Use of Description logic and SDRT in an NLG system

Adil El Ghali
LATTICE
Denis Diderot University - Paris
adil@linguist.jussieu.fr

1 Introduction

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

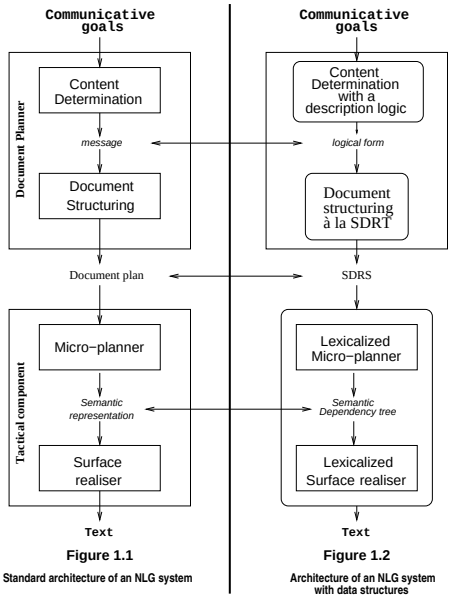


Figure 1.1
Standard architecture of an NLG system

Figure 1.2
Architecture of an NLG system with data structures

The paper follows this Figure 1.2: section 3 justifies the use of a description logic for the content determination task and its ouput, a "message"; section 4 justifies the use of SDRT for the document structuring task and its output, a "document plan"; section 5 briefly exposes the use of a lexicalized formalism in the tactical component. Each section is illustrated by means of GePhoX, a generator which produces texts explaining the steps taken by a proof assistant, PhoX. So we start by presenting GePhoX.

2 GePhoX

PhoX is an extensible proof assistant based on higher order logic, which was developped to help mathematicians building proofs and teaching mathematics (Raffalli and Roziere, 2002). Like other proof assistants, PhoX works in an interactive way. The user (a mathematician) gives first the theorem to be proved (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 output is a list of successive goals equivalent to a *Proof tree*.

Both the *Proof script* and PhoX output are difficult to read (even for a mathematician), as the reader can see for him/herself in Table 1 and Table 2. Hence, the necessity of an NLG system to get an easy to read version of the proof.

GePhoX is given as input both the *Proof script* and the *succesive goals* of PhoX output. It can produce texts during the interactive session (from an incomplete proof). This is quite useful to help the mathematician user: before entering a new command in the *Proof script*, he/she can read a text reminding him/her what he/she has been doing so far.

Taking into account the *Proof script* in GePhoX input is one of the main originalities of our generator

goal $\forall p, d : N(d \neq N0 \rightarrow \exists q, r : N(r < d \wedge p = q * d + r))$
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: PhoX output for Euclidian division

(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 to identify the reasoning strategies that have been used (reasoning by contradiction, by induction), while it is very hard (if not impossible) to retrieve this information from a *Proof tree* with its quite 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 a (GePhoX) 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 with the introduction of T-Boxes and A-Boxes (respectively for terminological and assertional knowledge). Kl-ONE has been broadly 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. So we believe that DLs are appropriate for the content determination task as shown in 3.2. Let us first present DLs briefly.

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). A *description logic* is characterized by a set of *constructors* that allow us to build complex concepts and 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 ∘ R

Table 3: Syntax of standard constructors

Examples of concepts with their semantic

`Theorem, Variable, {H1}, ∃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 / ∃ u : `User, CHOOSE`(u,x) }

Examples of roles with their semantic

`IMPLIES, PROVE`

{ x,y / `IMPLIES`(x,y) } : x implies y

{ x,y / `PROVE`(x,y) } : x prove y

Let us underline that 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. domain with non trivial temporal information), without altering the mathematical and computational properties.

