COMPARING TWO GRAMMAR-BASED GENERATION ALGORITHMS: A CASE STUDY

Miroslav Martinovic and Tomek Strzalkowski

Courant Institute of Mathematical Sciences
New York University
715 Broadway, rm. 704
New York, N.Y., 10003

ABSTRACT

In this paper we compare two grammar-based generation algorithms: the Semantic-Head-Driven Generation Algorithm (SHDGA), and the Essential Arguments Algorithm (EAA). Both algorithms have successfully addressed several outstanding problems in grammar-based generation, including dealing with non-monotonic compositionality of representation, left-recursion, deadlock-prone rules, and nondeterminism. We concentrate here on the comparison of selected properties: generality, efficiency, and determinism. We show that EAA's traversals of the analysis tree for a given language construct, include also the one taken on by SHDGA. We also demonstrate specific and common situations in which SHDGA will invariably run into serious inefficiency and nondeterminism, and which EAA will handle in an efficient and deterministic manner. We also point out that only EAA allows to treat the underlying grammar in a truly multi-directional manner.

1. INTRODUCTION

Recently, two important new algorithms have been published ([SNMP89], [SNMP90], [S90a], [S90b] and [S91]) that address the problem of automated generation of natural language expressions from a structured representation of meaning. Both algorithms follow the same general principle: given a grammar, and a structured representation of meaning, produce one or more corresponding surface strings, and do so with a minimal possible effort. In this paper we limit our analysis of the two algorithms to unification-based formalisms.

The first algorithm, which we call here the Semantic-Head-Driven Generation Algorithm (SHDGA), uses information about semantic heads[1] in grammar rules to obtain the best possible traversal of the generation tree, using a mixed top-down/bottom-up strategy.

[1] The semantic head of a rule is the literal on the right-hand side that shares the semantics with the literal on the left.

The second algorithm, which we call the Essential Arguments Algorithm (EAA), rearranges grammar productions at compile time in such a way that a simple top-down left-to-right evaluation will follow an optimal path.

Both algorithms have resolved several outstanding problems in dealing with natural language grammars, including handling of left recursive rules, non-monotonic compositionality of representation, and deadlock-prone rules[2]. In this paper we attempt to compare these two algorithms along their generality and efficiency lines. Throughout this paper we follow the notation used in [SNMP90].

2. MAIN CHARACTERISTICS OF SHDGA'S AND EAA'S TRAVERSALS

SHDGA traverses the derivation tree in the semantic-head-first fashion. Starting from the goal predicate node (called the *root*), containing a structured representation (semantics) from which to generate, it selects a production whose left-hand side semantics unifies with the semantics of the root. If the selected production passes the semantics unchanged from the left to some nonterminal on the right (the so-called *chain rule*), this later nonterminal becomes the new root and the algorithm is applied recursively. On the other hand, if no right-hand side literal has the same semantics as the root (the so called *non-chain rule*), the production is expanded, and the algorithm is recursively applied to every literal on its right-hand side. When the evaluation of a non-chain rule is completed, SHDGA connects its left-hand side literal (called the *pivot*) to the initial root using (in a backward manner) a series of appropriate chain rules. At this time, all remaining literals in the chain rules are expanded in a fixed order (left-to-right).

[2] Deadlock-prone rules are rules in which the order of the expansion of right-hand side literals cannot be determined locally (i.e. using only information available in this rule).

Since SHDGA traverses the derivation tree in the fashion described above, this traversal is neither top-down (TD), nor bottom-up (BU), nor left-to-right (LR) globally, with respect to the entire tree. However, it is LR locally, when the siblings of the semantic head literal are selected for expansion on the right-hand side of a chain rule, or when a non-chain rule is evaluated. In fact the overall traversal strategy combines both the TD mode (non-chain rule application) and the BU mode (backward application of chain rules).

