MATHÉMATIQUES ET SCIENCES HUMAINES

GÉRARD MILHAUD
ÉLISABETH GODBERT

A logic-based environment for developing natural language processing applications

Mathématiques et sciences humaines, tome 143 (1998), p. 41-63
<http://www.numdam.org/item?id=MSH_1998__143__41_0>

© Centre d'analyse et de mathématiques sociales de l'EHESS, 1998, tous droits réservés.

L'accès aux archives de la revue « Mathématiques et sciences humaines » (http://msh.revues.org/) implique l'accord avec les conditions générales d'utilisation (http://www.numdam.org/conditions). Toute utilisation commerciale ou impression systématique est constitutive d'une infraction pénale. Toute copie ou impression de ce fichier doit contenir la présente mention de copyright.

NUMDAM
Article numérisé dans le cadre du programme
Numérisation de documents anciens mathématiques
http://www.numdam.org/

Math. Inf. Sci. hum., (36e année, n°143, *1998*, pp.41-63)

A LOGIC-BASED ENVIRONMENT FOR DEVELOPING NATURAL LANGUAGE PROCESSING APPLICATIONS

Gérard MILHAUD and Élisabeth GODBERT*

RÉSUMÉ — Un environnement basé sur la logique pour développer des applications du traitement automatique du langage naturel

Nous présentons un système formé d'un ensemble d'outils pour le développement d'applications du traitement automatique du langage naturel (interfaces en langage naturel, systèmes d'aide à la communication, etc.). Ce système est construit sur deux principes : modularité de la représentation des connaissances pour assurer la portabilité du système, et aide à la composition des phrases pour assurer la transparence, c'est-à-dire assurer que les phrases produites sont bien formées aux niveaux lexical, syntaxique, sémantique et conceptuel. Nous décrivons les formalismes que nous avons définis pour la représentation des connaissances, le processus d'aide à la composition de phrases, et quelques exemples d'applications développées à partir de ce système.

SUMMARY - We present a system providing a set of tools for developing natural language processing (NLP) applications such as natural language interfaces, communication aid systems, etc. This system is based on two principles: modularity of knowledge representation to ensure the portability of the system, and guided sentence composition to ensure transparency, i.e. to ensure that the produced sentences are well-formed at the lexical, syntactic, semantic and conceptual levels. We first describe the formalisms we have defined for knowledge representation. Then we explain how guided sentence composition is carried out. Finally, we give several examples of applications developed from this system.

1 Introduction

The current state of art in the area of computational linguistics clearly shows that the elaboration of a general natural language processing (NLP) system requires a large amount of work in modelling all the knowledge there is to be taken into account: knowledge about the language itself, its semantics, and the world of discourse. Besides, it is clear that all the current functional NLP systems only work on subsets of natural language, because all the knowledge to be modelled is so extensive and complex that defining a *total* model for it seems quite utopian.

To be realistic in view of this state of facts, the ILLICO system (see [Battani et al. 91], [Milhaud 94], [Pasero et al. 95]) has been designed from an original approach we think pertinent regarding the unavoidable limitations of any NLP system. ILLICO provides a set of tools making it possible to generate *various* NLP systems (NL interfaces, communication aid systems, computer assisted teaching or learning systems, etc.); it has been based on the following two principles :

- Modularity in knowledge representation, to ensure the portability of the system;

*Laboratoire d'Informatique de Marseille, {milhaud, godbert}@lim.univ-mrs.fr

- Guided sentence composition to ensure transparency (i.e. to ensure that the produced sentences are well-formed at each level: lexical, syntactic, conceptual and contextual).

Guided sentence composition requires a real-time processing of language; to realize it, we use a logical approach, where :

- The whole necessary knowledge is modelled in logic-like formalisms;
- All the constraints on the different levels of well-formedness are simultaneously used through parallel-like logical calculations.

The paper is organized as follows:
In section 2 we describe the main objectives of the approach: portability and transparency of the system. In section 3 we describe the different formalisms we have defined for knowledge representation, and, for each of them, we recall related works. In section 4 we explain how the constraints on the different levels of well-formedness are coroutined, to perform real-time analysis/synthesis of sentences. In section 5 we make explicit the limitations of the system. In section 6 we give several examples of applications developed from this generic system.

2 Objectives: portability and transparency

For the development of any general NLP system, one must admit that no present linguistic model provides an exhaustive grammar for natural language (or even for a significant part of it). Moreover, it is likely that this situation will not have a short or even middle term appreciable evolution. As for the semantic level in NLP, one is far from being able to represent the meaning of sentences, even if the study is restricted only to sentences composed from presently available grammars. In such circumstances, we can only choose between two main options:

- Continue trying to conceive systems with a syntactic coverage presumably equal to their users' syntactic knowledge;
- Admit our present inability to elaborate systems provided with all human linguistic knowledge, but take all possible steps to ensure that the unavoidable limitations are *i*) as tolerable (i.e. the least constraining) as possible and *ii*) explicit for the user.

We think that the elaboration of a system belonging to the first option is bound to fail: despite all the conceptor's efforts, the system will reject unavoidably and unexpectedly some sentences. Besides, a systematic production of a paraphrase is desirable so that the user can be sure that his sentence has been correctly understood, but this processing is costly and makes the dialogue much slower. Such a system is called *opaque*: the user has to investigate with some uncertainty to find a "convenient style" with respect to a grammar issued from the conceptor's own preferences. We think that such a system is not viable.

The generic ILLICO system stems from the second option. Our aim is to prove that it is now possible to elaborate dialogue systems that are actually usable. The two basic principles of ILLICO are *i*) portability by means of knowledge modularity and *ii*) transparency by means of guided composition of sentences.

2.1 Portability by means of knowledge modularity

We think that the requirement of *generality* must be replaced by a requirement of *portability*. Instead of aiming at a single system fitting all (or at least many) domains, — which implies that it contains considerable knowledge —, we think it is better to design a system easily portable from one domain to another.
Portability is related to knowledge *modularity*: ILLICO is meant to be an environment for developing dialogue systems for any applications, and the investment required for the transition from one application to another will be all the more reduced if the different types of knowledge (lexical, syntactic, etc.) are independently encoded in distinct modules.
Besides the fact that modularity is basic to achieve portability, it also makes it possible to illustrate our views on the processing of sentences with respect to different hierarchically sorted levels of *representation* and *well-formedness*.
Generally a sentence is considered completely well-formed if it respects the constraints defined at different levels: prosodic, phonetic, lexical, syntactic, semantic, conceptual, contextual, pragmatic. In ILLICO, we study the well-formedness of sentences at the lexical, syntactic, semantic, conceptual and contextual levels.
Any NLP system developed from ILLICO contains five modules, each associated to a different level of representation and well-formedness:

- A lexicon (lexical level) contains expected words and expressions.
- A grammar (syntactic level) specifies the expected sentence structures and the grammatical agreement.
- A set of semantic composition rules produce semantic representations from the syntactic rules of the grammar.
- A conceptual model (conceptual level) specifies the lexical presuppositions associated with the expected words and expressions.
- A contextual and discursive model (contextual level) specifies the state of the "world" described by the preceding sentences (i.e. existential presuppositions on individuals in the world of discourse).

A sentence is lexically well-formed if the lexicon contains all the words occurring in it; it is syntactically well-formed if the grammar contains all the structures occurring in it; it is conceptually well-formed if it describes a conceptually possible situation, i.e. in which all the mentioned individuals and relations are compatible; it is contextually well-formed if the situation it describes is compatible with the reference situation, i.e. with the discourse context.
The formalisms adopted for knowledge representation in the first four modules are described in section 3. The module related to the last level is presently being defined and implemented, and is described in [Mouret 98]

Separating the different types of knowledge provides other advantages. Incremental development of an NLP system generated from ILLICO is the most important one: the lexical and syntactical modules are first developed, the system is checked at these first two levels, and then the other levels are developed in following steps.

2.2 Transparency by means of guided composition

Our objective is to enable elaboration of NLP systems

- Provided by a limited but clearly defined competence, to ensure a total reliability within this competence;
- As little constraining as possible, so that the user can easily adjust.