3.2 Content determination in DL

The *Domain model* is the set of concepts and roles necessary to express the input of the generator. More formally, let $\mathcal{T}_\mathcal{D}$ be a TBox, such that each input `I` can be descibed by means of an ABox $\mathcal{A}_\mathcal{D}$ corresponding to $\mathcal{T}_\mathcal{D}$. The knowledge base $\Sigma_D = (\mathcal{T}_\mathcal{D}, \mathcal{A}_\mathcal{D})$ is called knowledge base for the domain (or domain model) and noted DKB. The *User model* is a knowledge base $\Sigma_U = (\mathcal{T}_\mathcal{U}, \mathcal{A}_\mathcal{U})$ such that $\mathcal{T}_\mathcal{U}$ and $\mathcal{A}_\mathcal{U}$ are respectivly subsets of $\mathcal{T}_\mathcal{D}$ and $\mathcal{A}_\mathcal{D}$. 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

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

Translation: The input of the generator (assertional information) is first translated into concepts of the TBox. For that purpose, a correspondancy between the elements of the input and concepts and roles in the DKB is established. 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 in choosing the relevant concepts among those constructed in the translation phase in regard of 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* C_0.

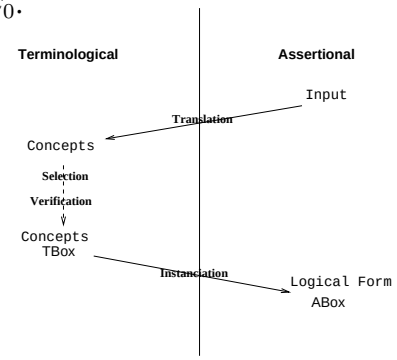


Figure 2: Content Determination Tasks

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/her that it is not possible.

Instanciation: Thanks to the informations 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.

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

Table 5: DL-Message for Euclidian division

As it is well known, designing knowledge bases (DKB and UKB) and translating the input of the generator into concepts and roles of the DL is an heavy task which has to be achieved for each 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.

4 Using SDRT for document structuring

We adopt SDRT (Segmented Discourse Representation Theory (Asher, 1993; Asher and Lascarides, 1998)). The reasons for this choice can be found in (Danlos et al., 2001). Let us present SDRT briefly.

4.1 A brief introduction to SDRT

SDRT which was designed first for text understanding, was introduced as an extension of DRT (Discourse Representation Theory, (Kamp and Reyle, 1993)) in order to account for specific properties of discourse structure. SDRT can be viewed as a superlayer 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 $\langle$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 $\langle$**U**,**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-monotically by means of a defeasible glue logic exploiting lexical and world knowledge.

4.2 Building a SDRS

Starting from a "message" encoded into a logical form, the document structuring module builds a

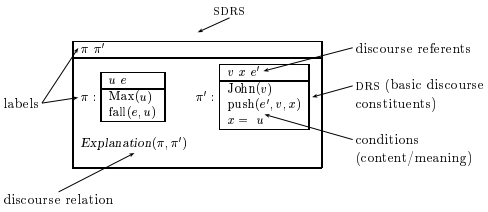
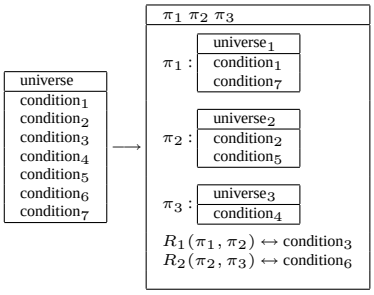


Figure 3: SDRS for *Max fell. John pushed him.*

SDRS. In a first step, the logical form is translated into a DRS. In case of a purely existential formula[1], this could just amount to putting all the variables into the universe of the DRS and split the formula into elementary conjoined conditions. However, there is an important difference between SDRSs and logical forms. SDRSs represent discourses and their variables are discourse referents. Logical forms represent meanings and their variables are pure logical variables. Therefore, it has to be decided which variables in the logical form become discourse referents (the linguistic consequences of this decision are explained in section 5). For that purpose, logically equivalent formulae are computed through two operations called *reification* and *dereification*. From a formula such as $\exists e_1, e_2$ *cause*(e_1, e_2), the causal relation can be *reified* to get $\exists f, e_1, e_2$ *cause*(f, e_1, e_2). Then *f* appears in the final SDRS as a discourse referent. Conversely, from $\exists f, e_1, e_2$ *cause*(f, e_1, e_2), the causal relation can be *dereified* if no other condition than *cause*(f, e_1, e_2) has *f* as an argument. It is not in the scope of this paper to explain when the *reification* and *dereification* operations should be applied.

