



Gouvernement du Canada
Ministère des Communications

Government of Canada
Department of Communications

Le Centre canadien de recherche sur l'informatisation du travail
Canadian Workplace Automation Research Centre

Bibliothèque Queen

2
Grammar Bidirectionality
through
Controlled Backward Deduction

/ Marc Dymetman /

Pierre Isabelle

CO

QUEEN
P
308
.D965
1990
c.2

Canada

Queen

P
308
D965
1990
c.2

Industry Canada
Library Queen
JUL 24 1998
Industrie Canada
Bibliothèque Queen

²
Grammar Bidirectionality
through
Controlled Backward Deduction

/ Marc Dymetman/
Pierre Isabelle

COMMUNICATIONS CANADA
FEB 14 1992
LIBRARY - BIBLIOTHÈQUE

Canadian Workplace Automation Research
Centre
Department of Communications Canada
Laval

March 90

This report also appears as:

Dymetman, Marc and Pierre Isabelle. 1990. Grammar Bidirectionality through Controlled Backward Deduction. In *Logic and Logic Grammars for Language Processing*, eds. Saint Dizier, P. and S. Szpakowicz. Chichester, England: Ellis Horwood.

Cat. N°: Co 28-1/48-1990E

ISBN: 0-662-17808-4

The views expressed in this report are those of the authors only.

Ce rapport est aussi disponible en français.

DD 9600836
DL 1116 8114

P
308
D965
1990
C.2-

Abstract

Grammars can be seen as logical theories, and parsers as special purpose theorem provers for these theories. *Generators* can and should be viewed in the same way. We present an approach in which the same goal-oriented theorem prover is used for parsing and generation. Grammar rules (ie axioms of the theory) are annotated with *control directives* addressed to the parsing mode or to the generation mode. This ensures a strict declarative equivalence of the two modes, as well as linguistic perspicuity, while allowing the linguist to specify processing behaviour in both modes with some flexibility. Such flexibility turns out to be highly valuable, considering our practical aims: building a bidirectional English-French translation system capable of handling real texts in a limited domain.

1. Introduction

Machine translation is a natural domain of application for bidirectional grammars, on economic grounds - using one system to translate from Language1 to Language2 or conversely - as well as on theoretical grounds - maintaining a clear linguistic specification of which parsing and generation are viewed as computational realizations.

Historically, one of the first attempts at reversible grammatical description was the Q-Systems formalism (Colmerauer 71), Prolog's forerunner, and the algorithmic backbone of the TAUM-METEO translation system. Colmerauer's paper gives an example of reversible processing, but the applicability of the approach was limited by the unavailability of full unification and of a clear logical status for Q-Systems, both of which were later achieved with the advent of Prolog.

The CRITTER project (Isabelle et al 88) aims at bidirectional English/French translation of agricultural market reports produced by the Canadian Department of Agriculture.

CRITTER makes use of 2 bidirectional grammars - one for English, one for French - , 2 bidirectional monolingual lexicons (in the sense of allowing for efficient string access and also efficient semantic access), and a bidirectional transfer lexicon - describing the correspondences between language-dependent semantic units.

As an example of the current coverage of the system, consider the following English sentence:

it is reported by the OHMB that the market remained strong in Toronto ;

CRITTER will translate this sentence nondeterministically into any one of the following French sentences:

{ (f1) l'OHMB signale que le marché est resté soutenu à Toronto,
(f2) il est signalé par l'OHMB qu'à Toronto le marché est resté
soutenu,...}.

Retranslating (f1) (for instance) into English will result in the set of following English sentences:

{ (e1) the OHMB reports that in Toronto the market remained strong,
(e2) it is reported by the OHMB that the market remained strong in
Toronto, (e3) the market is reported by the OHMB to have remained
strong in Toronto,...} ;

Of these, (e2) is identical with the original sentence, and (e1) and (e3) can be seen as its paraphrases .

Paraphrase and Translation. While CRITTER is primarily intended as a translation system, it can also function, thanks to its bidirectional components, as a paraphrasing system for English or for French.

Paraphrase and translation are closely related: whereas *paraphrase* relates expressions of one language which share a semantic form, *translation* relates expressions of two languages which either share a semantic form (*interlingual* approach), or have directly relatable semantic forms (*transfer* approach).

Paraphrase and Bidirectionality. Consider the following sentences¹:

(p1) Jack seems to be given a black cat by Mary
(p2) Mary seems to give a black cat to Jack
...
(p3) It seems that Jack is given a black cat by Mary
(p4) It seems that Mary gives a black cat to Jack
...

¹ For purposes of exposition, we will use examples featuring lexical material with little connection to agricultural market reports.

In the CRITTER system, these sentences are all mapped by the parsing process onto the same "shallow" semantic representation (see Fig. 1). The reader will find detailed explanations of these representations in (Isabelle and al. 1988).

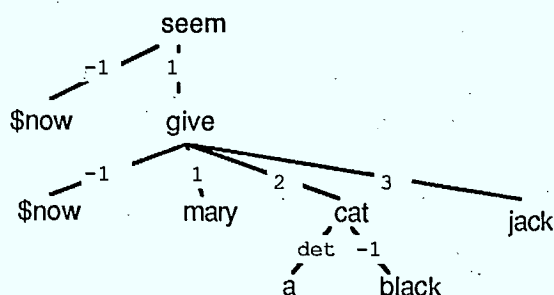


Fig. 1. In this semantic representation, the labels 1,2,3 denote argument positions relative to a predicate: for instance 'mary' is the first argument of 'give', etc.... The labels -1, -2, -3 (only the first of which appears in the example) have the same meaning, except that they must be read "bottom-up": thus '\$now' is a one-predicate argument (applying to an object of type event, and meaning that this event has present time location). In the example, one of the two instances of the '\$now' predicate applies to the giving event. This notational device permits us to keep a uniform predicate-argument form for the semantic representation, while retaining a tree structure. This tree structure has a well defined root, unlike an oriented graph structure, for which such a root is not defined (for more details, see [Isabelle et al. 1988]).

According to the definition given above, (p_1) , (p_2) , ..., can be said to be *paraphrases* of each other, relative to a certain specific mapping of strings to semantic structures.

The bidirectionality of the system appears when the generation process is fed the semantic structure (Fig. 1): each of the paraphrases (p_i) (and

only these) will then be produced nondeterministically by the generation process.

2. The problem of computational reversibility

In abstract terms, a grammar is a specification of a recursively enumerable set of (finitely encoded) linguistic structures A_v , classified according to their (terminal or non-terminal) syntactic category v .

These linguistic structures are assumed to encode information of various types such as: syntax, semantics, string of characters, which can be accessed through functions *syn*, *sem*, *string*. Thus one writes: $\text{syn}(A) = \text{Syn}$, or alternatively in "dot" notation²: $A.\text{syn} = \text{Syn}$ to express the fact that the syntactic content of A is Syn .