Such systems are called *transparent* since the unavoidable restrictions are clearly exhibited to the user: they are permanently able to specify their exact competence, so that the dialogue is never blocked and the user never disappointed.
Transparency, when carried out correctly, provides many other advantages:

- The user rapidly perceives the limits of the system, which is used at its best.
- The conceptor does not have to provide a lexicon and a grammar with many synonymous variants: one of them is enough. This makes it possible to work rapidly, with reasonably limited linguistic components.
- The system is never blocked, for it shows the user the right possible expressions as soon as an error occurs.
- Finally, if the system is well implemented, no ambiguous sentences will ever be produced, and paraphrases are no longer needed.

In [Milhaud 94] we describe several more or less transparent approaches and show that ours is the only *totally* transparent one: all the advantages enumerated above are effective.

To achieve transparency, a "guided mode" has been chosen for the composition of sentences: at each step in the production of a sentence, the system proposes to the user the only words that can lead to a lexically, syntactically, conceptually and contextually well-formed sentence. In order to do this, the words are selected from linguistic, conceptual and contextual knowledge about the world of the considered application. Guided composition of sentences (GC) is done by partial synthesis of sentences, a principle introduced in [Colmerauer et al. 82] and discussed in [Sabatier 89]; the same system is used for analyzing and for providing partial synthesis of a sentence.
In the ILLICO system, the GC can be thrown out of gear, so that the user works in a "free mode": in this case, his sentence is classically analyzed by the system. This mode of sentence composition is the best one for people who are used to working with the GC: the user feels as free as in an opaque system. The GC mode occurs when the user directly asks the system for help, or as soon as the system has detected an unexpected expression. At any moment, the user can go back to the free mode.
Finally, the GC mode is the feature that makes the system a very general tool, whose applications can be, as mentioned above, natural language interfaces, communication aid systems, etc. Applications developed from ILLICO are described in section 6.

3 Knowledge representation and adopted formalisms

Each type of knowledge is represented by means of a distinct formalism, within a distinct module. The adopted formalisms are simple, high-level, and independent of any programming language (although our work has been driven by a logical-based and even Prolog-based approach, according to a long Marseilles tradition for NLP). In what follows, we briefly describe these formalisms and the underlying theories (for a complete description see [Milhaud 94]).

3.1 Modelling syntactic knowledge

We represent syntactic knowledge in a formalism stemming from *metamorphosis grammars* (MG) introduced in [Colmerauer 75].
Here is an example of a MG rule (the last expression between brackets expresses a condition):

```
noun_phrase(t_ype) =>
     preposition_or_void(t_ype)
     determiner(t_ype,x)
     common_noun(y)
     relative_clause(z)
     {gender_number_agreement(x,y,z)}
```

We have chosen MG for two reasons:
- We want to use a uniform approach based on logic and logic programming; since MG are issued from Colmerauer's research on NLP which have led to Prolog, MG rules can almost be considered as Prolog code.
- MG constitute a very simple formalism for writing grammars, and this is important because the grammars of the systems generated from ILLICO will not always be written by linguists.

3.2 Modelling semantic knowledge

Our objective is to associate a semantic representation s to any lexically and syntactically well-formed sentence of a fragment F of a natural language, in order to be able to calculate from s the well-formedness of the sentence at the conceptual and contextual levels.

The semantic representation is expressed in a logical language L, defined by the developer by means of one of the tools provided by ILLICO, the *semantic knowledge description formalism* (SKDF); in our system, such logical languages defined by the developer are called *meaning representation intermediate formalisms* (MRIF).
So we do not want to achieve a *direct interpretation* of conceptual and contextual well-formedness from the sentences themselves, but an *indirect interpretation*, from the translation of sentences into L. The *indirect interpretation* makes it possible to get rid of natural language ambiguities by using the rigourous syntax and semantics of a logical language. To perform a strict and systematic translation we adopt an approach similar to Montague's ([Montague 73]).

Our SKDF only imposes two constraints:

1. The MRIF must be logic-like (first-order);
2. The translation of a sentence into the MRIF must be compositional: the translation of a compound expression is a function of the translation of its components and of their mode of combination.

The second constraint is related to the "guided sentence composition mode": we must detect any error as soon as it occurs, even before the end of the composition of the sentence.

So each partially composed sentence must have a translation.
Since our system is modular, arguments of grammatical symbols must only convey syntactic knowledge and cannot be used to build the translations of the analyzed/synthesized sentences. So *rules of translation* are defined in a special heading of the SKDF: to each definition of a syntactic category C as a function of its components $c_1, \ldots, c_n$, we associate in the semantic module a rule giving the translation $[\![C]\!]$ of C as a function of the translations $[\![c_1]\!], \ldots, [\![c_n]\!]$ of $c_1, \ldots, c_n$.
Our formalism must therefore allow the developer to express the combination of the $[\![c_1]\!], \ldots, [\![c_n]\!]$. Our SKDF makes this combination possible via the *typed* $\lambda-$*calculus* formalism.

Then, the compiler mixes these rules with the syntactic and lexical ones into a single program: the program *sentence(D,R)* is able to generate *in parallel* all the pairs (D, R) where D is a derivation tree generated according to the grammar, and R its translation into the MRIF.
To perform a strict and systematic translation like Montague's, we process like him. To obtain the necessary homomorphism between the syntactic trees of the sentences of the fragment F and the syntactic trees of the translations of the sentences into the MRIF, we define a translation process whose compositionality ensures the homomorphism.
First, each elementary category of the MRIF (predicates, formulas,...) is labelled by a type which expresses how the categories can be combined to define higher-level categories (and associated types).
Then, a systematic correspondence is established between the syntactic categories of F (grammatical categories) and those of the MRIF (types) (heading *types* of the SKDF, described in part 3.2.1). If the C_f category in F corresponds to the C_l category in the MRIF, then each element of C_f must be translated into an element of C_l.
Then, the rules of translation of non-terminal categories are defined, expressing how the expressions of these categories of F have to be translated into the MRIF. These rules must respect the above mentioned correspondence, to ensure that the categories of the two languages are associated in a coherent way.
The last step is the definition, always respecting the above defined correspondence, of the rules translating the basic expressions of F, i.e. the expressions of preterminal categories. Each basic expression of F is translated into a single expression of the MRIF.

The typed $\lambda-$calculus is a very crucial point of our SKDF. We use the types in a usual way, shortly described below.
Creation. From an initial set of *basic types*, an infinite set of types can be generated: each basic type is a type and if a and b are types, $< a, b >$ is a type.
Use. Types are used to label sets of expressions, and to express the possible combinations of these expressions. This is done in accordance with the simplification rule: if A is an expression of type $< \alpha, \beta >$, then it can only be combined with an expression B of type α to give the resulting expression $A(B)$ (*application of A on B*) of type β.
For example, if t_1, t_2 and t_3 are basic or complex types, then $t_4 =< t_1, < t_2, t_1 >>$ and $t_5 =<< t_3, t_2 >, < t_3, t_2 >>$ are complex types. An expression of type t_4 can only be applied on an expression of type t_1 to obtain an expression of type $< t_2, t_1 >$; an expression of type t_5 can only be applied on $< t_3, t_2 >$ to obtain an expression of type $< t_3, t_2 >$.

