HOW TO INVERT A NATURAL LANGUAGE PARSER INTO AN EFFICIENT GENERATOR: 
AN ALGORITHM FOR LOGIC GRAMMARS

Tomek Strzalkowski
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012

ABSTRACT
The use of a single grammar in natural language parsing and generation is most desirable for variety of reasons including efficiency, perspicuity, integrity, robustness, and a certain amount of elegance. In this paper we present an algorithm for automated inversion of a PROLOG-coded unification parser into an efficient unification generator, using the collections of minimal sets of essential arguments (MSEA) for predicates. The algorithm is also applicable to more abstract systems for writing logic grammars, such as DCG.

INTRODUCTION
In this paper we describe the results obtained from the experiment with reversing a PROLOG parser for a substantial subset of English into an efficient generator. The starting point of the experiment was a string parser for English (Grishman, 1986), which is used in an English-Japanese MT project. The PROLOG version of this parser was inverted, using the method described here, into an efficient PROLOG generator working from regularized parse forms to English sentences. To obtain a PROLOG parser (or any PROLOG program) working in the reverse, requires some manipulation of the clauses, especially the ordering of the literals on their right-hand side, as noted by Dymetman and Isabelle (1988). We do not discuss here certain other transformations used to "normalize" the parser code in order to attain maximum efficiency of the derived generator program (Strzalkowski, 1989).

IN AND OUT ARGUMENTS
Arguments in a PROLOG literal can be marked as either "in" or "out" depending on whether they are bound at the time the literal is submitted for execution or after the computation is completed. For example, in

\[ \text{tovo}([\text{to}, \text{eat}, \text{fish}], T4, [\text{np}, [\text{n}, \text{john}]], P3) \]

the first and the third arguments are "in", while the remaining two are "out". When tovo is used for generation, i.e.,

\[ \text{tovo}(T1, T4, P1, [\text{eat}, [\text{np}, [\text{n}, \text{john}]], [\text{np}, [\text{n}, \text{fish}]]]) \]

then the last argument is "in", while the first and the third are "out"; T4 is neither "in" nor "out". The information about "in" and "out" status of arguments is important in determining the "direction" in which predicates containing them can be run. As a further example consider the literal

\[ \text{subject}(A1, A2, \text{WHQ}, \text{NUM}, P) \]

where A1 and A2 are input and output strings of words, \text{WHQ} indicates whether the subject phrase is a part of a clause within a wh-question, \text{NUM} is the number of the subject phrase, and P is the final translation. During parsing, the "in" arguments are A1 and \text{WHQ}, the "out" arguments are A2, NUM and P; during generation, the "in" arguments are P and WHQ, the "out" arguments are A1 and NUM. In generating, A2 is neither "in" nor "out". Thus, upon reversing the direction of computation, an "out" argument does not automatically become an "in" argument, nor does an "in" argument automatically become an "out" argument. Below is a method for computing "in" and "out" status of arguments in any given literal in a PROLOG program, as required by the inversion procedure. This algorithm is already general enough to handle any PROLOG program.

An argument X of literal \( \text{pred}(\cdots X \cdots) \) on the rhs of a clause is "in" if

(A) it is a constant; or
(B) it is a function and all its arguments are "in"; or
(C) it is "in" or immediately "out" in some previous literal \( \text{pred}_0 \) on the rhs of the same clause, i.e., \( \text{i}(Y) \leftarrow \text{pred}_0(X,Y), \text{pred}(X) \); or
(D) it is "out" in an rhs literal \( \text{pred}_0 \) delayed until after some predicate \( \text{pred}_1 \) such that \( \text{pred}_0 \) precedes

1 Barring the presence of non-reversible operators.

2 For more discussion on directed predicates in PROLOG see Shoham and McDermott (1984), and Debray (1989).
pred₁, and pred₁ precedes pred on the rhs;¹ or

(E) it is "in" in the head literal L on lhs of the same clause.