We shall assume that the linguistic structures S of category s have nonempty values for at least $S.\text{sem}$ and $S.\text{string}$.

The problem of *computational reversibility* is to derive efficient³ programs Parse_G and Generate_G (possibly identical) from the specification of a grammar G , such that:

- If Parse_G is given *String* as input, it enumerates all well-formed S such that $S.\text{string} = \text{String}$,
- If Generate_G is given *Sem* as input, it enumerates all well-formed S such that $S.\text{sem} = \text{Sem}$.

² This notation is a variant of the functional notation used in LFG (Kaplan and Bresnan 1982), or in PATR (see Shieber 1986). See also (Dymetman 1989).

³ Obviously, finding any old program meeting the requirements is no challenge: filtering a recursively enumerable set through a recursive criterion will produce another r.e. set. On the other hand, finding efficient programs can be a difficult task, and can depend on the form of the initial specification: think of public key cryptography, where *public key* and *private key* are functionally equivalent specifications, but where only from *private key* is it possible to derive both efficient encoding and decoding schemes.

3. Grammars as logical theories

It has become popular to consider grammars as special forms of axiomatic theories, related to the subset of first-order logic known as *Horn clause* (or *definite clause*) logic.

Glossing over the many differences of detail between formalisms, grammar rules can be seen as logical axioms of the following general format:

$$(1) \quad nt(NT) \leq v_1(V_1), v_2(V_2), \dots, v_n(V_n), \text{constraint}(NT, V_1, \dots, V_n).$$

This says that if V_1, \dots, V_n are well-formed linguistic structures of categories v_1, \dots, v_n , and constraint *constraint* holds between NT, V_1, \dots, V_n , then NT is a well-formed linguistic structure of category *nt*.

Note that the literals on the right-hand side of (1) are unordered: it is the responsibility of *constraint* to ensure, for example, that $Nt.string$ is the concatenation of $V_1.string, \dots, V_n.string$, as well as the syntactic and semantic compositionality relations holding between NT and its daughters.⁴

Hereafter, we assume that (1) is of the standard Definite Clause type, ie that: i) the intended interpretations are Herbrand interpretations (Kowalski 79b), ii) equations of the form $X.function=Y$ are abbreviations for expressions of the form: $X = t(\dots Y\dots)$, where $t(\dots Y\dots)$ is a term containing variable Y at some level of embedding.

⁴ Many specificities of the various formalisms are being ignored in this account; most conspicuously: i) the *formal* nature of the objects denoted by variables (*DAGs* in so-called "unification grammars", *terms* in so-called "logic grammars"; ii) the nature of *constraints*: can they be recursively defined as in DCG or must they be defined with limited expressive means (eg, by means of equations, perhaps with the exception of the string concatenation relation, viewed as a primitive), allowing for stronger decidability results.

Under this simplifying assumption, the set of well-formed linguistic structures defined by the grammar is the set of ground terms NT such that nt(NT) is a valid consequence of the axioms.

Classes of deduction strategies. Most deduction strategies used for grammatical processing fall into one of three classes, according to their fundamental inference rule:

Backward deduction:
$$\frac{\text{<-- } B_1 \dots B_m, B'_i \text{ :- } C_1 \dots C_n}{\theta[\text{<-- } C_1 \dots C_n \ B_1 \dots B_{i-1} \ B_{i+1} \dots B_m]}$$

Forward deduction:
$$\frac{A \text{ :- } B_1 \dots B_n, B'_1 \text{ <--}, \dots, B'_n \text{ <--}}{\theta[A \text{ <-- }]}$$

Lemma deduction:
$$\frac{A \text{ <-- } B_1 \dots B_n, B'_i \text{ <--}}{\theta[A \text{ <-- } B_1 \dots B_{i-1} \ B_{i+1} \dots B_n]}$$

Remarks on notation:

- i) In each case, θ is the substitution which performs the most general unification between B_i and B'_i .
- ii) The difference between the '<--' and the ':-' symbols is that '<--' corresponds to a clause which has been added to the "agenda", while ':-' corresponds to a program axiom.

Backward deduction and forward deduction have long been part of the logic programming tradition, while what is here called "lemma deduction" was introduced in (Pereira & Warren 83), under the name "Earley deduction"⁵ with linguistic applications in mind, in analogy with the Earley parsing algorithm (Earley 1970).

In both forward and lemma deduction, a serious control issue is to be able to provide, from an a priori analysis of potential goals and the properties of the theory, efficient filters which eliminate from further

⁵ What these authors call Earley deduction is the instance of our lemma deduction (which corresponds to their reduction rule) using a specific prediction rule, which we shall call the Earley-type prediction rule (see note 3 below).

consideration all but a few axiom instantiations, the rest being unable to participate in successful proofs:

- *Universal filters*: thus forward and Earley deduction applied to parsing must rely strongly on universal projection relations (such as are imposed by a context free grammar) that hold between the string associated with a node and the strings associated with its daughters, in order to eliminate most instantiations of the lexical axioms⁶ or in order to express strong *predictions*⁷ on the instances of grammar rules which can be of potential use in a proof. Similarly, applying Earley deduction to generation as in (Shieber 88) requires that certain strong hypotheses (*semantic monotonicity*) be made on the relations between a node's semantics and its daughters' semantics.

- *Reachability relations*: other kinds of filters that have been applied involve the use of *reachability tables* (Kay 1980), or *links*, which are precompiled constraints holding between the linguistic structure associated with a node and the linguistic structure associated with some descendant of this node in the derivation tree (eg constraints on syntactic categories for parsing or on semantic structures for generation). Such methods have been used for parsing in BUP (Matsumoto & al. 1983) as well as for generation in BUG (Van Noord 1989).

⁶ The lexicon being seen as the set of "unconditional" grammar axioms.

⁷ Such as the following *instantiation* rule of (Pereira & Warren 1983) (called *prediction* rule in [Shieber 88]):

Earley-type prediction:
$$\frac{A \leftarrow B_1 \dots B_n, \quad B'_1 :- C_1 \dots C_m}{\theta[B_1 \leftarrow C_1 \dots C_m]}$$

On the same basis, one can introduce a *lc-type* prediction rule inspired from left-corner parsing (which can be considered as belonging either to the forward deduction or to the lemma deduction family, depending on versions):

lc-type prediction:
$$\frac{B'_1 \leftarrow, \quad A :- B_1 \dots B_m}{\theta[A \leftarrow B_1 \dots B_m]}$$

(Remark on notation: As above, θ is the substitution which performs the most general unification between B_1 and B'_1).