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)DRSs. This process is schematized below.



Let us first examine the principles governing the splitting of the conditions. All the conditions in the

[1]More complex formulas are not considered here.

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 $R_1(\pi_1, \pi_2) \leftrightarrow \text{condition}_3$, 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 for *Narration* [2]. 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.

For text generation, this axiom is reversed in the rule below (Roussarie, 2000, p. 154).

- If k_1 and k_2 are DRS the main eventualities of which are not states,
- and if the *me* of k_1 occurs before the *me* 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"

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 brand new variable y is created in the universe of the current sub-DRS labelled π_y and the condition $y = x$ is added into 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.

Document structuring module à la SDRT base 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 to establish discours relations (obtained by reversing the rules in NLU) are generic. Below an example of SDRS in GePhoX.

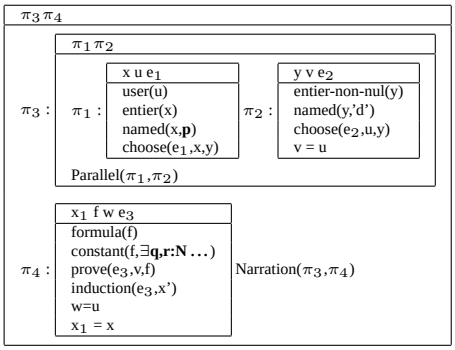


Table 6: SDRS for Euclidian division

5 Generating a text from a SDRS

A SDRS can be given as the input of existing tactical components. Here, we illustrate the process of generating a text from a SDRS using G-TAG (Danlos, 2000) whose architecture is represented at the bottom of Figure 1.2.

The microplanner is based on a lexicalized conceptual-semantic interface. This interface is made up of *concepts*; each concept is associated with a lexical data base. In our model, a concept is either a term in the T-Box or a discourse relation. A lexical data base 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 $\langle$**U**,**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,

[2] $\Box(\textit{Narration}(\pi_1, \pi_2) \rightarrow \textit{me}(\pi_1) < \textit{me}(\pi_2))$

and Cue is a cue phrase which is encoded in the lexical data base 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 counterpart of constraints in understanding)[3]:

  - a discourse referent is generated as an NP or a tensed verb.
  - conditions guide lexical choices. Conditions such as $x =$ John correspond to proper names. 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 is based on a TAG grammar which 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

We have shown in this paper how to integrate DL, SDRT, and a lexicalized grammar into an NLG system. Moreover, GePhoX illustrates the applicabilty of our system, which 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.

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 $d \neq 0$. We prove the following by induction on p: $\exists q,r{:}\mathbb{N}\ (r < d \wedge p = q.d + r)$. 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 $a' = a - d$. Using the induction hypothesis on a', we find two naturals q, r such that $r < d$ and $a' = q.d + r$. We take S q and r as quotient and remaining for the division of a. We must prove $a =$ S $q.d + r$ which is immediate. ∎

Table 7: A Text of proof for Euclidian division

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. 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.

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.

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 développement de générateurs de textes CLEF. In *Actes du 2è Colloque Francophone sur la Génération Automatique de Textes, GAT'99*, Grenoble.

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 Denis Diderot, Paris 7.

[3] With these constraints, an element which is *reified*, e.g. *cause*(f, e_1, e_2), gives rise to an NP or a verb (*the cause of, provoke*) and an element which is not *reified*, e.g. *cause*(e_1, e_2), gives rise to a modifier on e_1 or e_2 with e_1 and e_2 generated either as verbs or NPs.