EAA takes a unification grammar (usually Prolog-coded) and normalizes it by rewriting certain left recursive rules and altering the order of right-hand side nonterminals in other rules. It reorders literals in the original grammar (both locally within each rule, and globally between different rules) in such a way that the optimal traversal order is achieved for a given evaluation strategy (eg. top-down left-to-right). This restructuring is done at compile time, so in effect a new executable grammar is produced. The resulting parser or generator is TD but not LR with respect to the original grammar, however, the new grammar is evaluated TD and LR (i.e., using a standard Prolog interpreter). As a part of the node reordering process EAA calculates the *minimal sets of essential arguments* (*msea's*) for all literals in the grammar, which in turn will allow to project an optimal evaluation order. The optimal evaluation order is achieved by expanding only those literals which are *ready* at any given moment, i.e., those that have at least one of their mseas instantiated. The following example illustrates the traversal strategies of both algorithms. The grammar is taken from [SNMP90], and *normalized* to remove deadlock-prone rules in order to simplify the exposition.[3]

```
(0)   sentence/decl(S) --> s(finite)/S.
(1)   sentence/imp(S) --> vp(nonfinite,[np(_)/you])
                                  /S.
   .......
(2)   s(Form)/S --> Subj, vp(Form,[Subj/S.
   .......
(3)   vp(Form,Subcat)/S --> v(Form,Z)/S,
                             vp1(Form,Z)/Subcat.
(4)   vp1(Form,[Compl|Z])/Ar --> vp1(Form,Z)/Ar,
                                  Compl.
(5)   vp1(Form,Ar)/Ar.
   .......
(6)   vp(Form,[Subj])/S --> v(Form,[Subj])/VP,
                             aux(Form,[Subj],VP)/S.
   .......
(7)   aux(Form,[Subj],S)/S.
(8)   aux(Form,[Subj],A)/Z --> adv(A)/B,
                                aux(Form[Subj],B)/Z.
   .......
(9)   v(finite,[np(_)/O,np(3-sing)/S])/love(S,O) -->
                                           [loves].
(10)  v(finite,[np(_)/O,p/up,np(3-sing)/S])/
                             call_up(S,O) --> [calls].
(11)  v(finite,[np(3-sing)/S])/leave(S) --> [leaves].
   .......
(12)  np(3-sing)/john --> [john].
(13)  np(3-pl)/friends --> [friends].
   .......
(14)  adv(VP)/often(VP) --> [often].
```

The analysis tree for both algorithms is presented on the next page. (Figure 1.). The input semantics is given as *decl(call_up(john,friends)).* The output string becomes *john calls up friends*. The difference lists for each step are also provided. They are separated from the rest of the predicate by the symbol |. The different orders in which the two algorithms expand the branches of the derivation tree and generate the terminal nodes are marked, in italics for SHDGA, and in roman case for EAA. The rules that were applied at each level are also given.

If EAA is rerun for alternative solutions, it will produce the same output string, but the order in which nodes *vp1(finite,[p/up,np(3-sing)/john])/[Subj]/S1_S2*, and *np(_)/friends/S2_[]* (level 4), and also, *vp1(finite,[np(3-sing)/john])/[Subj]/S1_S12*, and *p/up/S12_S2*, at the level below, are visited, will be reversed. This happens because both literals in both pairs are *ready* for the expansion at the moment when the selection is to be made. Note that the traversal made by SHDGA and the first traversal taken by EAA actually generate the terminal nodes in the same order. This property is formally defined below.

Definition. Two traversals T' and T'' of a tree T are said to be the *same-to-a-subtree* (*stas*), if the following claim holds: Let N be any node of the tree T, and $S_1,\ldots,S_n$ all subtrees rooted at N. If the order in which the subtrees will be taken on for the traversal by T' is $S_i^1,\ldots,S_i^n$ and by T'' $S_j^1,\ldots,S_j^n$, then $S_i^1=S_j^1,\ldots,S_i^n=S_j^n$. ($S_k^l$ is one of the subtrees rooted at N, for any k, and l)

Stas however does not imply that the order in which the nodes are visited will necessarily be the same.

[3] EAA eliminates such rules using global node reordering ([S91]).