3.2.1 Our semantic formalism: SKDF

The SKDF allows the developer to define his own MRIF L, and to explain the compositional translation of the fragment F (described by its grammar) into the associated MRIF. The SKDF is made up of several headings, related to the different actions mentioned above.

a) Definition of the syntactic categories of the MRIF: the "types" heading

In the ILLICO system, there are only two basic types: the formula type, noted o, and the individual variables and constants one, noted i.
In the "types" heading, a syntactic category of the MRIF is associated to each grammatical category. The corresponding rules have the form $X = t$, where X is a grammatical category and t a type. Here are some examples of such rules:

```
sentence = o;
common_noun_0 = <i,o>;
determiner = < <i,o>,< <i,o>,o> >;
```

Typing expressions conveys a special meaning: if o is the type of formulas and i the type of individuals, it seems natural to associate the type o to the category *sentence* since sentences are usually translated into formulas. More explanations about the choice of types are given in [Milhaud 94].

b) Definition of the quantifiers of the MRIF and their types: the "quantifiers" heading

The developer of a NLP system generated from ILLICO must define the quantifiers of the MRIF. These quantifiers will appear in λ−expressions so, for each of them, the developer must specify its syntactic category, i.e. its type, which defines the possible types of the expressions in the scope of the quantifier.
For us, the expression $Qx\ p$ — where Q is a quantifier, x the quantified variable and p an expression — is compositionally built by the application of Q on p (p is the argument of Q).
For example, we give the rules defining the three quantifiers *the*, *some*, *two_or_three* (in our formalism, and in all the following parts, the application $A(B)$ of the expression A on the expression B is noted $[A, B]$; *inf* means ∞):

```
the / < <i,o>,o> / [the,p] = the(<x,<1,1> >,[p,<x,<1,1> >]);
some / < <i,o>,o> / [some,p] = some(<x,<2,+inf> >,[p,<x,<2,+inf> >]);
two_or_three / < <i,o>,o> /
   [two_or_three,p] = two_or_three(<x,<2,3> >,[p,<x,<2,3> >]);
```

The quantified variables (i.e. the variables of the MRIF) are of type i, for they model individuals. The above rules show that, in the definition of a quantifier, the quantified variable is coupled with a pair of constants $c =< a, b >$ which does not interfere with types and λ−calculus operations. This pair $< a, b >$ encodes the lower (a) and upper (b) bounds of the cardinality of *the set of individuals* represented by the quantified variable: we impose actually that the variables of all our MRIF represent *sets of* individuals so

that we can process conceptual constraints on cardinality to prevent conceptually incorrect expressions such as **The noses of Peter*. These constraints will be described in section 3.4.

c) Definition of the connectors of the MRIF and their types: the "connectors" heading

The definition of the connectors is done in the same way as the definition of quantifiers. Here are examples of rules defining connectors:

```
not / <o,o> / [not,f1] = not(f1);
and / <o,<o,o> > / [[and,f1],f2] = and(f1,f2);
or / <o,<o,o> > / [[or,f1],f2] = or(f1,f2);
```

d) Translation of the expressions of the non-terminal grammatical categories: the "rules" heading

We give below an example of two simple translation rules, labelled T_i, $i = 1,2$. We first mention the entire context, i.e. the associated grammar rules (labelled G_i) and the necessary definitions from the "types" heading. The abstraction $\lambda x\ N$ is noted $< x, N >$.

```
G1 sentence =>
     noun_phrase
     verb_phrase;

G2 verb_phrase =>
     verb1
     noun_phrase;

types;
sentence = o;
noun_phrase = < <i,o>,o>;
verb_phrase = <i,o>;
verb1 = <i,<i,o> >;

rules;
T1 sentence = [noun_phrase,verb_phrase];
T2 verb_phrase = <x,[noun_phrase,[verb1,x]]>;
```

One can verify, as the compiler does it, that the types of the right and left expressions of T_i rules are identical. How does the compiler process the free $\lambda-$variables (like x in T_2)? It associates the type which makes correct the first occurrence encountered (here the $< x, \ldots >$; so the type i is associated to x) and then it checks whether this type is also convenient for every other occurrence it will encounter (here $[verb1, x]$; it is correct): if not, it shows an error.

e) **Translation of the expressions of the preterminal grammatical categories: the "preterminal" heading**

Finally, to complete the definition of his MRIF, the developer must define, for each basic expression of the fragment, i.e. each expression E_{p_c} of the preterminal categories, its translation $[\![E_{p_c}]\!]$. Here is an example, for indefinite article:

```
indef_article = < p , < q , exists(i , [p , i]) , [q , i]) > >;
```

3.3 Modelling lexical knowledge

The lexical module constitutes the terminal part of the syntactic and semantic ones. It is a set of lexical items. Each of them is a 4−uple $< e, f, g, l >$ where e is a lexical entry (word or expression), f a set of phonetic information, g a set of grammatical information and l the logical terminal (for the only preterminal categories which require it, see 3.2.1, e)). Information is expressed by means of features (couples of the form *<attribute, value>*). Morphological rules may calculate derived lexical items (e.g. different forms of verbs) from basic ones. For example, the word "*tree*" would be encoded as follows (only lexical entry, preterminal category and logical terminal are required by the ILLICO system).

```
lexical entry            : tree
initial                  : consonnant
initial of following word : any
preterminal category     : common_noun
gender                   : masculine
number                   : singular
logical terminal         : be_a_tree
```

3.4 Modelling conceptual knowledge

In this section, our objective is the definition of a logical model for conceptual knowledge associated to the application, i.e. to the world of discourse; it expresses common sense inferences and we elaborate it to prevent the production of conceptual incoherences during the composition of sentences.
We focus on two types of conceptual constraints (introduced in [Godbert 94]): domain constraints prevent incoherences such as *the mother of the mountain, the mountain eats an idea,* which come from incompatibility between words, and connectivity constraints prevent incoherences such as *the mothers of Peter, the average of my mark, the bird is numerous*, which come from the misuse of a singular or plural form.
From the semantic representation of sentences and from the conceptual constraints we define in this section, logical calculations enable us to determine whether a sentence is conceptually coherent. These calculations are described in section 4.2.

3.4.1 Related works on conceptual incoherence

As it is usually done, we consider that a sentence is incoherent if it is in opposition to common sense inferences i.e. if it expresses information that cannot be true in any normal situation. Besides, an incoherent expression "contaminates" any sentence in which it occurs; for example, if a proposition is incoherent, its negation is also incoherent ([Cruse 86]).

Common sense inferences have been studied in different areas and modelled in different ways: in lexical semantics (lexical relations, semantic features, selectional restrictions, etc., e.g. in [Cruse 86]); in knowledge representation (ISA link, role restrictions, role cardinality, e.g. in [Brachman 85]); in relational data bases (cardinality of attributes, functional or multivalued dependencies, e.g. in [Aho et al. 79]). To model this common sense knowledge in our system, we define in the following sections a logical formalism for conceptual constraints, which takes into account these various related works.

3.4.2 Modelling relations between sets

We consider the world of discourse as a set $\mathcal{E}$ of individuals inter-connected by relations expressed through natural language sentences. In plural noun phrases, these individuals occur inside sets, so we study relations connecting sets.
For the formal point of view, the logical language L used for the representation of sentences (i.e. the MRIF) will be interpreted in $\mathcal{P}(\mathcal{E})$, the set of all the subsets of $\mathcal{E}$.
We associate a set x, element of $\mathcal{P}(\mathcal{E})$ to each noun phrase occurring in a sentence, and we note $\mid x \mid$ its cardinality (in the present state of our work, we only study countable sets). According to the type of the noun phrase denoting the set x, $\mid x \mid$ is more or less well known (*a rabbit, John and Paul, three cats, books,* etc.).

Let $\mathcal{I}$ be the set of all intervals of the type $[a, b]$, where a is a positive integer and b is an integer greater or equal to a, or b is $+\infty$.
From a natural interpretation of determiners, we associate an element of $\mathcal{I}$ to each noun phrase n: if x is the set representing n, there exists a minimal element of $\mathcal{I}$ to which $\mid x \mid$ belongs. For example, if n is a singular noun phrase, $\mid x \mid \in [1, 1]$; for an indefinite plural, we only know that $\mid x \mid$ is greater than 1, so $\mid x \mid \in [2, +\infty]$.
We then consider a set R of relations on $\mathcal{P}(\mathcal{E})$: each verb, noun or adjective defines a verb phrase, and we associate to it a relation on $\mathcal{P}(\mathcal{E})$; each n-ary relation r in R is an application from $(\mathcal{P}(\mathcal{E}))^n$ into $\{0, 1\}$; its graph is a subset of $(\mathcal{P}(\mathcal{E}))^n$.
Examples: *talk(x), send(x,y,z), be-human(x), be-John(x), be-father-of(x,y), be-blue(x),* etc.; the arguments $x, y, \ldots$ represent the elements of $\mathcal{P}(\mathcal{E})$ associated by the relations (in a sentence represented by the formula $r(x_1, x_2, \ldots, x_n)$, they correspond to the subject and various complements of the verb phrase modelled by r).
These relational symbols (*talk(x), send(x,y,z), etc.*) of the logical language are associated with verbs, nouns and adjectives *via* the logical terminals defined in the lexical module (see 3.3).