By contrast, in *backward* deduction, the "relevance" of each inference step to the goal theorem is a natural consequence of the approach; this means that one can somewhat relax the requirements made on the relation between the semantics of a given node and the semantics of its daughters without impairing the generation process⁸. The main control⁹ issue with backward deduction is the question of how to select the goal literal B_i which will be expanded at the next inference step. Among the parameters of this choice are the potential non-determinism and the level of instantiation of each literal compared to the others.

4. Backward Deduction and Computational Reversibility

Ordering Problems. Definite Clause Grammars (DCG's) together with their standard Prolog interpretation constitute the best-known instance of the backward deduction scheme. Consider a DCG rule such as (2), and the corresponding Prolog procedure (3):

<pre>(2) s(S) --> np(NP), vp(VP), {combine1(NP, VP, S)}.</pre>	<pre>(3) s(S, L1, L3) :- np(NP, L1, L2), vp(VP, L2, L3), combine1(NP, VP, S).</pre>
--	--

Assume that *combine1* defines the relation between the structures of each constituent, stipulating in particular how *S.sem* is built out of *NP.sem* and *VP.sem*.

Notice that under interpretation (3), rule (2) conflates two different types of ordering: a) the relative order of the strings associated with the non-terminals (the *np* precedes the *vp*); and b) the order in which the corresponding Prolog goals are evaluated (the *np* goal is evaluated before the *vp* goal). While non-terminal ordering is an integral part of the grammatical specification, goal ordering only has to do with control.

⁸ This can be useful for handling idioms (see section 7).

⁹ But see the discussion of *left-recursion* in section 5.

Procedure (3), interpreted in the standard way, is adequate for parsing purposes, but may well give rise to a hopelessly inefficient generation program. For example, consider a grammar based on a "lexicalist" model¹⁰. In such a grammar, the mapping between the semantic structures and the syntactic configurations is determined in the lexicon. In particular, the relation between the sentence semantic structure S.sem and the subject semantic structure S.subj.sem can be determined only through the lexical entry of the main verb. Given the existence of phenomena like *raising*, S.subj.sem may be embedded arbitrarily deeply within S.sem:

- (4) a) Max seemed to begin to be tougher to convince.
 b) past'(seem'(begin'(tougher'(convince'(one',Max')))))

The same linguistic models postulate a much more straightforward relationship between S.sem and VP.sem (or more generally between X.sem and X.head.sem).

Procedure (3) thus turns out to embody a very poor strategy for generation, in which case it is called with only S.sem known.

Control annotations and task-specific compilation. In order to use the same grammar for both parsing and generation tasks, a task-specific control of the deduction process is needed. For that purpose, grammatical rules are separated into two components: 1) a description of the (declarative) grammatical content; and 2) a description of how deduction is to be controlled relative to each particular processing task. These two components thus correspond to the logic/control separation advocated in (Kowalski 1979a).

DCG rules are augmented in the following way: a) goals are prefixed with unique identifiers; and b) rules include control annotations which

¹⁰ For an introduction to the linguistic notions used in the sequel see (Sells 1985). For a linguistic theory to which our approach is related, see (Pollard and Sag 1987).

stipulate how the goals are to be processed for each task. For example, the DCG rule (5) can be augmented as in (6):

<pre>(5) a(A) --> b(B), c(C), d(D), {y(A,B,C,D)}, {z(A,B,C,D)}.</pre>	<pre>(6) a(A) --> 1: b(B), 2: c(C), 3: d(D), 4: {y(A,B,C,D)}, 5: {z(A,B,C,D)}, << parse: order: 1 < 2 < 4 < 3 < 5; gen: order: 5 < 4 < 2 < 1 freeze: 3 until: cond(A,B,C,D) >></pre>
---	---

In the control annotation "<< ... >>", *parse* and *gen* refer to two different processing tasks. For task *parse*, goals 1, 2 and 3 are to be fired in that order (the order of the remaining goals is considered irrelevant). For task *gen*, goal 2 is to be fired before goal 1; moreover goal 3 will be delayed through *goal freezing* (Colmerauer 82) until the precondition *cond* is satisfied.

The annotated rules are precompiled into task-specific Prolog programs¹¹. Given the nature of the control annotations, these programs are guaranteed to be declaratively equivalent, but each one is tuned for optimal performance with respect to its target task (see Fig. 2).

¹¹ These programs are interpreted through an extended Prolog interpreter, which supports goal-freezing (see Cochard and Dymetman (to appear)).

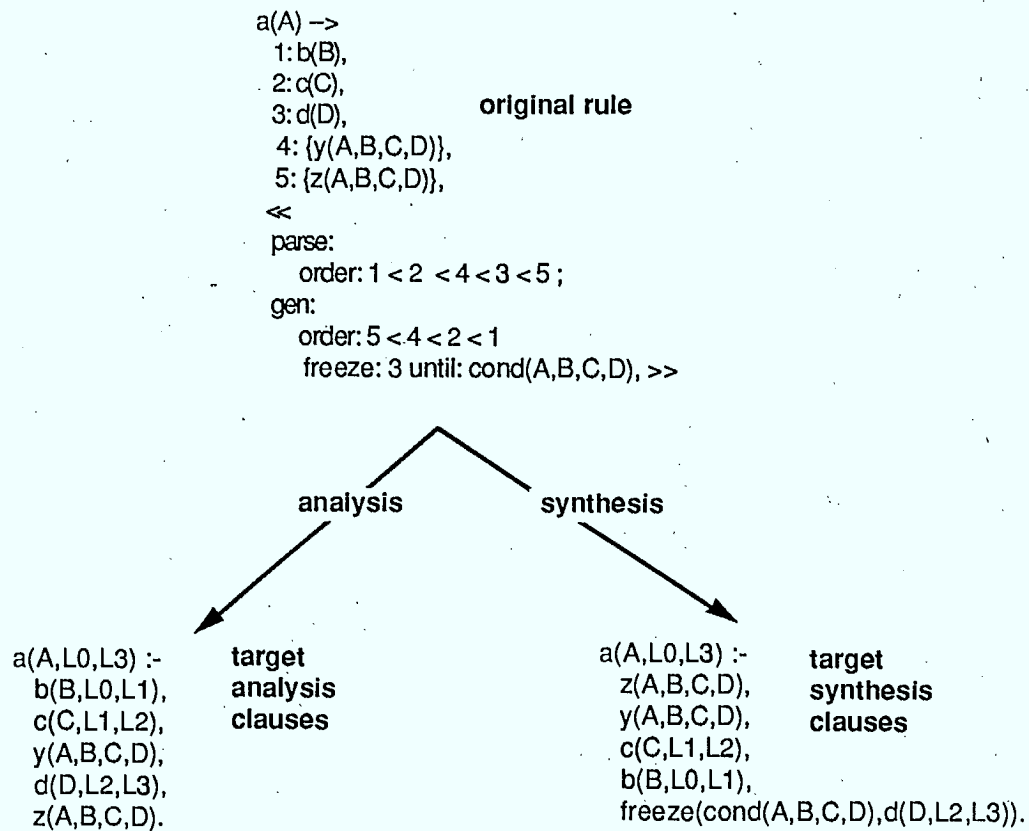


Fig. 2

Rule (2) can be reformulated as:

```
(7)  s(S) -->
      1: np(NP),
      2: vp(VP),
      3: {combinel(NP, VP, S)},
      <<
      parse:
        order: 1 < 2 < 3 ;
      gen:
        order: 3 < 2 < 1 >>
```

This rule will be compiled into the parsing Prolog procedure (3), but into the generation procedure (8):

```
(8)  s(S, L1, L2) :-
      combinel(NP, VP, S),
      vp(VP, L3, L2),
      np(NP, L1, L3).
```

Thus, goals are reordered for generation: *combinel* is applied first, resulting in the instantiation of *VP.sem*; then *vp* is fired and eventually, lexical access will cause *NP.sem* to become instantiated. Finally *np* is called with known semantics.