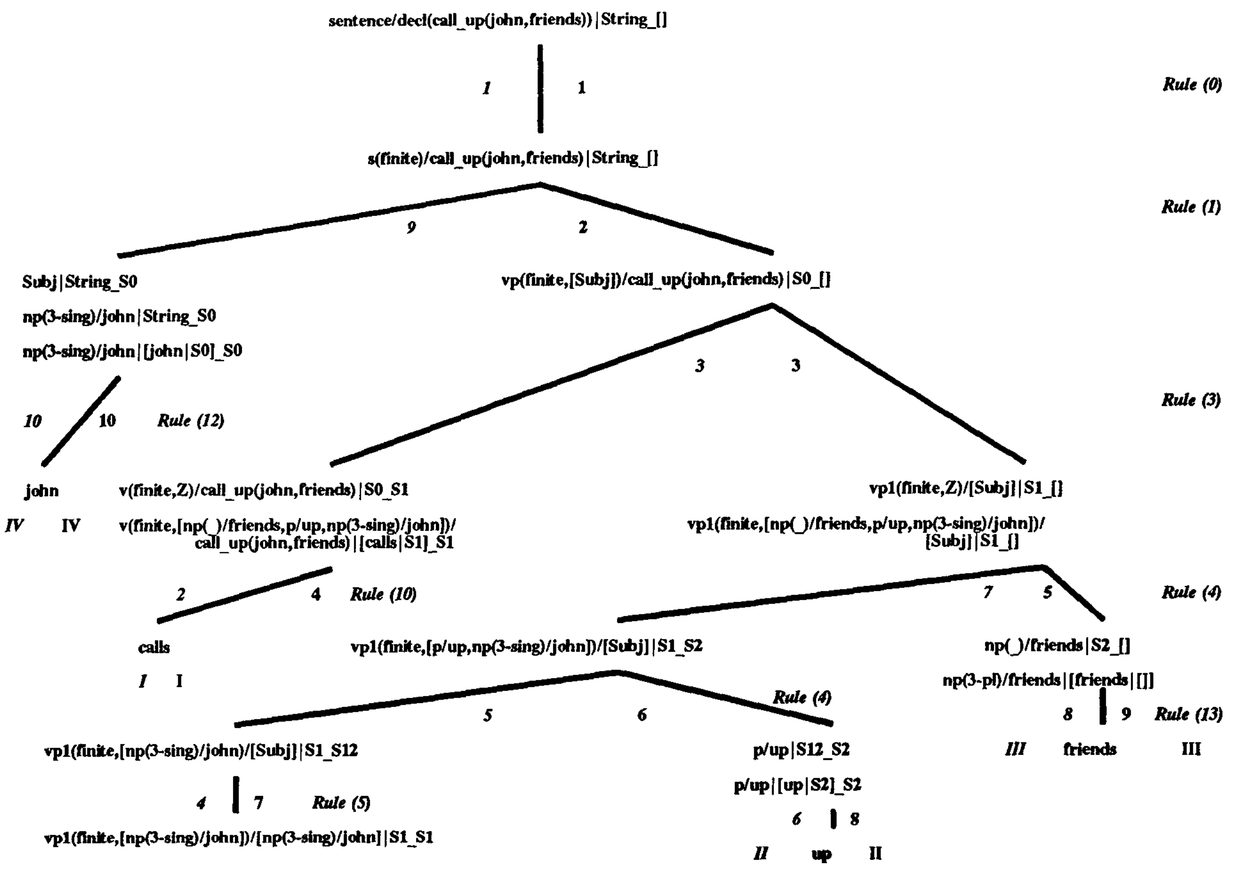
sentence/decl(call_up(john,friends)) | String_[]
1
1
Rule (0)
s(finite)/call_up(john,friends) | String_[]
9
2
Rule (1)
Subj | String_S0
np(3-sing)/john | String_S0
np(3-sing)/john | [john | S0]_S0
vp(finite,[Subj])/call_up(john,friends) | S0_[]
3
3
Rule (3)
10
10
Rule (12)
john
IV
IV
v(finite,Z)/call_up(john,friends) | S0_S1
v(finite,[np(_)/friends,p/up,np(3-sing)/john])/call_up(john,friends) | [calls | S1]_S1
vp1(finite,Z)/[Subj] | S1_[]
vp1(finite,[np(_)/friends,p/up,np(3-sing)/john])/[Subj] | S1_[]
2
4
Rule (10)
7
5
Rule (4)
calls
I
I
vp1(finite,[p/up,np(3-sing)/john])/[Subj] | S1_S2
np(_)/friends | S2_[]
np(3-pl)/friends | [friends | []]
5
6
Rule (4)
8
9
Rule (13)
III
friends
III
vp1(finite,[np(3-sing)/john)/[Subj] | S1_S12
p/up | S12_S2
p/up | [up | S2]_S2
4
7
Rule (5)
6
8
vp1(finite,[np(3-sing)/john])/[np(3-sing)/john] | S1_S1
II
up
II

FIGURE 1: EAA's and SHDGA's Traversals of An Analysis Tree.