The conceptual constraints we have to model to prevent incoherences in sentences will be defined on elements of R: the conceptual constraints attached to a n-ary relation r correspond to common sense inferences associated to the verb phrase represented by r. More precisely, they define necessary conditions an element $(x_1, x_2, \ldots, x_n)$ of $(\mathcal{P}(\mathcal{E}))^n$ must fulfill to be in the graph of r (i.e. conditions on r and $(x_1, x_2, \ldots, x_n)$ in order to $r(x_1, x_2, \ldots, x_n)$ to be true).

3.4.3 Domain constraints

Definition

Domain constraints correspond to the selectional restrictions defined in lexical semantics, and to the role restrictions defined in knowledge representation. A domain constraint on a relation r imposes that the argument $(x_1, x_2, \ldots, x_n)$ of r is included in a special subset of $\mathcal{E}^n$. To model this type of constraint, it is necessary to distinguish a number of subsets of $\mathcal{E}$ and to specify their possible inclusion or disjunction inter-relations.

We first define a set D of *domains* of $\mathcal{E}$ to which the individuals belong: a domain d is a subset of $\mathcal{E}$, defined by one property, and often related to a natural species (*human, animal, fruit tree,* etc.). This notion of domain is close to the classical notion of concept defined for knowledge representation (see [Brachman 85]). $\mathcal{E}$ is the greatest domain. The set D of all the domains is partially ordered by the inclusion relation defined in $\mathcal{P}(\mathcal{E})$.
We then define a set $\mathcal{D}ec$ of *decompositions* of domains.

Definition
A *decomposition* of a domain d is a set $\{d_1, d_2, \ldots, d_p\}$ of disjoint domains, each strictly included in d; the decomposition is noted: $dec(d, d_1.d_2. \ \ldots \ d_p)$.
Our system allows the definition of several decompositions of the same domain, and this enables the developer to build a "multiple" taxonomy, much less restricted than the usual taxonomies of classes defined for knowledge representation.

Examples
dec(Animate, Human.Animal); *dec*(Human, Man.Woman.Child);
dec(Animal, Dog.Horse.Rabbit.Cow); *dec*(Human,Teacher.Doctor.Farmer.Trader).

Definition
We define an application *dom* from R into $\bigcup_{n>1} D^n$ that, to each n-ary relation r, associates an element of D^n.
$dom(r) = (d_1, d_2, \ldots, d_n)$ is the domain constraint of r.
This definition means: if the n-uple $(x_1, x_2, \ldots, x_n)$ is in the graph of r, x_i is included in d_i for each i.

Examples
For the relations $eat(x)$, $speak(x)$ and $operate\text{-}on(x, y)$, we define the domain constraints: *dom*(eat) = (Animate); *dom*(speak) = (Human); *dom*(operate-on) = (Doctor,Animate).

3.4.4 Connectivity constraints

Connectivity constraints are defined to prevent incoherences due to a misuse of singular or plural. A connectivity constraint on a relation r specifies dependencies between the cardinalities of the sets x_i occurring in a formula $r(x_1, x_2, \ldots, x_n)$. The notion of connectivity introduced here for natural language processing is an adaptation of the notions of cardinality and multi-valued dependency.

Examples
- In the formula $be\text{-}mother\text{-}of(x, y)$, $|x|$ must be smaller or equal to $|y|$.
- In the formula $practise(x, p, d)$, (*the person x practises the profession p at the date d*), if we assume that a person can practise only one profession at a time, each pair (x, d) can be connected to only one p. So, if $|x|$=k and $|d|$=h, $|p|$ must be smaller or equal to $(k \times h)$.

To model connectivity constraints, we need the notion of c-triplet.

Definitions
For $n \geq 1$, we call *c-triplet of type n* any triplet (s, k, j) such that:
s is a strict subset of $\{1, \ldots, n\}$, $k \in \{1, \ldots, n\}$, $k \notin s$, $j \in \mathcal{I}$.

We note $T^{(n)}$ the set of the c-triplets of type n, and $T = \bigcup_{n>1} T^{(n)}$

We define an application *conn* from R into $\mathcal{P}(T)$: for any n-ary symbol r, $conn(r)$ is a finite set of c-triplets of type n.
$conn(r) = \{t_1, \ldots, t_q\}$ is the connectivity constraint of r.
The relation $(s, k, j) \in conn(r)$ has the following meaning: if the n-uple $(x_1, x_2, \ldots, x_n)$ is in the graph of r, if each argument x_i, with i in the set s, is a set with cardinality 1, then the cardinality of the k-th argument is in j.
Examples
For the relations *be-mother-of*(x, y) and *practise*(x, p, d) we define the connectivity constraints:
conn(be-mother-of) $= \{(\{2\}, 1, [1, 1])\}$, *conn(practise)* $= \{(\{1, 3\}, 2, [1, 1])\}$.
We generalize the conditions defined above for these two examples by the definition of the formula $\varphi(t, x_1, x_2, \ldots, x_n)$, where t is a c-triplet:

If $t = (s, k, j), s = \{i_1, \ldots, i_h\}$ and $j = [m, p]$,
$\varphi(t, x_1, x_2, \ldots, x_n)$ is: $m \leq |x_k| \leq p \times |x_{i_1}| \times \ldots \times |x_{i_h}|$

If $t = (s, k, j), s = \{\}$ and $j = [m, p]$, $\varphi(t, x_1, x_2, \ldots, x_n)$ is: $m \leq |x_k| \leq p$.

Then, the formula $t \in conn(r)$ means that: if the n-uple $(x_1, x_2, \ldots, x_n)$ is in the graph of r, $\varphi(t, x_1, x_2, \ldots, x_n)$ is true.

4 A real-time processing of sentences

4.1 Coroutining constraints for a real-time processing of sentences

To compose sentences in a guided mode by partial synthesis, all the different levels of well-formedness must be simultaneously taken into account when the system analyzes a given sentence or synthesizes a partial one. Our exigence of modularity makes this necessary parallelism much more complex.
The solution we have adopted is to simulate it by *coroutining* constraints associated to the different levels of well-formedness. The core algorithm (in Prolog) is the following:

```
well_formed(Sentence):-
  conceptually_well_formed(SemanticRepresentation),
  lexically_and_syntactically_well_formed(Sentence, DerivationTree),
  sentence(DerivationTree,SemanticRepresentation).
```

We coroutine constraints in a classic way. The program *sentence(D,R)*, generated from the lexical, syntactic and semantic rules by the ILLICO compilator (see 3.2), is a *blind* building process controlled by two coroutined constraints. As mentioned above, it generates all the couples (D, R) where D is a derivation tree of the grammar and R its translation into the MRIF. Derivation trees and their translations are generated according to basic Prolog stategy (top-down, depth-first, left-to-right and non-deterministic).

According to this method, a sentence S is lexically and syntactically well-formed if S is the ordered (from left to right) list of leaves of a derivation tree D generated by *sentence*(D, R).
The *lexically_and_syntactically_well_formed* constraint is coroutined on D in such a way that a mismatch between D and S immediatly induces backtracking of the building process *sentence*(D, R). The algorithm of this constraint is based on the predicate *list_of_leaves* (see [Giannesini et al. 85]), coroutined by the built-in Prolog II predicate *freeze* and adapted for partial synthesis context.

The constraint *conceptually_well_formed* is a call to the conceptual constraints expressed in the conceptual model. It is coroutined on S in such a way that a conceptual inconsistency immediatly induces backtracking of the process *sentence*(D, R). The conceptual control must operate on non-reduced λ−expressions, as the semantic representation is progressively built. Why? Given the fact that the reduction of a λ−expression can be done only if all its elements are known, if the conceptual control is executed *after* the reduction, it may fall behind the building process; in this case lexically and syntactically correct but conceptually incorrect sentences would be analyzed or synthesized (this is related to the problem of belated dead ends, discussed in section 5). The part 4.2 describes the *conceptually_well_formed* constraint.