Surprisingly enough, this simple *static* reordering of goals, *local* to rules, will in many cases be sufficient to ensure that the grammar performs reasonably well in both directions.

Asynchronous evaluation. However, because conflicting demands may arise as to the optimal ordering, it is sometimes necessary to take advantage of the dynamic *nonlocal* reordering of goals made possible by the use of goal freezing. For example, given the order just suggested for the generation of *s*, a blind hypothesis is made on the verb inflection, since the agreement information is available only once the subject *np* was generated. Goal freezing provides a solution to this problem:


```

(9)  v(V) -->
      1: rad(RAD),
      2: suff(SUFF),
      3: {combine3(RAD, SUFF, V)},
      <<
      parse:
        order: 1 < 2 < 3 ;
      gen:
        order: 3 < 1
        freeze: 2 until: nonvar(SUFF.agr) >>

```

In the generation mode, *combine3* is applied first. Among other things, this operation instantiates *RAD.sem* and unifies *V.agr* with *SUFF.agr*. The verb radical can then be generated (call to *rad*) through an access to the lexicon. The generation of the inflectional suffix (goal *suff*), however, is postponed until *SUFF.agr* is instantiated. This condition will be met only after enough of the subject noun phrase has been generated for *NP.agr* to become instantiated. *SUFF.agr* will at the same time become instantiated, by the play of agreement rules (built into *combine1* in the *s* rule) and the transmission of head features from VP to V.

Static ordering, Dynamic ordering and Backward deduction. Let us now reexamine how i) *static goal ordering* and ii) *dynamic goal ordering through asynchronous evaluation* mesh with the backward deduction rule of section 3, which is reproduced here:

$$\frac{\langle \text{-- } B_1 \dots B_m, B'_i \text{ :- } C_1 \dots C_n}{\theta[\langle \text{-- } C_1 \dots C_n \ B_1 \dots B_{i-1} \ B_{i+1} \dots B_m]}$$

We now assume that the goal list appearing on the right-hand side of a clause is *ordered*, and we need to describe how the selection of B_i is done, and how the goal list resulting from the deduction step is ordered. The connection between static ordering, dynamic ordering and the previous deduction rule is given by the following specifications:

- selection of B_i : among the $B_1 \dots B_m$, there may be a B_f which is a "frozen" goal whose firing precondition is now met; in this case take $B_i = B_f$, otherwise take $B_i = B_1$.

- ordering of resulting goal list:

i) divide the body $C_1 \dots C_n$ of the program axiom into two lists: $L_s = [S_1 \dots S_k]$ and $L_f = [F_{k+1} \dots F_n]$, where the F_i 's are the frozen goals in the body of the axiom, and where the S_i 's are the remaining goals, ordered according to their order annotations (for the processing task considered). In contrast to that of L_s , the order of L_f is not considered significant;

ii) output then as the resulting goal list:

$\theta[\leftarrow S_1 \dots S_k \ B_1 \dots B_{i-1} \ B_{i+1} \dots B_m \ F_{k+1} \dots F_n]$.

5. Handling left-recursion

A well-known problem with parsers using the backward deduction approach is that of handling left-recursive constructions in the grammar.

Consider the following rules:

```
(10) np(NP) -->
      1: np(NP'),
      2: pp(PP),
      3: {combine2(PP, NP', NP)},
      <<
      parse:
        order: 1 < 2 < 3 ;
      gen:
        order: 3 < 2 < 1 >>.
np(NP) -->
  nl(NP').
```

The intent of the *np* rule given above is to incorporate within a *np* an unspecified number of prepositional modifiers (*the hotel in Vancouver with a Chinese roof*). *nl* is the "bare" *np* (*the hotel*), and *combine2* builds the syntactic and semantic structures of NP from those of PP and NP'.

The obvious problem with this rule, however, is that it is "left-recursive": although its declarative reading is perfectly natural and directly represents the intended meaning of the rule, the standard DCG translation results in a loop when used in analysis. For this reason, no DCG programmer ever writes such rules in the way shown in (10). Rather,

he transforms them into some equivalent rules, which although more complex, will display the desired computational behavior (see infra Fig. 3).

On the other hand, it is interesting to observe that rule (10), when used in synthesis, will work perfectly well if only one takes care - as in the previous vp example - to order the *combine2* goal before the *pp* and *np* goals.¹²

Fortunately, there is a way out of this difficulty: the grammarian should be allowed to write the *np* rule in the natural way, and the compiler should be made to perform the transformations needed for parsing and generation.

¹² This is not to say that a problem analogous to that of left-recursion cannot in principle appear in the case of generation, but rather that it does not appear if grammars respect the conditions of section 6 below. The problem of eliminating left-recursion in generation for grammars which do not respect such conditions is beyond the scope of this paper.

This is precisely what is done in the CRITTER system, as illustrated in Fig. 3:

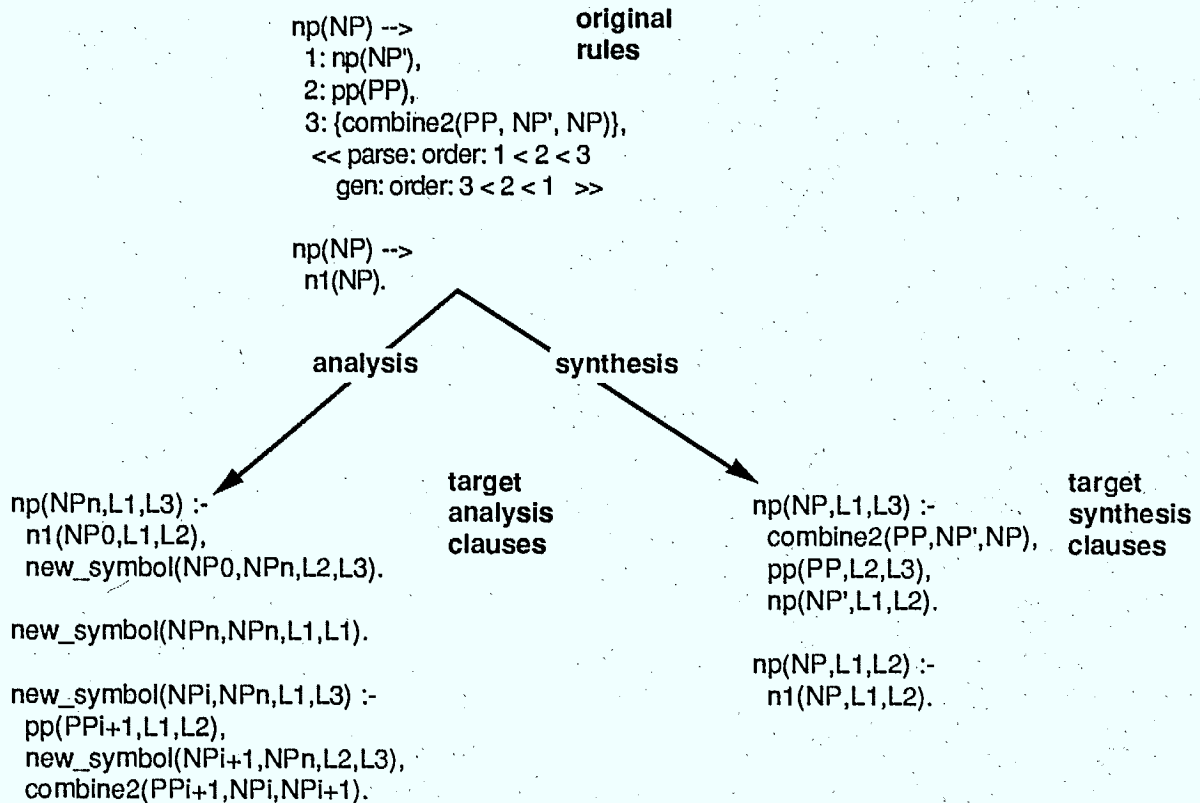


Fig. 3

In the current implementation, it is necessary to flag left-recursive procedures¹³ for the compiler. The analysis-specific compilation will then introduce an auxiliary nonterminal *new_symbol*, and perform a transformation leading to the target analysis clauses shown in the figure. Space is lacking here to explain the details of the transformation, but it can be formally demonstrated that the resulting clauses keep the intended meaning of the rule unchanged¹⁴.

¹³ As usual in Prolog jargon, a procedure is the set of clauses defining a predicate.

¹⁴ The process is akin to techniques for putting a context-free grammar into some normal form. The delicate part, however, is the way in which the constraints between variables are to be transferred.

As for the synthesis clauses, they are similar to the original rules, apart from the fact that the order annotations in these have been used to order subgoals, as in the case of Fig. 2.

6. Conditions for reversibility

Our experience has shown that the techniques presented above are sufficient to ensure the practical bidirectionality of English and French grammars applied to real texts of moderate complexity. It is therefore interesting to step back and consider what are the general grammar properties that make this possible.

Semantic normal forms. As has been noted by several researchers , the problem of reversibility is sensitive to certain basic assumptions that are made concerning the level of semantic representation. Thus, if a semantic representation is taken to be an abstract FOL *proposition* (ie an equivalence class of (logically equivalent) FOL *statements*), one cannot (*bijectively* and *effectively*) encode this proposition into a finite representation. As shown in (Appelt 87), this results in insuperable difficulties for the generation process.

The approach presented here is only applicable to cases where semantic objects *do* possess finitely encoded normal forms. These semantic objects are not as abstract as logical propositions: roughly speaking, they are just as abstract as needed to "disregard" the grammatical idiosyncrasies of two (or more) different languages.¹⁵

Grammar properties sufficient for reversibility. We now turn to the problem of generation, and assume, without loss of generality, that derivation trees are binary branching at most.

¹⁵ The notion of paraphrase that we want to characterize is of course much more restrictive than that implied by FOL equivalence: in this context "paul drinks" and "paul drinks and ideas sleep or ideas do not sleep" cannot be considered valid paraphrases, no more than "paul drinks" and "paul boit et les idées dorment ou les idées ne dorment pas" can be considered valid translations.

The following four grammar properties are then sufficient to ensure that a local ordering of goals (internal to a grammar rule) will result in a complete generation procedure (in the sense of enumerating all sentential structures S having a given semantic structure):

(Prop.1) All linguistic structures A have semantic content Sem as well as syntactic content Syn.

(Prop.2) In a derivation tree, if linguistic structure A has daughters B and C, then the semantics of one of the daughters - say C - (the head) is identical to that of A (case of argument incorporation: B is an argument) or is "contained" in it - according to some partial order on semantic structures admitting of no infinite descending chains - (case of modifier incorporation: B is a modifier).

(Prop.3) With the same notation as above, if the semantics of A is known, and if the semantics and syntax of C are known, then the semantics of B is uniquely determined. (see example (8) above, where the semantics of the subject NP (B) will be determined only once the VP (C) will have "taken a decision", recorded conjointly in its syntax and semantics, concerning the mapping of arguments onto syntactic positions.)

(Prop.4) There exists some device in the grammar to denote the bar level of a phrase¹⁶ - either through the names of nonterminals or through some index internal to the linguistic structures, this bar level being strictly incremented for each argument incorporation.

Sketch of completeness proof: Props. 2 and 4 guarantee that it is possible to transmit semantic information from a mother to its head

¹⁶ See (Sells 1985, p. 28).

daughter and that repeating this operation from head to head will eventually reach a lexical structure C, that is a structure which will be completely known (syntax and semantics) by simple lookup. Prop. 3 will then ensure that the semantics of at least the non-head sister B of C will be known; by induction, B will eventually be completely known, and the mother A of B and C will be completely known from B and C; thus the complete determination of all nodes of the derivation tree will propagate upwards until finally S becomes completely known.

7. Further research

Idioms. It is possible to relax the requirement that the semantics of daughters and mother be related through inclusion (which is essentially the *semantic monotonicity* condition of [Shieber 88]). For instance, one could introduce an "idiom-reduction" predicate *idiom_red*(*Sem_init*, *Sem_fin*). Starting from the "initial semantics" of A, obtained in the standard compositional way from the "final semantics" of its daughters B and C, *idiom_red* would produce the "final semantics" of A, by looking up a table of "idiomatic formations". Such a device would permit various normalization operations, such as reducing *kick(bucket)* to *die*, or normalizing "dormir volontiers"¹⁷ (the French equivalent of the notorious Dutch "graag slapen") into "aimer dormir"¹⁸.