3. GENERALITY-WISE SUPERIORITY OF EAA OVER SHDGA

The traversals by SHDGA and EAA as marked on the graph are stas. This means that the order in which the terminals were produced (the leaves were visited) is the same (in this case: *calls up friends john*). As noted previously, EAA can make other traversals to produce the same output string, and the order in which the terminals are generated will be different in each case. (This should not be confused with the order of the terminals in the output string, which is always the same). The orders in which terminals are generated during alternative EAA traversals are: *up calls friends john, friends calls up john, friends up calls john.* In general, EAA can be forced to make a traversal corresponding to any permutation of *ready* literals in the right-hand side of a rule.

We should notice that in the above example SHDGA happened to make all the right moves, i.e., it always expanded a literal whose msea happened to be instantiated. As we will see in the following sections, this will not always be the case for SHDGA and will become a source of serious efficiency problems. On the other hand, whenever SHDGA indeed follows an optimal traversal, EAA will have a traversal that is same-to-a-subtree with it.

The previous discussion can be summarized by the next theorem.

Theorem: If the SHDGA, at each particular step during its implicit traversal of the analysis tree, visits only the vertices representing literals that have at least one of their sets of essential arguments instantiated at the moment of the visit, then the traversal taken by the SHDGA is the *same-to-a-subtree* (*stas*) as one of the traversals taken by EAA.

The claim of the theorem is an immediate consequence of two facts. The first is that the EAA always selects for the expansion one of the literals with a msea currently instantiated. The other is the definition of traversals being same-to-a-subtree (always choosing the same subtree for the next traversal).

The following simple extract from a grammar, defining a *wh-question*, illustrates the forementioned (see Figure 2. below):

```
..........
(1)  whques/WhSem --> whsubj(Num)/WhSubj,
         whpred(Num,Tense,[WhSubj,WhObj])
                         /WhSem, whobj/WhObj.
..........
..........
(2)  whsubj(_)/who --> [who].
(3)  whsubj(_)/what --> [what].
..........
(4)  whpred(sing,perf,[Subj,Obj])/wrote(Subj,Obj)
                   --> [wrote].
..........
(5)  whobj/this --> [this].
..........
```

The input semantics for this example is *wrote(who,this)*, and the output string *who wrote this*. The numbering for the edges taken by the SHDGA is given in italics, and for the EAA in roman case. Both algorithms expand the middle subtree first, then the left, and finally the right one.

Each of the three subtrees has only one path, therefore the choices of their subtrees are unique, and therefore both algorithms agree on that, too. However, the way they actually traverse these subtrees is different. For example, the middle subtree is traversed bottom-up by SHDGA and top-down by EAA. *whpred* is expanded first by SHDGA (because it shares the semantics with the root, and there is an applicable non-chain rule), and also by EAA (because it is the only literal on the right-hand side of the rule (1) that has one of its msea's instantiated (its semantics)).

After the middle subtree is completely expanded, both sibling literals for the *whpred* have their semantics instantiated and thus they are both ready for expansion. We must note that SHDGA will always select the leftmost literal (in this case, *whsubj*), whether it is ready or not. EAA will select the same in the first pass, but it will expand *whobj* first, and then *whsubj*, if we force a second pass. In the first pass, the terminals are generated in the order *wrote who this*, while in the second pass the order is *wrote this who*. The first traversal for EAA, and the only one for SHDGA are same-to-a-subtree.

4. EFFICIENCY-WISE SUPERIORITY OF EAA OVER SHDGA

The following example is a simplified fragment of a parser-oriented grammar for *yes or no* questions. Using this fragment we will illustrate some deficiencies of SHDGA.

```
..........
(1)  sentence/ques(askif(S)) --> yesnoq/askif(S).
..........
(2)  yesnoq/askif(S) -->
         aux_verb(Num,Pers,Form)/Aux,
         subj(Num,Pers)/Subj,
         main_verb(Form,[Subj,Obj])/Verb,
         obj(_,_)/Obj,
         adj([Verb])/S.
```