The program *well_formed* runs both in analysis mode (S is bound) and in synthesis mode (S is free). We give in the appendix an example of the complete process involved in the analysis or the production of a sentence.
We describe below the processing which, from the semantic representation of a sentence, calculates whether the sentence is conceptually coherent.

4.2 Computing conceptual well-formedness

We give here some explanations about the algorithm of the *conceptually_well_formed* constraint.
We have seen in 4.1 that we must consider the non-reduced forms of the (partial) translations of the sentences. So we must take into account the abstractions and applications of the λ−calculus to analyze translations.

The first aim of the algorithm is the detection of connectivity incoherences. It is processed by the procedure

$$connectivity\_verif(r(< x_1, c_1 >, \ldots, < x_n, c_n >), l)$$

where l is a list implementing the set $conn(r)$ of c-triplets associated to the relational symbol r. The *connectivity_verif* procedure is based on calculations defined from the formula $\varphi(t, x_1, x_2, \ldots, x_n)$ introduced in part 3.4.4, and fails if it detects an incoherence between the cardinalities c_i of the sets x_i.

The second aim of the algorithm is the detection of domain incoherences. It may be seen as a type checking (see [Milhaud 94]). After the compilation, all the formulas obtained from translation rules are correctly typed according to type-hierarchy based on simple types i and o. But, when the system is running, the conceptual constraints are

involved and operate a *sub-categorization* of the type i, for the tree of domains constitutes a finer hierarchy of the individuals.

Let us consider that a variable of type i which represents individuals from a domain d is now of the type T_d associated to d. In this context, checking that a sentence s is conceptually coherent consists in verifying that the abstractions and applications of $\lambda-$calculus involved in the translation of s are possible regarding the new sub-types of i. This verification is similar to the one carried out by the compiler for i/o type-hierarchy. The substitution of two variables of type T_{d_1} et T_{d_2} must be possible only if d_1 and d_2 are compatible domains, i.e. not disjoint. In Prolog, logical substitution is simulated by unification. To process the verification, we have implemented an internal representation of domains by means of lists ended by a free variable so that the representations T_{d_1} and T_{d_2} cannot be unified if d_1 and d_2 are disjoint.
With this method, as soon as a variable is constrained through $\lambda-$applications to represent a set of individuals from incompatible domains (i.e. as soon as a conceptual incoherence occurs), a unification fails and forces the backtracking of the building process *sentence(D,R)*.
More technically, we do not unify directly the variables with the types. When a free individual variable $< x, c >$ has to be typed by a call to conceptual constraints, we unify it with $<< v, T_{d_1} >, c >$ where v is a new free variable and T_{d_1} the type. So individual variables remain free and can be used for the next steps in the processing, such as the contextual treatment.
If $x =< v, T_{d_2} >$ before the call, then the unification is possible only if d_1 and d_2 are not disjoint. What about T_{d_1} and T_{d_2} after unification if they are not disjoint and not equal? The internal representation of the domains is such that T_{d_1} and T_{d_2} are unified into T_{d_3} where d_3 is the finest domain out of d_1 and d_2 (i.e. the farthest from $\mathcal{E}$).
Some more details on the algorithm of the *conceptually_well_formed* constraint and the algorithm itself are given in the appendix.

5 Limitations of our NLP system

Obviously, each application developed from the ILLICO system has its own linguistic coverage, depending on the contents of the lexical, syntactic and conceptual modules: so each application is limited by the number of words encoded in the lexicon, the number of syntactic rules encoded in the grammar, and the set of lexical presuppositions associated to the words of the lexicon.
But the main limitation of our system is related to the problem of *dead ends*, which is a crucial point in guided sentence composition: the system must never allow the user to compose by partial synthesis the beginning of a sentence such that it will not be possible to complete it.
We differentiate two kinds of dead ends: those due to the belated controls (*belated dead ends*) and those due to a lack of data (*data dead ends*). In both cases, the system cannot propose any word to continue the sentence. But the algorithm of the *lexically_and_syntactically_well_formed* constraint never backtracks on the already accepted words (the beginning of the sentence): backtracking only concerns the current word to be synthesized. So if there is no convenient word in the lexicon for the current level, the system is jammed.

The two kinds of dead ends differ on an important point. When a belated one occurs, last word(s) of the beginning of the sentence are always conceptually incorrect (but lexically and syntactically correct). On the other hand, no incorrect words are proposed before a data dead end arises.
Let us examine the two classes of dead ends.

• A belated dead end occurs when the *conceptually_well_formed* constraint *falls behind on* the building process *sentence(D,R)*. During a certain length of time, this constraint "is asleep" and the proposed words are not checked. When the constraint "wakes up", if the user has chosen conceptually incorrect words during the sleeping time, the incoherence is suddenly detected, but too late, and the system is jammed.
This phenomenon is related to the translation rules. The waiting for the result of λ−reductions is the cause of these dead ends. So, we have implemented an algorithm which operates the conceptual control on non-reduced λ−expressions. We have thus nearly completely eliminated belated dead ends. The remaining cases are due to a particular typing and combination of λ−expressions. We have isolated these situations and shown how to avoid them in [Milhaud 94].

• The data dead ends are independent of the processing. They are only related to the insufficiency of lexical or syntactic data. If the user composes a beginning of a sentence which is correct but imposes following words the lexicon does not contain, the system is also jammed: it is a *lexical* data dead end.
We consider the developer to be responsible for his lexicon: he must be sure that the preterminal categories always have a possible derivation into the lexicon. Moreover, this derivation must be possible for all the conceptual types which can be needed: for example, if there is in the lexicon a verb V, the condition to satisfy is not that the lexicon contains at least a common noun, but that the lexicon contains at least a common noun *of a compatible conceptual type* with the one required for the agent of V.

6 Applications

Each application developed by means of ILLICO environment has a modular architecture, and contains:
- The ILLICO kernel, composed with the linguistic engineering units (compiler, parsing/synthesis, guided composition, etc.).
- The specific linguistic data, consisting of the lexicon, the grammar, the semantic module, the conceptual model and the contextual model defined for the considered application.
- A graphic interface.
- The supervisor which manages the different tasks and operations.

We describe below several applications which have been developed from ILLICO.

• Natural language interfaces to databases

Several ILLICO natural language interfaces have been developed by our research team, for example:
- A prototype interface to a database on scientific library (see [Milhaud 94]), shown in Figure 1.

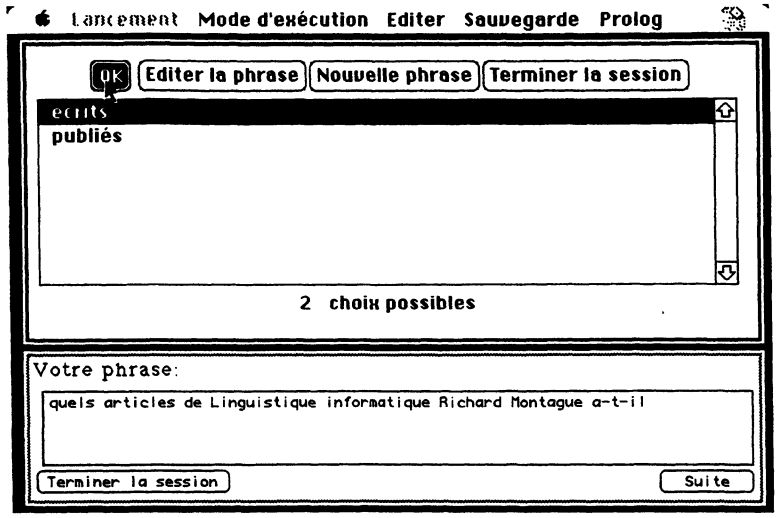

Figure 1: Example of a natural language interface

In this example, the user has composed the beginning of sentence *Quels articles de Linguistique Informatique Richard Montague a-t-il...* (*Which papers of Computational Linguistics Richard Montague has...*). Then ILLICO displays the two possible words to go on *écrits, publiés* (*written, published*) and the user chooses the word *écrits* (*written*) to go on.