Such a mechanism will not impede reversibility, provided *idiom_red*, used in reverse to go from final semantics to initial semantics, has a limited non-determinism. On the other hand, a similar device would be difficult to implement in a forward or Earley deduction scheme, for the sentence semantics would hold no obvious relation to the semantics of the lexemes used to build it, a seemingly necessary property of these approaches. (But see (Van Noord 1989) for an approach to idiomaticity in the forward deduction paradigm).

¹⁷ Literally: "sleep willingly".

¹⁸ Literally: "like to sleep".

Computational reversibility and linguistic reversibility. This paper addresses the problem of *computational reversibility*: deriving efficient analysis and synthesis programs from the same declarative grammar.

Assuming the computational reversibility problem has been solved, there remains the problem of *linguistic reversibility*. This is the problem of defining a bidirectional grammar in a way *linguistically adequate* for both parsing and generation. As the following remarks will indicate, this is far from being a trivial question.

A grammar BG which is bidirectional has to be much closer to *observational adequacy* than would have to be two different grammars, one for analysis (AG), the other for synthesis (SG). This is because AG can be overly permissive on the strings it accepts, since it can rely to a large degree on the well-formedness of the input text¹⁹. On the other hand, SG can be overly stingy on the strings it generates, for it has to guarantee only the well-formedness of its output, not that it will output all possible variations that express the same content. In contrast, working with a *bidirectional* grammar BG does put some pressure on the linguist, who is obliged to fine-tune his descriptions to match linguistic reality: if a text can be analysed by BG, it can also be generated by BG and conversely.

The requirement that a bidirectional grammar be observationally adequate is theoretically well motivated, but the constraints it imposes on linguistic descriptions may be too demanding in practice.

This is especially true in the case of some semi-productive phenomena, be it in syntax, or in derivational or compositional morphology, for which it is infeasible in practice to provide a complete description. Consider such phenomena as:

¹⁹ Of course, this doesn't come for free: underspecification of AG will eventually lead to spurious syntactic ambiguities, even when the input text can be relied on to be well-formed!

(a₁) Reagan supporter

(b₁) reaganite

(a'₁) Roosevelt supporter

(b'₁) *rooseveltite

(a₂) africain de l'ouest

(b₂) ouest-africain

(a'₂) américain de l'ouest

(b'₂) *ouest-américain

(a₃) portrait by Goya of the
duchess of Alba

(b₃) portrait of the duchess
of Alba by Goya

(a'₃) portrait of Saskia by
the greatest Dutch painter

(b'₃) *portrait by the
greatest Dutch painter of
Saskia

In each of the examples 1,2 and 3 we have assumed that:

- a₁ and b₁ share the same semantics (idem for a'₁ and b'₁);
- the a-type rendering of the semantics is fully productive, but the b-type rendering is semi-productive: thus a₁ and a'₁ are both possible, but b₁ and b'₁ have different acceptability status²⁰;
- a complete description of the conditions under which the b-type rendering is acceptable is difficult to obtain in practice.

Under these conditions, a good strategy would be:

- in analysis, to admit both a-type and b-type renderings; this would result in an *overgenerating* analysis component, but this would be compensated by the fact that ill-formed input would be rare;

²⁰ In example (3), it is assumed that the a-type rule for a *np* having two argumental complements is: order the complements according to their length with shorter first, while the b-type rule does not impose such an order constraint. For the sake of the argument, an assumption is made here that a more complete description of these phenomena eludes current linguistic knowledge; this is probably not really the case here.

- in generation, to discard b-type renderings; this would result in an *undergenerating* generation component, a reasonable price to pay in order to guarantee a higher level of grammaticality.

Such an approach would thus require a relaxation of the condition that the parsing grammar and the generation grammar (and the corresponding lexical components) be declaratively equivalent. Instead, one would require that the generation grammar "imply" the analysis grammar, in the sense that the set of linguistic structures described by the synthesis grammar would be a subset of those described by the analysis grammar. One way to achieve this might be to use one and the same bidirectional grammar, as done here, while allowing to flag a chosen goal *gc* in the body of a rule with a new annotation *generation_constraint*: this would have the effect of imposing the verification of this goal *only* in generation mode²¹.

Extrapolation. For expository reasons, the previous discussion has been limited to the case of DCG's, although in fact we use an extension of DCG's known as Extrapolation Grammars (XG), which permit the handling of an interesting class of *unbounded dependency* phenomena (Pereira 1981).

XGs pose a serious efficiency problem when used in *generation*: the order of calls in generation need not respect the left to right order of constituents; this has as a consequence that at the time of calling a nonterminal *nt*, the input and output extrapolations lists of this non constituent are *both* unbound, and that generation has no way of "knowing" whether the constituent should be generated as a string or extrapolated to the left. The consequences for processing can be disastrous.

²¹ An extreme case of the use of such constraints would be to call the goal 'fail' only in generation mode, thus resulting in the invalidation of the grammar rule containing it. A generalization of the idea of *generation_constraint* annotation would be to impose *acceptability level* annotations on the different goals of a program clause.

We have developed some techniques to make XG's reasonably efficient in generation. They involve "marking" the semantics of extraposed phrases in such a way that in generation, the decision to produce the phrase in the string or to extrapose it to the left can be made univocally by looking at the semantics of the phrase for the presence of this mark. In this way, we have been able to handle several types of relative clauses, including cases of *pied piping* ("the owner of which");

Although we suspect that these techniques could be made general, we have not as yet come up with a satisfying theoretical rationalization of these practices. This is an area where more research is needed.

8. Related work

Approaches to grammar reversibility can be classified roughly according to two dimensions²²:

full vs. *partial* reversibility: are the parsing and generation modes two computational realizations of the same abstract relation or not?

automatic vs. *manual* reversibility: are the parsing and generation program derived automatically (either by mode-sensitive compilation

²² A remark: In the abstract, any linguistic theory postulating some sort of semantic level should be a choice prey for a full and at least manual reversible implementation. In practice, however, this would suppose: 1) that a stable "reference grammar" exist for the theory, that it be completely specific on admitted constructions (ie that it be a *formal grammar*) and that it be neutral between its parsing and generation modes; 2) that a reasonably efficient *full parsing implementation* of the grammar be attainable; 3) that a reasonably efficient *full generation implementation* of the grammar be attainable.

In practice, the history of the subject bears witness that these three conditions rarely obtain; Most often, computational linguists seem to aim at partial implementations of the formal grammar, using the freedom afforded by this partialness to make the implementation efficient in the chosen mode - either parsing or generation; or (which is worse) they bias the formal grammar specification towards one of the modes, obliterating the distinction between formal specification and program, and making the implementation of the other mode that much more difficult.

or by mode-sensitive interpretation) from a common specification, or are they derived manually from this specification?

Without any claim to completeness, the following are some of the approaches that have been discussed in the literature:

PHRED (Jacobs 1985), Ariane (Vauquois & Chappuy 1985), and Rosetta (Landsbergen 1987) can all be considered - in different degrees - as instances of the "non-automatic" approach to reversibility.