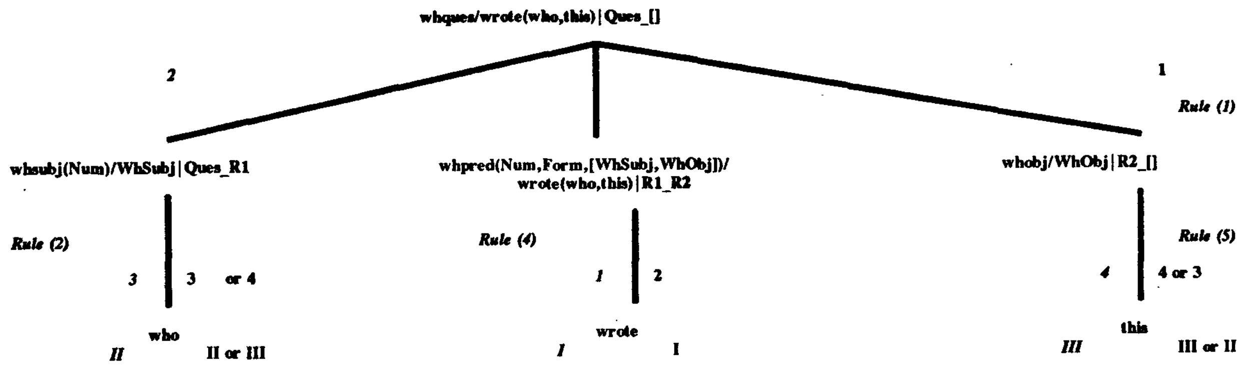

FIGURE 2: EAA's and SHDGA's *STAS* Traversals of *Who Question*'s Analysis Tree.

```
(3)  aux_verb(sing,one,pres_perf)/have(pres_perf,sing)
                    --> [have].
  ..........
(4)  aux_verb(sing,one,pres_cont)/be(pres_cont,
          sing-1) --> [am].
  ..........
(5)  aux_verb(sing,one,pres)/do(pres,sing-1) --> [do].
(6)  aux_verb(sing,two,pres)/do(pres,sing-2) --> [do].
(7)  aux_verb(sing,three,pres)/do(pres,sing-3) -->
                                         [does].
(8)  aux_verb(pl,one,pres)/do(pres,pl-1) --> [do].
  ..........
(9)  subj(Num,Pers)/Subj --> np(Num,Pers,su)/Subj.
(10) obj(Num,Pers)/Obj --> np(Num,Pers,ob)/Obj.
(11) np(Num,Pers,Case)/NP
          --> noun(Num,Pers,Case)/NP.
(12) np(Num,Pers,Case)/NP
          --> pnoun(Num,Pers,Case)/NP.
(13) pnoun(sing,two,su)/you --> [you].
(14) pnoun(sing,three,ob)/him --> [him].
  ..........
(15) main_verb(pres,[Subj,Obj])/see(Subj,Obj)
                       --> [see].
(15a) main_verb(pres_perf,[Subj,Obj])/seen(Subj,Obj)
                                  --> [seen].
(15b) main_verb(perf,[Subj,Obj])/saw(Subj,Obj)
                                   --> [saw].
  ..........
(16) adj([Verb])/often(Verb) --> [often].
```

The analysis tree (given on Figure 3.) for the input semantics *ques (askif (often (see (you,him))))* (the output string being *do you see him often*) is presented below.

Both algorithms start with the rule (1). SHDGA selects (1) because it has the left-hand side nonterminal with the same semantics as the root, and it is a non-chain rule. EAA selects (1) because its left-hand side unifies with the initial query *(-?- sentence (OutString_[]) / ques(askif(often(see(you,him)))))*.

Next, rule (2) is selected by both algorithms. Again, by SHDGA, because it has the left-hand side nonterminal with the same semantics as the current root (yesnoq/askif...), and it is a non-chain rule; and by EAA, because the yesnoq/askif... is the only nonterminal on the right-hand side of the previously chosen rule and it has an instantiated msea (its semantics). The crucial difference takes place when the right-hand side of rule (2) is processed. EAA deterministically selects *adj* for expansion, because it is the only rhs literal with an instantiated msea's. As a result of expanding *adj*, the main verb semantics becomes instantiated, and therefore *main_verb* is the next literal selected for expansion. After processing of *main_verb* is completed, *Subject*, *Object*, and *Tense* variables are instantiated, so that both *subj* and *obj* become ready. Also, the tense argument for *aux_verb* is instantiated (*Form* in rule (2)). After *subj*,