An argument X is "in" in the head literal L = pred(⋯X⋯) of a clause if (A), or (B), or
(F) L is the top-level literal and X is "in" in it (known a priori); or
(G) X occurs more than once in L and at least one of these occurrences is "in"; or
(H) for every literal L₁ = pred(⋯Y⋯) unifiable with L on the rhs of any clause with the head predicate pred₁ different than pred, and such that Y unifies with X, Y is "in" in L₁.

We distinguish two categories of "out" arguments in literals appearing on the right-hand side of a clause: immediate and delayed. An argument X occurring in literal pred(⋯X⋯) is immediately "out" if it is fully bound immediately after pred(⋯X⋯) is executed. An argument X in pred(⋯X⋯) is "out" delayed until after pred₁, if it is fully bound only after pred₁, following pred on rhs, is executed. For example, consider the following fragment:

vp(SN) :- agree(SN, VN), v(VN).
agree(N, M).

If VN is immediately "out" in v, then SN in agree is "out" delayed until after v. For arguments with their "out" status delayed until after pred₁, the "out" status is assigned only after pred₁ is executed.

An argument X of literal pred(⋯X⋯) on the rhs of a clause is immediately "out" if
(A) it is "in" in pred(⋯X⋯); or
(B) it is a functional expression and all its arguments are either "in" or immediately "out"; or
(C) for every clause with the head literal pred(⋯Y⋯) unifiable with pred(⋯X⋯) and such that Y unifies with X, Y is either "in", "out" or "unknown", and Y is marked "in" or "out" in at least one case.

An argument X of literal pred(⋯X⋯) on the rhs of a clause is "out" delayed until after pred₁(⋯Y⋯) following pred₁ if
(D) Y is immediately "out" in pred₁ and X=f(Y); or
(E) X is a functional expression and all of its arguments are either "in" or immediately "out" or "out" delayed until after pred₁; or

(F) there is a predicate pred₁(⋯Z⋯) preceding pred₁ on the rhs, where Z is a subset of arguments of pred₁ such that every argument in Z is "out" delayed until after pred₁ and whenever Z is "in" then X is immediately "out" in pred₁.

An argument X of literal pred(⋯X⋯) on the lhs of a clause is "out" if
(G) it is "in" in pred(⋯X⋯); or
(H) it is "out" (immediately or delayed) in literal pred₁(⋯X⋯) on the rhs of this clause, providing that pred₁ ≠ pred (again, we must take provisions to avoid infinite descend, cf. (H) in "in" algorithm); if pred₁ = pred then X is marked "unknown".

ESSENTIAL ARGUMENTS

Some arguments of every literal are essential in the sense that the literal cannot be executed successfully unless all of them are bound, at least partially, at the time of execution. A literal may have several alternative, possibly overlapping, sets of essential arguments. If all arguments in any one of such sets of essential arguments are bound, then the literal can be executed. Any set of essential arguments which have the above property is called essential. We shall call the set MSEA of essential arguments a minimal set of essential arguments if it is essential, and no proper subset of MSEA is essential. If we alter the ordering of the rhs literals in the definition of a predicate, we may also change its set of MSEA's. We call the set of MSEA's existing for a current definition of a predicate the set of active MSEA's for this predicate. To run a predicate in a certain direction requires that a specific MSEA is among the currently active MSEA's for this predicate, and if this is not already the case, then we have to alter the definition of this predicate so as to make this MSEA become active. As an example consider the following clause from our PROLOG parser:

objectbe(O₁, O₂, P₁, P₂, PSA, P) :-
  venpass(O₁, O₂, P₁, P₃),
  concat([P₂, P₃], PSA, P).