PHRED. PHRED and PHRAN (Wilensky & Arens 1980) are respectively the generation and the analysis components of a natural language interface to the Unix system: PHRAN analyzes the user's questions about system operation, and PHRED generates the answers.

PHRED was developed after PHRAN, and while they implement a form of reversibility, by sharing a "common linguistic knowledge base", they do not seem to implement full reversibility in the above sense.

Ariane and Rosetta are machine translation systems which incorporate some degree of reversibility.

Ariane. In the Ariane approach, the first phase in the implementation of analysis and generation programs for a language (or sublanguage) consists in the writing of a declarative linguistic specification, a so-called "static grammar" using a special-purpose formalism. This static grammar then provides the basis for the manual derivation of two so-called "dynamic grammars" for analysis and generation, which are both implemented in the NL programming language ROBRA (GETA's²³ tree transducer formalism) (Boitet & Nedobejkine 1981). Although the two dynamic grammars are supposed to closely reflect the static grammar specification, it is not completely clear to what extent this can be strictly guaranteed.

²³ GETA: Groupe d'Etudes pour la Traduction Automatique, Grenoble.

Rosetta. In the Rosetta approach, a semantic structure takes the form of a "semantic derivation tree", with nodes labeled by "semantic rules". The semantic derivation tree is used as the basis for building surface syntactic structures (called S-trees), which is done by means of a "compositional function"²⁴. It is the responsibility of the linguist to make sure that this compositional function respects the "reversibility condition", ie the condition that there exists an "analytical function", which is the reverse of the compositional function, and which has to be defined "in tandem" with the compositional function.

Under these conditions, it is then possible either to start from a semantic structure and obtain an S-tree for the sentence (generation mode), or conversely to start with an S-tree and obtain a semantic structure (analysis mode), in a guaranteed reversible way.

One possible problem with the approach, however, may be the fact that it is somewhat asymmetrical between generation and analysis. Generation is straightforward: starting from the semantic structure, a S-tree is obtained, and from this S-tree, the sentence string is produced from a simple enumeration of this S-tree's leaves. On the other hand, in the analysis mode, it would be hopelessly inefficient, starting with a sentence string, to blindly hypothesize an S-tree without any way to check its syntactic well-formedness, and then try and see if the analytical functions can apply to it to produce some semantic structure. This is why one has to supplement the system with a "surface syntax component", which strongly constrains the S-trees which can be hypothesized from a sentence string. This component is therefore redundant with the other components of the system, and this fact may mean that it will be difficult in practice to make sure that the surface syntax component does not add constraints to the analysis mode which are not in effect in the generation mode, thereby undermining full system reversibility.

²⁴ This "compositional syntax" approach can be considered as a kind of dual to the more usual "compositional semantics" approach.

Instances of the automatic approach to reversibility are CRITTER (discussed in the present paper), the approach discussed in (Shieber 1988) and the BUG model of (Van Noord 1989)²⁵.

These three approaches have in common: i) that they aim at full reversibility - either logic grammars or unification grammars - and ii) that they use formalisms relying on unification.

Shieber 1988. The approach of (Shieber 1988) differs from our own by using Earley deduction both for parsing and generation. In generation mode, a "semantic filter" is used to constrain lemmas which can be added to the agenda. For this semantic filter to work, the grammar is presupposed to respect a "semantic monotonicity criterion", namely that the semantics of a subphrase SP of the phrase P be "subsumed by a portion of the semantics of P". Under this condition, a lemma 'A <-- B₁...B_n' can only be part of a successful proof of the main S goal when the (partially instantiated) semantics of A subsumes a part of the semantics of S (which of course is assumed to be given as the input to the generation process). This is the property which is checked by the semantic filter, allowing only "promising" lemmas to be added to the agenda.

As the author acknowledges, semantic monotonicity may be too strong a condition on realistic grammars and "finding a weaker constraint is an important research objective". By contrast, in a backward deduction approach, such a constraint is not necessary for generation to work (see section 7). On the other hand, parsing with a backward deduction approach has inconveniencies not displayed by a lemma (or forward) approach, most conspicuously the problem of handling left-recursion in a general way (compare with section 5).

BUG. The BUG model (Van Noord 1989) takes a forward deduction approach (comparable to the left-corner parsing algorithm of [Pereira & Shieber 1987]) to the problem of generation (the paper is not concerned with

²⁵ Since the writing of this chapter, a paper by Shieber and al. (1989) has appeared, which elaborates on both these approaches.

parsing). It makes use of a semantic filter using a precompiled semantic reachability relation 'link', relating the semantics of a node to the semantics of its "semantic head" (and transitively of the semantic head of this semantic head, etc...). This relation allows deduced facts whose semantics is not "linked" to the (previously known) semantics of their maximal projection to be filtered out, thereby permitting strong predictions to be made on the lexical head of a phrase. The grammar is supposed to obey the "semantic head condition", which is somewhat similar to conditions 2) and 3) of section 6, and which does not imply semantic monotonicity, thus allowing certain cases of idiom to be handled.

9. Conclusions

We have described a practical solution to the computational problem of *grammar bidirectionality*. Grammars are formalized as logical theories, and both parsing and generation are seen as special cases of backward theorem proving. The deductive process is made efficient in both modes through the use of *task-specific control annotations*.

Grammar bidirectionality provides some important theoretical and practical benefits:

- On the *theoretical* side, it seems reasonable to assume that a *common grammatical competence* underlies both parsing and generation. Bidirectionality tends to impose virtue upon the linguist whose job is to specify this common competence. Unidirectional *parsers* are usually based on grammars that, in the abstract sense, grossly *overgenerate*. This is because the grammar writer assumes, sometimes detrimentally, that the input will not contain ill-formed sentences. Conversely, unidirectional *generators* are usually based on grammars that grossly *undergenerate*. The grammar writer arbitrarily selects a restricted NL subset. Now, there is no way for one and the same grammar to both overgenerate and undergenerate: a bidirectional grammar has to generate the *right* set of sentences.

- On the practical side, *bidirectionality* helps make the grammar more accurate. Without it, it is extremely difficult to figure out in what ways a unidirectional parser is overly permissive: one has to test a great many ungrammatical sentences for failure, an extremely unnatural endeavor. It is equally difficult to test a unidirectional generator by itself: one has to type in complicated semantic structures. But when from the result of a parse one can generate all the paraphrases admitted by the grammar, the flaws immediately become obvious. Thus a bidirectional grammar is in a sense *self-debugging*.