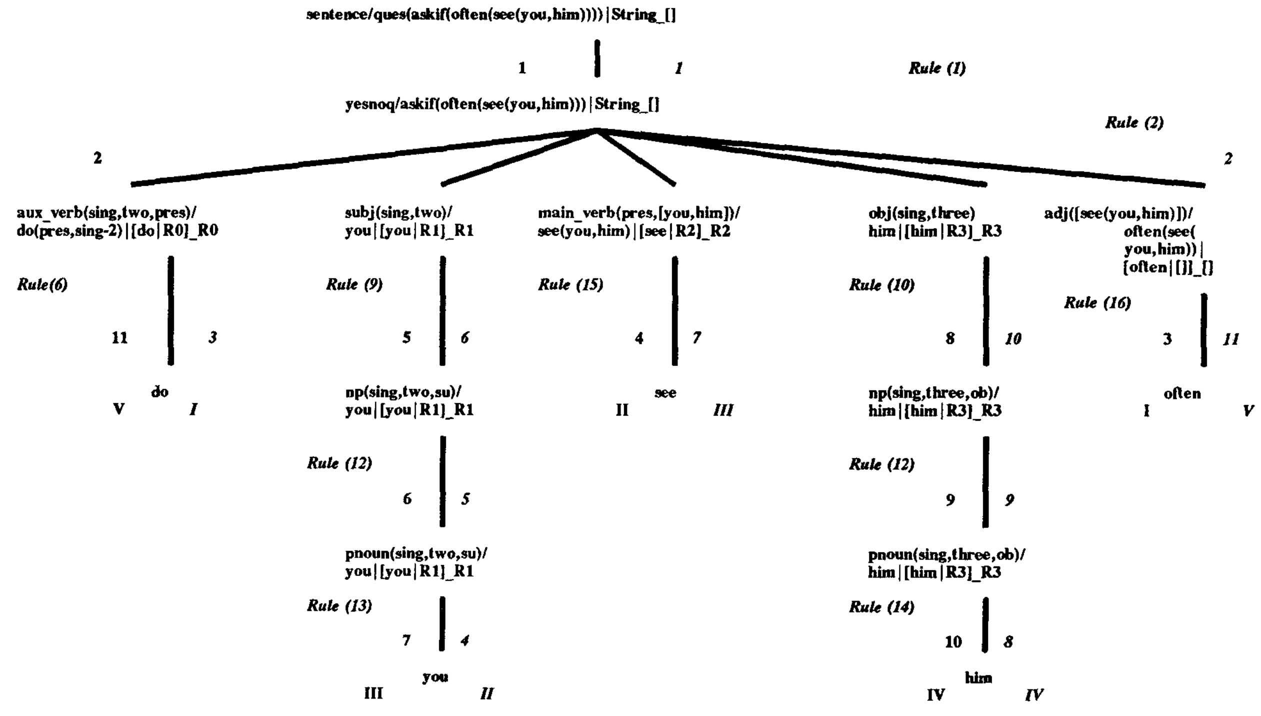
sentence/ques(askif(often(see(you,him))))|String_[]
1
1
Rule (1)
yesnoq/askif(often(see(you,him)))|String_[]
Rule (2)
2
2
aux_verb(sing,two,pres)/ do(pres,sing-2)|[do|R0]_R0
Rule(6)
11
3
do
V
I
subj(sing,two)/ you|[you|R1]_R1
Rule (9)
5
6
np(sing,two,su)/ you|[you|R1]_R1
Rule (12)
6
5
pnoun(sing,two,su)/ you|[you|R1]_R1
Rule (13)
7
4
you
III
II
main_verb(pres,[you,him])/ see(you,him)|[see|R2]_R2
Rule (15)
4
7
see
II
III
obj(sing,three) him|[him|R3]_R3
Rule (10)
8
10
np(sing,three,ob)/ him|[him|R3]_R3
Rule (12)
9
9
pnoun(sing,three,ob)/ him|[him|R3]_R3
Rule (14)
10
8
him
IV
IV
adj([see(you,him)])/ often(see(you,him))| [often|[]]_[]
Rule (16)
3
11
often
I
V

FIGURE 3: EAA's and SHDGA's Traversals of *If Question*'s Analysis Tree.

and *obj* are expanded (in any order), *Num*, and *Pers* for *aux_verb* are bound, and finally *aux_verb* is ready, too.