Assuming that {O₁} and {P₃} are MSEA's of venpass and that P₃ is "out" in venpass whenever O₁ is "in", we obtain that {O₁} is the only candidate for an active MSEA in objectbe. This is because P₃ is not present on the argument list of objectbe, and thus cannot receive a binding before the execution of venpass commences. Moving to the concat literal, we note that its first argument is partially bound since P₃ is "out" in venpass. This is enough for concat to execute, and we conclude that {O₁} is in fact the only essential argument in objectbe. If we reverse the order of venpass and concat, then {P} becomes the new active MSEA for objectbe, while {O₁} is no longer active. Given the binding to its third argument, concat returns bindings to the

² The precedence is with respect to the order of evaluation, which in PROLOG is left-to-right.

³ An argument is considered fully bound if it is a constant or it is bound by a constant; an argument is partially bound if it is, or is bound by, a term in which at least one variable is unbound.

348
first two, and thus it also binds P3, which is an essential argument in *venpass.* Below is the general procedure MSEAS for computing the active sets of essential arguments in the head literal of a clause as proposed in (Strzalkowski and Peng, 1990).

Let's consider the following abstract clause defining a predicate R1:

\[ R_1(X_1, \cdots, X_k) \leftarrow \] (R)

\[ \cdots \]

\[ Q_2(\cdots), \]

\[ Q_{i}(\cdots). \]

Suppose that, as defined by (R), R1 has the set \( MS_1 = \{ m_1, \cdots, m_s \} \) of active MSEA’s, and let \( MR_1 \supseteq MS_1 \) be the set of all MSEA for \( R_1 \) that can be obtained by permuting the order of literals on the right-hand side of (R). Let us assume further that \( R_1 \) occurs on rhs of some other clause, as shown below:

\[ P(X_1, \cdots, X_k) \leftarrow \] (P)

\[ \cdots \]

\[ R_1(X_1, \cdots, X_{1,k}), \]

\[ R_2(X_{2,1}, \cdots, X_{2,k}), \]

\[ \cdots \]

\[ R_i(X_{i,1}, \cdots, X_{i,k}). \]

We want to compute \( MS \), the set of active MSEA’s for \( P \), as defined by (P), where \( s \geq 1 \), assuming that we know the sets of active MSEA for each \( R_i \) on the rhs. In the following procedure, the expression \( VAR(T) \), where \( T \) is a set of terms, denotes the set of all variables occurring in the terms in \( T \).