We emphasize that the techniques presented in this paper are of practical significance. They have been extensively tested, with positive results. Specifically, our control annotation scheme has permitted the development of fairly large-scale reversible grammars for French and English. These grammars are used in the CRITTER system (Isabelle et al. 1988), an experimental machine translation system that translates agricultural market reports in a fully reversible manner between English and French. The system is still incomplete, but the coverage of the grammars is already broad: control and raising verbs, passivization, dative movement, "there" insertion, some types of relatives (reduced and unreduced)²⁶, some types of coordination, various types of modification, etc. The dictionaries contain about 1000 lexical entries each. CRITTER is implemented in Quintus Prolog. Typical translation times are well below 1 second per word on a SUN 3/60. And, for typical sentences, parsing and generation times are about the same.

Acknowledgments.

Thanks to Guy Lapalme, Elliot Macklovitch and Stan Szpakowicz for suggestions and comments and to Jean-Luc Cochard, François Perreault and Michel Simard for their contribution to the work reported here.

²⁶ See the discussion of extraposition in section 7.

References.

- Appelt, Douglas E. 1987. Bidirectional Grammars and the Design of Natural Language Generation Systems. In *Theoretical Issues in Natural Language Processing 3 (TINLAP-3)*, 206-12. Las Cruces, NM: Association for Computational Linguistics, January.
- Beaven, John L. and Whitelock, Peter. 1988. Machine Translation using Isomorphic UCGs. In *Proceedings of the 12th International Conference on Computational Linguistics*, 32-35. Budapest, August.
- Boitet, Christian, and Nedobekine, Nicolas. 1981. Recent developments in Russian-French machine translation at Grenoble. *Linguistics* 19 (3/4): 199-271.
- Colmerauer, Alain. 1971. *Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur*. Montreal: Université de Montréal, Projet TAUM, Rapport de Recherche.
- Colmerauer, Alain. 1982. *PROLOG II : Manuel de référence et modèle théorique*. Marseilles, France: Faculté des Sciences de Luminy, Groupe d'Intelligence Artificielle.
- Dymetman, Marc and Isabelle, Pierre. 1988. Reversible Logic Grammars for Machine Translation. In *Proceedings of the Second International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*. Pittsburgh: Carnegie Mellon University, June.
- Earley, J. 1970. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM* 13:2 (February): 94-102.
- Hasida, K. and Isizaki, S.. 1987. Dependency propagation : a unified theory of sentence comprehension and generation. In *Proceedings of AAAI*, 664-670. Seattle, WA, July.
- Isabelle, Pierre, Dymetman, Marc and Macklovitch, Elliott. 1988. CRITTER: a Translation System for Agricultural Market Reports. In *Proceedings of the 12th International Conference on Computational Linguistics*, 261-266. Budapest, August.
- Jacobs, P. 1985. PHRED: a generator for natural language interfaces. *Computational Linguistics* 11:4 (October-December): 219-42.
- Kay, Martin. 1975. Syntactic Processing and Functional Sentence Perspective. In *Theoretical Issues in Natural Language Processing (TINLAP), Supplement to the Proceedings*, 12-15. Cambridge, MA: Association for Computational Linguistics, June.
- Kay, Martin. 1980. Algorithm Schemata and Data Structures in Syntactic Processing. In *Readings in Natural Language Processing*, eds. Grosz B., K. Sparck Jones and B. Lynn Webber (1986). Los Altos, CA: Morgan Kaufmann.
- Kowalski, R.A. 1979a. Algorithm = Logic + Control. *Communications of the ACM* 22 (July): 424-436.
- Kowalski, R.A. 1979b. *Logic for Problem Solving*. Reading, New York, NY: North-Holland.
- Landsbergen, J. 1987. *Montague Grammar and Machine Translation*. Eindhoven, Holland: Philips Research M.S. 14.026.
- Matsumoto Y., Tanaka, H., Hirikawa, H., Miyoshi, H. and Yasukawa, H. 1983. BUP: a bottom-up parser embedded in Prolog. *New Generation Computing* 1:2, 145-158.
- Pereira, Fernando C. N. 1981. Extraposition Grammars. *Computational Linguistics* 7:4, 243-56.
- Pereira, Fernando C. N. and Warren, David H. D. 1980. Definite Clause Grammars for Language Analysis. *Artificial Intelligence* 13, 231-78.
- Pereira, Fernando C. N. and David Warren, David H. D. 1983. Parsing as Deduction. In *Proceedings of the 21th Annual Meeting of the Association for Computational Linguistics*, 137-44. Cambridge, MA, June.
- Pereira, Fernando C. N. and Shieber, Stuart M. 1987. *Prolog and Natural Language Analysis*. CSLI lecture note No. 10. Stanford, CA: Center for the Study of Language and Information.
- Pollard, Carl and Ivan A. Sag. 1987. *Information-Based Syntax and Semantics*, vol. 1. CSLI lecture note No. 13. Stanford, CA: Center for the Study of Language and Information.
- Sells, Peter. 1985. *Lectures on Contemporary Linguistic Theories*. CSLI lecture note No. 3. Stanford, CA: Center for the Study of Language and Information.
- Shieber, Stuart M. 1986. A Uniform Architecture for Parsing and Generation. In *Proceedings of the 12th International Conference on Computational Linguistics*, 614-19. Budapest, August.
- Shieber, Stuart M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI lecture note No. 4. Stanford, CA: Center for the Study of Language and Information.
- Shieber, Stuart, M., van Noord, Gertjan, Moore, Robert and Pereira, Fernando. 1989. A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, 7-17. Vancouver, BC, Canada, June.
- Strzalkowski, Tomek. 1989. *Automated Inversion of a Unification Parser into a Unification Generator*. New York: Courant Institute of Mathematical Sciences, Department of Computer Science, Technical Report 465.

- Van Noord, Jan. 1989. *BUG: A Directed Bottom-up Generator for Unification Based Formalisms*. Working Papers in Natural Language Processing No. 4. Utrecht, Holland: RUU, Department of Linguistics.
- Vauquois, Bernard and Chappuy, Sylviane. 1985. Static Grammars. A Formalism for the Description of Linguistic Models. In *Proceedings of the International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, 298-322. Hamilton, NY: Colgate University, August.
- Wilensky, R. and Arens, Y. 1980. *PHRAN: a Knowledge-Based Approach to Natural Language Analysis*. Berkeley, CA: University of California, Electronics Research Laboratory Memorandum #UCB/ERL M80/34.


CACC / CCAC




95587

P
308
D965e
1990
c.2

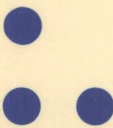
[illegible]



Pour plus de détails,
veuillez communiquer avec :



*Le Centre canadien de recherche
sur l'informatisation du travail*
1575, boulevard Chomedey
Laval (Québec)
H7V 2X2
(514) 682-3400



For more information,
please contact:

*Canadian Workplace
Automation Research Centre*
1575 Chomedey Blvd.
Laval, Quebec
H7V 2X2
(514) 682-3400