- An Arabic interface to a database on planets of the solar system, which is an interesting application showing the portability of ILLICO from one language to another ([Boïde 94]).

- **The KOMBE system**

An important application of ILLICO to date is the KOMBE system. It is a communication aid system, realised in a French and a German prototype, devoted to non-speakers with motor difficulties like Amyothrophic Lateral Sclerosis (ALS) patients (see [Pasero et al. 94]).
For this kind of disabled person, the computer assisted aids developed so far offer the user relatively slow communication, composing sentences letter by letter or selecting a pre-formulated phrase or sentence. Some systems include a dictionary allowing the user to select whole words. Guided composition of sentences is a very convivial way for them to communicate. Step by step, they can select on the screen words and expressions dynamically synthesized by the system and which always lead to the construction of a well-formed sentence.
Figure 2 shows an example of a partial composition of a sentence by an ALS patient communicating with a doctor. Here the patient has composed the beginning of sentence *J'ai beaucoup de mal à avaler ce...* (*It is very difficult for me to swallow down this...*) and the system proposes him the possible following words *cachet, médicament* (*tablet, medicine*).

- **An iconic interface**

Between communication aid systems and the computer assisted language learning systems, a system has been developed for persons who are unable to communicate by means of natural language, but only by means of icons. This system makes it possible both to learn and to communicate by means of a pictographic code. From icons as lexical items,

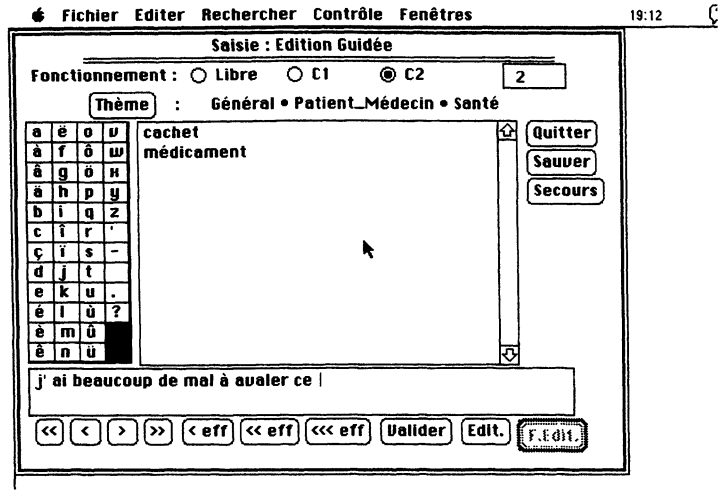

Figure 2: Example of a KOMBE session

grammar of the subset of natural language associated to the code and a classical conceptual model, ILLICO provides a system of pictographic guided composition.

- **The EREL system**

The EREL system is another application devoted to the disabled, which has been developed these last two years (see [Godbert 98]). EREL is a sofware devoted to the education and rehabilitation of children with language and possibly cognitive disorders, and especially to "autistic-like" children, who often suffer from language and communication disabilities. In addition to their difficulties to express themselves verbally, many of autistic children suffer also from cognitive and/or motor disorders which make graphic or written expression difficult.
The EREL system provides a set of user-friendly educational play activities, designed from communication and language training and learning exercises, and designed to stimulate, encourage, and help users to employ common language to build up an everyday language dialogue in interaction with the system, within a specific context illustrated through multimedia technology.
Generally speaking, users are able to express themselves on the subject proposed by the activity, with the assistance of the system (guided composition of sentences) or freely (free composition). The software has been designed as a multi-level and multi-user system: a system flexible enough to be adapted and to respond to specific needs according to the user's skills, which depend on his level of language and cognitive development, and his degree of autonomy.

The two characteristics of ILLICO described in the first part of this paper (modularity in the representation of knowledge and guided composition of sentence) are big assets for the development of EREL: first, the guided mode allows the user to compose rapidly, with minimal cognitive load, sentences which are always correct at every level; and modularity makes it easy to define various exercises about language with different levels of difficulty: by using linguistic modules (lexicon and grammar) with broad or restricted coverage, by allowing or not the guided mode, we obtain a lot of different exercises among which one can choose the one which is suitable for each user's capacities.

In EREL, the activities to be chosen by the users are dialogues on scenarios or on logic games.
In the first case, a scenario is illustrated by a picture or a photograph; the users comment on with sentences or questions about what they see on the screen.
In the second case, the users play by means of sentences: they may compose orders to achieve a goal (for example to move a piece of a puzzle), or they may comment on the progress of the game. One of the aims of this activity is the verbalization of actions.
Figure 3 shows one of the exercises proposed by EREL, in which the user has to manage screen objects, that he can put on a checker board, permute, move or stow away. The user can compose imperative, interrogative or affirmative sentences. If the sentence is well-formed, the system fully processes it and:

- An imperative sentence gives an order to the system, which immediately executes the corresponding action on the board.
- If the user has composed an affirmative sentence, the system checks the truth of its content.
- If the user has composed an interrogative sentence, the system answers to the question.

If the sentence has not been composed in the guided mode, it may be ill-formed, and in this case the system refuses it.

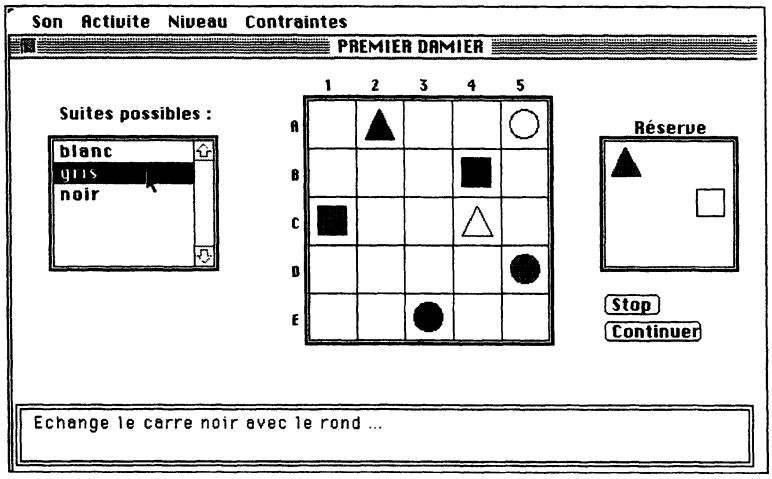

Figure 3: Example of a logic game in the EREL system

In this example, in the French version, the user has begun a sentence *Echange le carre noir avec le rond... (Permute the black square with the circle...)* and the system, according to the contextual situation, proposes the possible words to be selected: *blanc, gris, noir (white, grey, black).*

Here are some examples of sentences corresponding to different levels of linguistic and cognitive difficulty:

- *Permute the grey circle with the black triangle.*
- *Where is the black circle ?*
- *The white square is in the stock.*
- *Put the white circle in the square A 4.*
- *Add a white triangle under the pawn which is in the square D 5.*
- *Put the grey triangle at the left of the pawn which is under the pawn which is ...*

This type of exercise is an interesting application of the contextual processing of sentences, which has been recently developed and integrated into the ILLICO system.

7 Conclusion

The ILLICO system is an environment for developing NLP systems; it is based on strong modularity and "guided composition mode", that we think reasonable and pertinent options with regard to the complexity of elaborating general NLP systems. A real-time processing of language is necessary to carry out the guided composition mode and this is performed through the simultaneous control of the different levels of well-formedness of sentences. Different kinds of natural language interfaces and communication aid systems have already been developed from ILLICO.
We have described how the well-formedness of sentences in an ILLICO application is ensured at the lexical, syntactic and conceptual levels, independently of any context in which the sentence may be produced.
As we mentioned it in section 2.1 and above in the description of EREL, the work related to contextual coherence, to achieve text processing, has recently been developed. For this contextual processing of sentences, it is necessary to define (see [Mouret 98]):

- Logic-like contextual and discursive models, in which are described pre-existing individuals introduced in the preceding sentences.
- Logical calculations solving referential expressions and ensuring discourse consistency, to be uniformly coroutined with the processings defined for the first levels of well-formedness.