**MSEA \((MS, MSEA, VP, i, OUT)\)**

1. Start with \( VP = VAR(\{X_1, \cdots, X_k\}) \), \( MSEA = \emptyset \), \( i = 1 \), and \( OUT = \emptyset \). When the computation is completed, \( MS \) is bound to the set of active MSEA’s for \( P \).
2. Let \( MR_1 \) be the set of active MSEA’s of \( R_1 \), and let \( MRU_1 \) be obtained from \( MR_1 \) by replacing all variables in each member of \( MR_1 \) by their corresponding actual arguments of \( R_1 \) on the rhs of (C1).
3. If \( R_i = P \) then for every \( m_{1,k} \in MRU_1 \) if every argument \( Y_i \in m_{1,k} \) is always unifiable with its corresponding argument \( X_i \) in \( P \) then remove \( m_{1,k} \) from \( MRU_1 \). For every set \( m_{1,k} = m_{1,k} \cup \{ X_{1,j} \} \), where \( X_{1,j} \) is an argument in \( R_1 \) such that it is not already in \( m_{1,k} \) and it is not always unifiable with its corresponding argument in \( P \), and \( m_{1,k} \) is not a superset of any other \( m_{1,k} \) remaining in \( MRU_1 \), add \( m_{1,k} \) to \( MRU_1 \).
4. For each \( m_{1,j} \in MRU_1 \) (\( j = 1 \cdots r_1 \)) compute \( \mu_{1,j} \) := \( VAR(m_{1,j}) \cap VP \). Let \( MP_1 = \{ \mu_{1,j} \mid \phi(\mu_{1,j}) \neq \emptyset \} \), where \( r > 0 \), and \( \phi(\mu_{1,j}) = \emptyset \) or \( \phi(\mu_{1,j}) = \emptyset \). Let \( MP_1 = \emptyset \) then QUIT: (C1) is ill-formed and cannot be executed.
5. For each \( \mu_{1,j} \in MP_1 \) we do the following: (a) assume that \( \mu_{1,j} \) is "in" in \( R_1 \); (b) compute set \( OUT_{i,j} \) of "out" arguments for \( R_1 \); (c) call MSEAS(\( MS_j, \mu_{1,j}, VP, 2, OUT_{i,j} \)); (d) assign MS := \( \bigcup_j MS_j \).
6. In some \( i \)-th step, where \( 1 \leq i \leq s \), and MSEA = \( \mu_{1,-1,k}, \mu_{1,-1,k} \), let’s suppose that \( MR \) and \( MRU \) are the sets of active MSEA’s and their instantiations with actual arguments of \( R_i \), for the literal \( R_i \) on the rhs of (P).
7. If \( R_i = P \) then for every \( m_{i,u} \in MRU_i \) if every argument \( Y_i \in m_{i,u} \) is always unifiable with its corresponding argument \( X_i \) in \( P \) then remove \( m_{i,u} \) from \( MRU_i \). For every set \( m_{i,u} = m_{i,u} \cup \{ X_{i,j} \} \), where \( X_{i,j} \) is an argument in \( R_i \) such that it is not already in \( m_{i,u} \) and it is not always unifiable with its corresponding argument in \( P \) and \( m_{i,u} \) is not a superset of any other \( m_{i,u} \) remaining in \( MRU_i \), add \( m_{i,u} \) to \( MRU_i \).
8. Again, we compute the set \( MP_i = \{ \mu_{i,j} \mid j = 1 \cdots r_i \} \), where \( \mu_{i,j} = (VAR(\mu_{i,j}) \cap OUT_{i-1,k}, \) where \( OUT_{i-1,k} \) is the set of all "out" arguments in literals \( R_{i-1} \) to \( R_{i-1} \).
9. For each \( \mu_{i,j} \) remaining in \( MP_i \) where \( i \leq s \) do the following:
   (a) if \( \mu_{i,j} = \emptyset \) then: (i) compute the set \( OUT_{i,j} \) of "out" arguments of \( R_i \); (ii) compute the union \( OUT_{i,j} := OUT_{i,j} \cup OUT_{i-1,k}, \) (iii) call MSEAS(\( MS_j, \mu_{i,j} \cap VP, i+1, OUT_{i,j} \));
   (b) otherwise, if \( \mu_{i,j} \neq \emptyset \) then find all distinct minimal size sets \( V_i \subseteq VP \) such that whenever the arguments in \( V_i \) are "in", then the arguments in \( \mu_{i,j} \) are "out". If such \( V_i \)’s exist, then for every \( V_i \) do: (i) assume \( V_i \) is "in" in \( P \); (ii) compute the set \( OUT_{i,j} := \) arguments in all literals from \( R_1 \) to \( R_i \); (iii) call MSEAS(\( MS_j, \mu_{i,j} \cap V_i \cap VP, i+1, OUT_{i,j} \));
   (c) otherwise, if no such \( V_i \) exist, \( MS_j := \emptyset \).
10. Compute \( MS := \bigcup_{j=1,r} MS_j \);
(11) For \( i=s+1 \) set \( MS := \{ MSEA \} \).

In order to compute the set of all \( MSEA \)'s for \( P \), the procedure presented above need to be modified so that it would consider all feasible orderings of literals on the rhs of \( (P) \), using information about all \( MSEA \)'s for \( R_i \)'s. This modified procedure would regard the rhs of \( (P) \) as an unordered set of literals, and use various heuristics to consider only selected orderings. We outline the modified procedure briefly below.