In contrast, the SHDGA will proceed by selecting the leftmost literal (*aux_verb(Num,Pers,Form)/Aux*) of the rule (2). At this moment, none of its arguments is instantiated and any attempt to unify with an auxiliary verb in a lexicon will succeed. Suppose then that *have* is returned and unified with aux_verb with *pres_perf* as *Tense* and *sing_1* as *Number*. This restricts further choices of *subj* and *main_verb*. However, *obj* will still be completely randomly chosen, and then *adj* will reject all previous choices. The decision for rejecting them will come when the literal adj is expanded, because its semantics is *often(see(you,him))* as inherited from *yesnoq*, but it does not match the previous choices for *aux_verb*, *subj*, *main_verb*, and *obj*. Thus we are forced to backtrack repeatedly, and it may be a while before the correct choices are made.

In fact the same problem will occur whenever SHDGA selects a rule for expansion such that its leftmost right-hand side literal (first to be processed) is not *ready*. Since SHDGA does not check for *readiness* before expanding a predicate, other examples similar to the one discussed above can be found easily. We may also point out that the fragment used in the previous example is extracted from an actual computer grammar for English (Sager's String Grammar), and therefore, it is not an artificial problem.

The only way to avoid such problems with SHDGA would be to rewrite the underlying grammar, so that the choice of the most instantiated literal on the righthand side of a rule is forced. This could be done by changing rule (2) in the example above into several rules which use meta nonterminals *Aux*, *Subj*, *Main_Verb*, and *Obj* in place of literals *aux_verb*, *subj*, *main_verb*, and *obj* respectively, as shown below:

```
..........
yesnoq/askif(S) --> askif/S.
askif/S -->
        Aux, Subj, Main_Verb, Obj,
       adj([Verb],[Aux,Subj,Main_Verb,Obj])/S.
..........
```

Since *Aux*, *Subj*, *Main_Verb*, and *Obj* are uninstantiated variables, we are forced to go directly to *adj* first. After *adj* is expanded the nonterminals to the left of it will become properly instantiated for expansion, so in effect their expansion has been delayed.

However, this solution seems to put additional burden on the grammar writer, who need not be aware of the evaluation strategy to be used for its grammar.

Both algorithms handle left recursion satisfactorily. SHDGA processes recursive chain rules rules in a constrained bottom-up fashion, and this also includes deadlock prone rules. EAA gets rid of left recursive rules during the grammar normalization process that takes place at compile-time, thus avoiding the run-time overhead.

5. MULTI-DIRECTIONALITY

Another property of EAA regarded as superior over the SHDGA is its multi-directionality. EAA can be used for parsing as well as for generation. The algorithm will simply recognize that the top-level msea is now the string, and will adjust to the new situation. Moreover, EAA can be run in any direction *paved* by the predicates' mseas as they become instantiated at the time a rule is taken up for expansion.

In contrast, SHDGA can only be guaranteed to work in one direction, given any particular grammar, although the same architecture can apparently be used for both generation, [SNMP90], and parsing, [K90], [N89].

The point is that some grammars (as shown in the example above) need to be rewritten for parsing or generation, or else they must be constructed in such a way so as to avoid indeterminacy. While it is possible to rewrite grammars in a form appropriate for head-first computation, there are real grammars which will not evaluate efficiently with SHDGA, even though EAA can handle such grammars with no problems.

6. CONCLUSION

In this paper we discussed several aspects of two natural language generation algorithms: SHDGA and EAA. Both algorithms operate under the same general set of conditions, that is, given a grammar, and a structured representation of meaning, they attempt to produce one or more corresponding surface strings, and do so with a minimal possible effort. We analyzed the performance of each algorithm in a few specific situations, and concluded that EAA is both more general and more efficient algorithm than SHDGA. Where EAA enforces the optimal traversal of the derivation tree by precomputing all possible orderings for nonterminal expansion, SHDGA can be guaranteed to display a comparable performance only if its grammar is appropriately designed, and the *semantic heads* are carefully assigned (manually). With other grammars SHDGA will follow non-optimal generation paths which may lead to extreme inefficiency.

In addition, EAA is a truly multi-directional algorithm, while SHDGA is not, which is a simple consequence of the restricted form of grammar that SHDGA can safely accept.