The next step of text processing is the semantic evaluation of sentences and the study of discourse consistency. Conceptual and contextual knowledge are used to perform inferences about properties of objects in the world of discourse: for example, from *Duke is barking*, we infer that *Duke is a dog*, and therefore *an animal, a living being, not a cat,* etc.
The present work on ILLICO aims at the processing of the whole discourse: the system will enable to associate to each sentence a semantic representation where all the references are solved, and the ordered set of these representations will reflect the semantic contents of the whole discourse.

Appendix

We give below the algorithm of the *conceptually_well_formed(R)* coroutined constraint (noted *CWF(R)*). This algorithm fails and induces a backtracking of the building process as soon as a conceptual incoherence (connectivity or domain one) occurs. We have to specify two points about this algorithm: first, it runs both for R reduced or non-reduced; secondly, *CWF(R)* is constituted by a single call to $CWF1'(R,t)$ which initializes the verification of types. t is the type of R, and is a free variable. The aim of the algorithm is to "go down" into R to unify the free variables representing the types according to $\lambda-$operations (if there are some). Then when a relational symbol is encountered, the associated constraint is called and the conceptual types of the arguments are distributed into R through unifications. If the unifications required by the next encountered relational

symbol are not possible in R after it has been so typed, the algorithm fails; else it goes on.

Algorithm of the *conceptually_well_formed(R)* constraint

procedure CWF(R)
begin
  CWF1'(R,t)
end

procedure CWF1'(R,t)
begin
  coroutine(R,CWF1(R,t))
end

procedure CWF1(R,t)
begin

If $R = [e_1, e_2]$ **then**
  unify(t, t_3)
  unify$(t_{e_1}, < t_1, t_3 >)$
  unify(t_{e_2}, t_1)
  CWF1'(e_1, t_{e_1})
  CWF1'(e_2, t_{e_2})

If $R =< x, e >$ **then**
  unify$(t, < t_1, t_2 >)$
  unify(t_e, t_2)
  CWF1'(x, t_1)
  CWF1'(e, t_e)

If $R = c(f_1, \ldots, f_n)$ **then** /
  /* c is a connector of the MRIF */
  CWF1'(f_1, t_1)
  $\vdots$
  CWF1'(f_n, t_n)

If $R = Q(< x, c >, f_1, \ldots, f_n)$ **then**
  /* Q is a quantifier of the MRIF, $< x, c >$ the quantified variable */
  CWF1'$(< x, c >, t_0)$
  CWF1'(f_1, t_1)
  $\vdots$
  CWF1'(f_n, t_n)

If $R = r(< x_1, c_1 >, \ldots, < x_n, c_n >)$ **then**
  /* r is a n-ary relational symbol of the MRIF */
  call the connectivity constraint $r(l)$
  connectivity_verif$(r(< x_1, c_1 >, \ldots, < x_n, c_n >), l)$
  call the domain constraint $r(T_{d_1}, \ldots, T_{d_n})$
  /*T_{d_i} is the internal representation of the domain d_i of the i^{th} argument of r.*/
  unify$(< x_1, c_1 >, << v_1, T_{d_1} >, c_1 >)$
  $\vdots$
  unify$(< x_n, c_n >, << v_n, T_{d_n} >, c_n >)$

If $R =< x, c >$ **then**
  /* x is a variable of the MRIF */
  unify$(< x, c >, << v, t >, c >)$

end

The part of *CWF1* which processes the connectors and quantifiers occurrences is different for each new ILLICO application. It is generated by the compiler from the names and the functional writings of connectors and quantifiers declared in the semantic module.

Example

We give here a complete example of detection of domain incoherence which shows how the above algorithm works. The user composes the sentence *A surgeon sings.* and we show the evolution of the translation of the sentence into the MRIF after each word.
First, we have to introduce all the data involved by the system to process the sentence (and for the reader to follow the example).

Notations: we note $[\![\mathrm{A}]\!]$ the translation of A into the MRIF; we will underline each new free variable introduced by the algorithm; we note $[A, B]$ the $\lambda-$application of A on B; $a \equiv_u b$ means the unification of a and b;
S, NP, VP, IV, CN, Det mean respectively Sentence, Noun_Phrase, Verb_Phrase, Intransitive_Verb, Common_Noun, Determiner.
In this example, we are not absolutely rigourous; some details, not necessary for comprehension, are omitted to lighten the formulas.

Grammar rule	**Translation rule**	**Types**
S $\rightarrow$ NP VP	$[\![\mathrm{S}]\!] = [[\![\mathrm{NP}]\!], [\![\mathrm{VP}]\!]]$	$[\![\mathrm{S}]\!] : o$; $[\![\mathrm{NP}]\!] :<< i, o >, o >$; $[\![\mathrm{VP}]\!] :< i, o >$
NP $\rightarrow$ Det CN	$[\![\mathrm{NP}]\!] = [[\![\mathrm{Det}]\!], [\![\mathrm{CN}]\!]]$	$[\![\mathrm{Det}]\!] :<< i, o >, << i, o >, o >>$; $[\![\mathrm{CN}]\!] :< i, o >$
VP $\rightarrow$ TV	$[\![\mathrm{VP}]\!] = [\![\mathrm{IV}]\!]$	$[\![\mathrm{IV}]\!] :< i, o >$

We simplify the link with the lexicon and give here the translations of the determiner "a", the common noun "surgeon", and the intransitive verbs "sings", "barks".
$[\![\mathrm{a}]\!] = \lambda P \lambda Q \; \exists x \; [P, x] \wedge [Q, x]$
$[\![\mathrm{surgeon}]\!] = \lambda v \; \mathrm{be\_a\_surgeon}(v)$
$[\![\mathrm{dog}]\!] = \lambda w \; \mathrm{be\_a\_dog}(w)$
$[\![\mathrm{sings}]\!] = \lambda y \; \mathrm{sing}(y)$
$[\![\mathrm{barks}]\!] = \lambda z \; \mathrm{bark}(z)$
(To be rigourous here, we would have to consider the application of predicative constants *surgeon, sing, bark* of type $< i, o >$ on $\lambda-$variables v, w, y ,z of type i. We shorten the process by considering directly the functionnal notations be_a_surgeon(v), be_a_dog(w), sing(y) and bark(z)) of type o.)

Conceptual data:
Decompositions: *dec*($\mathcal{E}$, Human.Animal); *dec*(Human, Surgeon.Farmer); *dec*(Animal, Dog.Horse);
Domain constraints:
dom(be_a_surgeon) = (Surgeon); *dom*(be_a_dog) =(Dog); *dom*(sing) = (Human); *dom*(bark) = (Dog).
They will be used as it is done in the algorithm, in their internal form: be_a_surgeon(T_{Surgeon}), be_a_dog(T_{Dog}), etc.

Ok, let's begin. For each word we give the current translation of the sentence in the MRIF and all the unifications of types done by the algorithm. We note T_A the conceptual type of each expression A of the MRIF (subcategorization of i/o type hierarchy).

Step 1, sentence composed : "A..."

$[\![\mathrm{S}]\!] = [[\![\mathrm{NP}]\!], [\![\mathrm{VP}]\!]] = [[[\![\mathrm{Det}]\!], [\![\mathrm{CN}]\!]], [\![\mathrm{VP}]\!]] = [[\lambda P \lambda Q \; \exists x \; [P, x] \wedge [Q, x], [\![\mathrm{CN}]\!]], [\![\mathrm{VP}]\!]]$
$T_S = T_{[NP,VP]} \equiv_u \underline{t_3} \Rightarrow \{ \; T_{NP} \equiv_u < \underline{t_1}, t_3 >, T_{VP} \equiv_u t_1 \; \}$
$T_{[Det,CN]} = T_{NP} =< t_1, t_3 > \Rightarrow \{ \; T_{Det} \equiv_u < \underline{t_4}, < t_1, t_3 >>, T_{CN} \equiv_u t_4, \; T_P \equiv_u t_4, \; T_Q \equiv_u t_1 \; \}$
$\{ \; T_{[P,x]} \equiv_u \underline{t_6}, \; T_{[Q,x]} \equiv_u \underline{t_7} \; \} \Rightarrow \{ \; t_4 \equiv_u < \underline{t_5}, t_6 >, \; t_1 \equiv_u < t_5, \underline{t_7} >, \; T_x \equiv_u t_5 \; \}$
$P \equiv_u < \underline{v_1}, t_4 >=< v_1, < t_5, t_6 >> \; ; \; Q \equiv_u < \underline{v_2}, t_1 >=< v_2, < t_5, t_7 >>; x \equiv_u < \underline{v_3}, t_5 >$

Step 2, sentence composed : "A surgeon..."