Let \( RR \) denote this set, that is, \( RR = \{ R_i \mid i=1 \cdots s \} \). We add \( RR \) as an extra argument to \( MSEA \) procedure, so that the call to the modified version becomes \( MSEA(MS, MSEA, VP, RR, i, OUT) \). Next we modify step (2) in the procedure as follows:

(2') For every element \( R_{i,j} \in RR \), do (2) to (5):

(2) Let \( MR_{i,j} \) be the set of all \( MSEA \)'s of \( R_{i,j} \), and let \( MRU_{i,j} \) be obtained from \( MR_{i,j} \) by replacing all variables in each member of \( MR_{i,j} \) by their corresponding actual arguments of \( R_{i,j} \).

Further steps are modified accordingly. The reader may note that the modified \( MSEA \) procedure will consider all feasible ways of ordering elements of \( RR \).

In the steps shown above, we select all literals as potential leading elements on the right hand side, even though most of them will be rejected by steps (3) and (4). For those that survive, we will select elements from the rest of \( RR \) that can follow them. In step (5) the recursive call to \( MSEA \) will be \( MSEA(MS_{i,j}, MRU_{i,j}, VP, RR-\{R_{i,j}\}, 2, OUT_{i,j}) \). In step (6), that is, in \( i \)-th step of the recursion, we consider all elements of \( RR-\{R_{i,j}\} \mid j=1 \cdots i-1 \), for selection of the \( i \)-th literal on the right-hand side. By this time we will have already generated a number of possible orderings of \( \{R_i \mid l=1 \cdots i-1 \} \). We add step (6') which contains the head of an iteration over the remaining elements of \( RR \), and covering steps (6) to (11). Again, some of the elements of \( RR \) will be rejected in steps (7) and (10). We continue until \( RR \) is completely ordered, possibly in several different ways. For each such possible ordering a set of \( MSEA \)'s will be computed. Step (12) is an end condition with \( RR=\emptyset \). To obtain a meaningful result, \( MSEA \)'s in \( MR_{i,j} \) must be grouped into sets of those which are active at the same time, that is, they belong to the set of active \( MSEA \)'s for a specific definition of \( P \) (i.e., ordering of \( RR \)). \( MSEA \)'s belonging to different groups give rise to alternative sets of \( MSEA \)'s in the final set \( MS \). Note that in this modified algorithm, \( MS \) becomes a set of sets of sets.

An important part in the process of computing essential arguments for literals is the selection of \( MSEA \)'s for lexicon access and other primitives whose definitions are not subject to change. As an example, consider a fragment of a lexicon:

\[
\begin{align*}
\verb(\text{looks}[V]), V, sg, \text{look}. \\
\verb(\text{looks}[V]), V, pl, \text{look}. \\
\text{verb}(\text{arrives}[V], V, sg, \text{arrive}). \\
\text{verb}(\text{arrives}[V], V, pl, \text{arrive}).
\end{align*}
\]

The lexicon access primitive \( \text{verb}(V_1, V_2, Nm, P) \) has two sets of essential arguments: \( \{V_1\} \) and \( \{Nm, P\} \). This is because \( \{V_1\} \) can be consistently unified with at most one of \( \{\text{looks}[V]\}, \{\text{look}[V]\}, \{\text{arrive}[V]\}, \) etc., at a time. Similarly, \( \{Nm, P\} \) can be consistently unified at any one time with at most one of \( \{sg, look\}, \{pl, look\}, \{sg, arrive\}, \) etc. Note that neither \( \{P\} \) nor \( \{Nm\} \) alone are sufficient, since they would unify with corresponding arguments in more than one clause. This indeterminacy, although not necessarily fatal, may lead to severe inefficiency if the generator has to make long backups before a number agreement is established between, say, a verb and its subject. On the other hand, if the representation from which we generate does not include information about the lexical number for constituents, we may have to accept \( \{P\} \) as the generation-mode \( MSEA \) for \( \text{verb} \), or else we risk that the grammar will not be reversed at all.

