.

6 Conclusion

Since NLG is a subfield of NLP, which is itself a subfield of AI, it seems to be a good idea to reuse tools developped by the NLP or AI community. We have shown in this paper how to integrate DL, SDRT, and a lexicalized grammar into an NLG system, while following the standard pipelined architecture[3].

[3]Some authors (de Smedt et al., 1996) have made justified criticisms of the pipelined architecture. However, we decided to keep it for the time being.

Theorem.

$$\forall p, d{:}\mathbb{N}\ (d \neq 0 \rightarrow \exists q, r{:}\mathbb{N}\ (r < d \wedge p = q.d + r))$$

Proof. Let us choose p, d two natural numbers with $d \neq 0$. By induction on p we prove $\exists q, r{:}\mathbb{N}\ (r < d \wedge p = q.d + r)$. Let take a a strictly positive natural. We assume

$$\forall b{:}\mathbb{N}\ (b < a \rightarrow \exists q, r{:}\mathbb{N}\ (r < d \wedge b = q.d + r))$$

and we must prove $\exists q, r{:}\mathbb{N}\ (r < d \wedge a = q.d + r)$. We distinguish two cases: $a < d$ and $d \leq a$. In the first case, we choose $q = 0$ and $r = a$. In the second case, we take $\mathrm{a}' = a - d$. Using the induction hypothesis on a', we find two naturals q, r such that $r < d$ and $\mathrm{a}' = q.d + r$. We take S q and r as quotient and remaining for the division of a. We must prove $a = \mathrm{S}\ q.d + r$ which is immediate. ∎

Table 7: A Text of proof for Euclidian division

GePhoX illustrates the applicabilty of our system. It is currently being implemented in Java. The development of the document planner of GePhoX is work in progress. The goal is to interface this module with CLEF (Meunier and Reyes, 1999), an implementation of G-TAG. We intend to produce a text as shown in Table 7.

References

N. Asher and A. Lascarides. 1998. The semantics and pragmatics of presupposition. *Journal of Semantics*, 15(3):239–300.

N. Asher. 1993. *Reference to Abstract Objects in Discourse.* Kluwer, Dordrecht.

R. Branchman, R. Bobrow, P. Cohen, J. Klovstad, B. Webber, and W. Woods. 1979. Research in natural language understanding. Technical Report 4274, Bolt. Beranek and Newman, Cambridge MA.

L. Danlos, B. Gaiffe, and L. Roussarie. 2001. Document structring à la sdrt. In *ACL'2001 Toulouse Proceeding.*

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

L. Danlos. 2000. G-TAG: A lexicalized formalism for text generation inspired by Tree Adjoining Grammar. In A. Abeillé and O. Rambow, editors, *Tree Adjoining Grammars: formalisms, linguistics analysis and processing*, pages 343–370. CSLI Publications, Stanford.

K. de Smedt, H. Horacek, and M. Zock. 1996. Architectures for natural language generation: Problems and perspectives. In G. Adorni and M. Zock, editors, *Trends in NLG. Proceedings of the 4th European Workshop, EWNLG'93, Pisa.* Springer-Verlag.

F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. 1996. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information. CLSI Publications.

A. El Ghali. 2001. Une logique de description pour le module *quoi-dire-?* DEA de linguistique informatique, Université Paris 7.

I. Horrocks, U. Sattler, and S. Tobies. 2000. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264.

X. Huang and A. Fiedler. 1997. Proof verbalization as an application of NLG. In *IJCAI (2)*, pages 965–972.

H. Kamp and U. Reyle. 1993. *From Discourse to Logic.* Kluwer Academic Publishers, Dordrecht, The Netherlands.

F. Meunier and R. Reyes. 1999. La plate forme de dveloppement de gnrateurs de textes CLEF. In *Actes du 2e Colloque Francophone sur la Gnation Automatique de Textes, GAT'99*, Grenoble.

M. Minsky. 1974. A framework for representing knowledge. MIT-AI Laboratory Memo 306.

C. Raffalli and P. Roziere, 2002. *The PhoX Proof checker documentation.* LAMA, Université de Savoie / Université Paris 7.

E. Reiter and R. Dale. 2000. *Building Natural Language Generation Systems.* Cambridge University Press.

L. Roussarie. 2000. *Un modèle théorique d'inférences de structures sémantiques et discursives dans le cadre de la génération automatique de textes.* Thèse de doctorat en linguistique, Université Paris 7.

M. Stede. 1996. Lexical paraphrases in multilingual sentences generation. *Machine Translation*, 11.