This comparison can be broadened in several directions. For example, an interesting problem that remains to be worked out is a formal characterization of the grammars for which each of the two generation algorithms is guaranteed to produce a finite and/or optimal search tree. Moreover, while we showed that SHDGA will work properly only on a subset of EAA's grammars, there may be legitimate grammars that neither algorithm can handle.

7. ACKNOWLEDGEMENTS

This paper is based upon work supported by the Defense Advanced Research Project Agency under Contract N00014-90-J-1851 from the Office of Naval Research, the National Science Foundation under Grant IRI-89-02304, and the Canadian Institute for Robotics and Intelligent Systems (IRIS).

REFERENCES

[C78] COLMERAUER, A. 1978. "Metamorphosis Grammars." In *Natural Language Communication with Computers*, Edited by L. Bolc. Lecture Notes in Computer Science, 63. Springer-Verlag, New York, NY, pp. 133-189.

[D90a] DYMETMAN, M. 1990. "A Generalized Greibach Normal Form for DCG's." CCRIT, Laval, Quebec: Ministere des Communications Canada.

[D90b] DYMETMAN, M. 1990. "Left-Recursion Elimination, Guiding, and Bidirectionality in Lexical Grammars." To Appear.

[DA84] DAHL, V., and ABRAMSON, H. 1984. "On Gapping Grammars." *Proceedings of the Second International Conference on Logic Programming.* Uppsala, Sweden, pp. 77-88.

[DI88] DYMETMAN, M., and ISABELLE, P. 1988. "Reversible Logic Grammars for Machine Translation." *Proceedings of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages.* Carnegie-Mellon University, Pittsburgh, PA.

[DIP90] DYMETMAN, M., ISABELLE, P., and PERRAULT, F. 1991. "A Symmetrical Approach to Parsing and Generation." *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90).* Helsinki, Finland, Vol. 3., pp. 90-96.

[GM89] GAZDAR, G., and MELLISH, C. 1989. *Natural Language Processing in Prolog.* Addison-Wesley, Reading, MA.

[K90] KAY, M. 1990. "Head-Driven Parsing." In M. Tomita (ed.), *Current Issues in Parsing Technology,* Kluwer Academic Publishers, Dordrecht, the Netherlands.

[K84] KAY, M. 1984. "Functional Unification Grammar: A Formalism for Machine Translation." *Proceedings of the 10th International Conference on Computational Linguistics (COLING-84).* Stanford University, Stanford, CA., pp. 75-78.

[N89] VAN NOORD, G. 1989. "An Overview of Head-Driven Bottom-Up Generation." In *Proceedings of the Second European Workshop on Natural Language Generation.* Edinburgh, Scotland.

[PS90] PENG, P., and STRZALKOWSKI, T. 1990. "An Implementation of A Reversible Grammar." *Proceedings of the 8th Conference of the Canadian Society for the Computational Studies of Intelligence (CSCSI-90).* University of Ottawa, Ottawa, Ontario, pp. 121-127.

[S90a] STRZALKOWSKI, T. 1990. "How to Invert A Natural Language Parser into An Efficient Generator: An Algorithm for Logic Grammars." *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90).* Helsinki, Finland, Vol. 2., pp. 90-96.

[S90b] STRZALKOWSKI, T. 1990. "Reversible Logic Grammars for Natural Language Parsing and Generation." *Computational Intelligence Journal,* Volume 6., pp. 145-171.

[S91] STRZALKOWSKI, T. 1991. "A General Computational Method for Grammar Inversion." *Proceedings of a Workshop Sponsored by the Special Interest Groups on Generation and Parsing of the ACL.* Berkeley, CA., pp. 91-99.

[SNMP89] SHIEBER, S.M., VAN NOORD, G., MOORE, R.C., and PEREIRA, F.C.N. 1989. "A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms." *Proceedings of the 27th Meeting of the ACL.* Vancouver, B.C., pp. 7-17.

[SNMP90] SHIEBER, S.M., VAN NOORD, G., MOORE, R.C., and PEREIRA, F.C.N. 1990. "Semantic-Head-Driven Generation." *Computational Linguistics*, Volume 16, Number 1.

[W88] WEDEKIND, J. 1988. "Generation as Structure Driven Derivation." *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88).* Budapest, Hungary, pp. 732-737.