**REORDERING LITERALS IN CLAUSES**

When attempting to expand a literal on the rhs of any clause the following basic rule should be observed: never expand a literal before at least one its active \( MSEA \)'s is "in", which means that all arguments in at least one \( MSEA \) are bound. The following algorithm uses this simple principle to reorder rhs of parser clauses for reversed use in generation. This algorithm uses the information about "in" and "out" arguments for literals and sets of \( MSEA \)'s for predicates. If the "in" \( MSEA \) of a literal is not active then the rhs's of every definition of this predicate is recursively reordered so that the selected \( MSEA \) becomes active. We proceed top-down altering definitions of predicates of the literals to make their \( MSEA \)'s active as necessary, starting with the top level predicate parse(S,P), where \( P \) is marked "in" (parse structure) and \( S \) is marked "out" (generated sentence). We continue until we reach the level of atomic or non-reversible primitives such as concat, member, or dictionary look-up routines. If this process succeeds at reversing predicate definitions at each level, then the reversed-parser generator is obtained.

\[
\text{INVERSE("head :- old-rhs",ins,outs);} \\
[\text{ins and outs are subsets of VAR(head) which} \\
\text{are "in" and are required to be "out", respectively}. \\
\text{begin} \\
\text{compute M the set of all MSEA's for head;} \\
\text{for every MSEA m \in M do} \\
\text{begin} \\
\text{OUT} := \emptyset; \\
\text{if m is an active MSEA such that m \sqsubset ins then} \\
\text{begin} \\
\text{compute "out" arguments in head;} \\
\text{add them to OUT;} \\
\text{end} \\
\text{end} \\
\text{end} \\
\text{end} \\
\text{end}
\]
if outscOUT then DONE("head:-old-rhs")
end
else if m is a non-active MSEA and m|ins then
begin
new-rhs := Ø; QUIT := false;
old-rhs-1 := old-rhs;
for every literal L do M_L := ∅;
{done only once during the inversion}
repeat
mark "in" old-rhs-1 arguments which are
either constants, or marked "in" in head,
or marked "in", or "out" in new-rhs;
select a literal L in old-rhs-1 which has
an "in" MSEA m_L and if m_L is not active in L
then either M_L = ∅ or m_L e M_L;
set up a backtracking point containing
all the remaining alternatives
to select L from old-rhs-1;
if L exists then
begin
if m_L is non-active in L then
begin
if M_L = ∅ then M_L := M_L u {m_L};
for every clause "L1 :- rhs_{L1}" such that
L1 has the same predicate as L do
begin
INVERSE("L1 :- rhs_{L1}",M_L,∅);
if GIVEUP returned then backup, undoing
all changes, to the latest backtracking
point and select another alternative
end
compute "in" and "out" arguments in L;
add "out" arguments to OUT;
new-rhs := APPEND-AT-THE-END(new-rhs,L);
old-rhs-1 := REMOVE(old-rhs-1,L)
end {if}
else begin
backup, undoing all changes, to the latest
backtracking point and select another
alternative;
if no such backtracking point exists then
QUIT := true
end {else}
end
until old-rhs-1 = ∅ or QUIT;
if outscOUT and not QUIT then
DONE("head:-new-rhs")
end {elseif}
end; {for}
GIVEUP("grammar can't be inverted as specified")
end;

MOVING LITERALS BETWEEN CLAUSES

The inversion algorithm, as realized by the procedure INVERSE, requires that for each clause in the parser code we can find a definite order of literals on its right-hand side that would satisfy the requirements of running this clause in the reverse: appropriate minimal sets of essential arguments (MSEA’s) are bound at the right time. However, this requirement is by no means guaranteed and INVERSE may encounter clauses for which no ordering of the literals on the right-hand side would be possible. It may happen, of course, that the clause itself is ill-formed but this is not the only situation. It may be that two or more literals on the right-hand side of a clause cannot be scheduled because each is waiting for the other to deliver the missing bindings to some essential arguments. As an example, consider the grammar fragment below:

sent(P) :- sub(N1, P1),
          vp(N1, P1, P).
vp(N1, P1, P) :- v(N2, P2),
              agree(N1, N2),
              obj(P1, P2, P).