$[\![S]\!] = [[\lambda < v_1, < t_5, t_6 >> \ \lambda < v_2, < t_5, t_7 >> \ \exists x \ [< v_1, < t_5, t_6 >>, < v_3, t_5 >] \wedge [< v_1, < t_5, t_6 >>, < v_3, t_5 >], \lambda v \text{ be\_a\_surgeon}(v)], [\![VP]\!]]$

Call to domain constraint be_a_surgeon(T_{Surgeon}) $\Rightarrow T_v \equiv_u T_{\text{Surgeon}}$
$T_{CN} =< t_5, t_6 > \Rightarrow t_v = T_{\text{Surgeon}} \equiv_u t_5$
$v \equiv_u < \underline{v_4}, T_{\text{Surgeon}} >$

Step 3, sentence composed : "A surgeon sings."

$[\![S]\!] = [[\lambda < v_1, < T_{\text{Surgeon}}, t_6 >> \ \lambda < v_2, < T_{\text{Surgeon}}, t_7 >> \ \exists x \ [< v_1, < T_{\text{Surgeon}}, t_6 >>, < v_3, T_{\text{Surgeon}} >] \wedge [< v_1, < T_{\text{Surgeon}}, t_6 >>, < v_3, T_{\text{Surgeon}} >], \lambda < v_4, T_{\text{Surgeon}} > \text{be\_a\_surgeon}(< v_4, T_{\text{Surgeon}} >)], \lambda y \ \text{sing}(y)]$

Call to domain constraint sing(T_{Human}) $\Rightarrow T_y \equiv_u T_{\text{Human}}$
$T_{VP} =< T_{\text{Surgeon}}, t_7 > \Rightarrow t_y = T_{\text{Human}} \equiv_u T_{\text{Surgeon}}$: **this unification succeeds because the domains *Human* and *Surgeon* are not disjoint**. The process goes on. We note $T_{\text{Sur} \cap \text{Hum}}$ the result of the unification.
$y \equiv_u < \underline{v_5}, T_{\text{Sur} \cap \text{Hum}} >$

Now the translation of the sentence is completely known:
$[\![S]\!] = [[\lambda < v_1, < T_{\text{Sur} \cap \text{Hum}}, t_6 >> \ \lambda < v_2, < T_{\text{Sur} \cap \text{Hum}}, t_7 >> \ \exists x \ [< v_1, < T_{\text{Sur} \cap \text{Hum}}, t_6 >>, < v_3, T_{\text{Sur} \cap \text{Hum}} >] \wedge [< v_1, < T_{\text{Sur} \cap \text{Hum}}, t_6 >>, < v_3, T_{\text{Sur} \cap \text{Hum}} >], \ \lambda < v_4, T_{\text{Sur} \cap \text{Hum}} > \text{be\_a\_surgeon}(< v_4, T_{\text{Sur} \cap \text{Hum}} >)], \ \lambda < v_5, T_{\text{Sur} \cap \text{Hum}} > \text{sing}(< v_5, T_{\text{Sur} \cap \text{Hum}} >)]$

The analysis is ended and there is no unification failure between types: the sentence is conceptually coherent in terms of domains. Let's note that we have obtained the result without any (expensive) λ−reduction but only with type checking.

If the sentence had been, for example, *A surgeon barks*, **the unification $T_{\text{Surgeon}} \equiv_u T_{\text{Human}}$ would have been replaced by $T_{\text{Surgeon}} \equiv_u T_{\text{Dog}}$ which is impossible because the domains *Surgeon* and *Dog* are disjoint**. In this case, the unification failure implies the backtracking of the process. In analysis mode, the word *barks* is removed, and the system switches to synthesis mode: it tries all the other verbs and stores the ones which don't imply unification failure in the algorithm, in order to propose them to the user. In synthesis mode, the word *barks* of course cannot be proposed because it causes a fail.

Acknowledgements

We are very grateful to Robert Pasero and Paul Sabatier for their numerous and helpful discussions on our work on ILLICO.
Parts of this work were funded by the French *Ministère de la Recherche et de la Technologie* (*Intelligent Interfaces* program, ILLICO project) and the European commission *TIDE* program (Technology Initiative for Disabled and Elderly people, KOMBE project), and the Conseil Général des Bouches-du-Rhone (EREL project).

References

[Aho et al. 79] Aho A.V., Beeri C., Ullman J.D., 1979: The Theory of Joins in Relational Databases. *ACM Transactions on Database Systems*, Vol. 4, n° 3.

[Battani et al. 91] Battani G., Pasero R., Sabatier P., 1991: Le projet ILLICO, Interfaces en langage naturel et graphique, *Proc. of Conférence Génie Linguistique*, Versailles.

[Boïde 94] Boïde O.S., 1994: Interrogation de la banque de données ORBIS en langue arabe, mémoire de DEA, Laboratoire d'Informatique de Marseille, Univ. Aix-Marseille II.

[Brachman 85] Brachman R.J., 1985: An overview of the KL-ONE Knowledge Representation System, *Cognitive Science* 9.

[Colmerauer 75] Colmerauer A., 1975: Metamorphosis Grammars, *Natural language with computers*, Bolc L.Ed., Springer-Verlag.

[Colmerauer et al. 82] Colmerauer A., Kittredge R., 1982: ORBIS. *COLING Conference.*

[Cruse 86] Cruse D. A., 1986: *Lexical Semantics*, Cambridge Textbooks in Linguistics, Cambridge University Press.

[Giannesini et al. 85] Giannesini F., Kanoui H., Pasero R., Van Caneghem M., 1985: *Prolog*, InterÉditions, Addison Wesley.

[Godbert 94] Godbert E. 1994: Modelling domain and connectivity constraint in natural language processing, in Carlos Martin-Vide Ed., *Current issues in Mathematical Linguistics*, North-Holland Elsevier Science B.V., Amsterdam, The Netherlands.

[Godbert 98] Godbert E., 1998. EREL : a multimedia CALL system devoted to children with language disorders. In Keith Cameron Ed.,*Multimedia CALL: Theory and Practice*, Elm Bank Publications, Exeter, England.

[Milhaud 94] Milhaud G., 1994: *Un environnement pour la composition de phrases assistée*, Thèse de doctorat, Laboratoire d'Informatique de Marseille, Univ. Aix-Marseille II.

[Montague 73] Montague R., 1973: *The proper treatment of quantification in ordinary English*, in Hintikka et al. (ed.), Approaches to natural language, p. 221–242, Reidel, Dordrecht.

[Mouret 98] Mouret P., 1998: *Etude et intégration de contraintes contextuelles dans la compréhension automatique du francais*, Thèse de doctorat, Laboratoire d'Informatique de Marseille, Univ. de la Méditerranée, to appear.

[Pasero et al. 94] Pasero R., Richardet N. and Sabatier P., 1994: Guided Sentences Composition for Disabled People, *Proc. of Fourth Conference on Applied Natural Language Processing*, Stuttgart.

[Pasero et al. 95] Pasero R., and Sabatier P., 1995: Guided Sentences Composition : Some problems, solutions, and applications, *5th International Conference on Natural Language Understanding and Logic Programming* (NLULP), Lisbon.

[Sabatier 89] Sabatier P., 1989: Interfaces en langage naturel : du traitement du non attendu à la composition de phrases assistée, *Annales des Télécommunications*, 44, n°1-2, CNET.