In the generation mode, that is, with the variable P instantiated by the parse structure of a sentence, the following active MSEA’s and "out" arguments have been computed:

<table>
<thead>
<tr>
<th>predicate</th>
<th>MSEA</th>
<th>&quot;out&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>sent</td>
<td>{P}</td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>{P1}</td>
<td>N1</td>
</tr>
<tr>
<td>vp</td>
<td>{P2}</td>
<td>N2</td>
</tr>
<tr>
<td>v</td>
<td>{P}</td>
<td></td>
</tr>
<tr>
<td>agree</td>
<td>{N1, N2}</td>
<td></td>
</tr>
<tr>
<td>obj</td>
<td>{P}</td>
<td>P1, P2</td>
</tr>
</tbody>
</table>

In order to use these rules for generation, we would have to change the order of literals on the right-hand side of sent clause, so that the vp is expanded first. However, doing so would require that the variable N1 is bound. This we could get by firing sub first, but we can’t do this either, since we wouldn’t know the binding to P1. We note, however, that if we consider the two clauses together, then a consistent ordering of literals can be found. To see it, we expand vp on the right-hand side of the first clause replacing it with the appropriately unified literals in the right-hand side of the second clause, and obtain a single new clause that can be reordered for generation as follows:

sent(P) :- obj(P1, P2, P),
          v(N2, P2),
          sub(N1, P1),
          agree(N1, N2).

Now we can reintroduce the non-terminal vp, and break the above rule back into two. Note that as a result agree migrated to the first clause, and N2 replaced N1 on the argument list of vp. Note also that N2 is not an essential argument in the new vp.

sent(P) :- vp(N2, P1, P),
          sub(N1, P1),
          agree(N1, N2).
vp(N2, P1, P) :- obj(P1, P2, P),
              v(N2, P2).
The only thing that remains to be done is to automatically determine the arguments of the new vp predicate. Doubtless, it will be a subset of the arguments occurring in the literals that create the right-hand side of the new clause. In the example given this set is \{n2, p1, p2, p\}. From this set, we remove all those arguments which do not occur in other literals of the original clause, that is, before the break up. The only such argument is p2, and thus the final set of arguments to vp becomes \{n2, p1, p\}, as shown above. The complete algorithm for interclausal reordering of goals can be described by a largely straightforward extension to INVERSE (Strzalkowski, 1989)\(^8\)

CONCLUSIONS

In this paper we presented an algorithm for automatic inversion of a unification parser for natural language into an efficient unification generator. The inverted program of the generator is obtained by an off-line compilation process which directly manipulates the PROLOG code of the parser program. We distinguish two logical stages of this transformation: computing the minimal sets of essential arguments (MSEA's) for predicates, and generating the inverted program code with INVERSE. We have completed a first implementation of the system and used it to derive both a parser and a generator from a single DCG grammar for English (Strzalkowski and Peng, 1990).

This method is contrasted with the approaches that seek to define a generalized but computationally expensive evaluation strategy for running a grammar in either direction without a need to manipulate its rules (Shieber, 1988), (Shieber et al., 1989), and see also (Colmerauer, 1982) and (Naish, 1986) for some relevant techniques, employing the trick known as goal freezing. To reduce the cost of the goal freezing, and also to circumvent some of its deficiencies, Shieber et al. (1989) introduced a novel top-down/bottom-up goal expansion strategy, in which only selected goals are expanded during the top-down phase of the interpreter. This technique, still substantially more expensive than a fixed-order top-down interpreter, does not by itself guarantee that the underlying grammar formalism can be used bidirectionally, and it may need to be augmented by static goal reordering, as described in this paper.

ACKNOWLEDGMENTS

Ralph Grishman, Ping Peng and other members of the Natural Language Discussion Group provided valuable comments to earlier versions of this paper.

---

\(^8\) It should be noted that recursive clauses are never used for literal expansion during interclausal ordering, and that literals are not moved to or from recursive clauses, although argument lists of recursive literals may be affected by literals being moved elsewhere.

REFERENCES


