Using Restriction to Optimize Unification Parsing

Dale Gerdemann [*] [†]

Department of Linguistics
Cognitive Science Group
Beckman Institute for Advanced Science and Technology
University of Illinois

1 Introduction

Since Shieber (1985), restriction has been recognized as an important operation in unification parsing. [1] As Shieber points out, the most straightforward adaptation of Earley's algorithm [2] for use with unification grammars fails because the infinite number of categories in these grammars can cause the predictor step in the algorithm to go into an infinite loop, creating ever more and more new predictions (i.e. the problem is that new predictions are not subsumed by previous predictions). The basic idea of restriction is to avoid making predictions on the basis of all of the information in a DAG, but rather to take some subset of that information (i.e. a restricted DAG–henceforth RD) and use just that information to make new predictions. Since there are only a finite number of possible RDs the predictor step will no longer go into the infinite loop described above. The price you pay for this move is that some spurious predictions will be made, but as Shieber points out, the algorithm is still correct since any spurious predictions will be weeded out by the completer step.

[*]Cognitive Science Group, Beckman Institute, 405 N. Mathews, Urbana, Ill 61801; dale@tarski.cogsci.uiuc.edu

[†]I would like to thank Alan Frisch, Erhard Hinrichs, Lucja Ivańska, Jerry Morgan, Mike Mendelson, and Tsuneko Nakazawa for their useful comments. Any deficiencies must rest with me. Thanks also to the UIUC Cognitive Science/Artificial Intelligence fellowship committee for the support that made this research possible.

[1]By unification parsing I mean parsing of unification grammars. See Seifert (1988) for a precise definition of a unification grammar.

[2]I will assume familiarity with the basic steps of Earley's algorithm as presented in Earley (1970). For an introduction to Earley's algorithm and its relationship to chart parsing in general see Winograd (1983).

Shieber's use of restriction in the predictor step is by now well established. On the other hand, there has been little discussion of the uses of restriction in other stages of parsing. In this paper, I will argue that restriction can be used to advantage in at least three additional ways. First, restriction can be used to significantly speed up the subsumption check on new predictions. Second, it can be used in the completer step in order to speed up the process of finding the correct states in the state sets to be completed. And third, it can be used to add a lookahead component to the unification parser. I will begin this paper by briefly reviewing Shieber's use of restriction and then I will discuss the three additional uses for restriction mentioned above.

2 Restriction in the Predictor Step

The original motivation for restriction was to avoid infinite cycles in the predictor step of Earley's algorithm. Shieber illustrates this problem with a "counting grammar" but the same point can be made using a type of grammar that is somewhat more familiar in recent linguistic theory. Specifically, infinite cycles can arise in grammars that handle subcategorization with list valued features such as Head Driven Phrase Structure Grammar (Pollard and Sag, 1987) or PATR style grammars (Shieber, 1986). To illustrate the problem, suppose that we are parsing a sentence using a grammar with the PATR style rules in (1,2). The problem of non-termination can arise with this grammar since rule (2) allows for lexical items with indefinitely long subcategorization lists.

(1) $x0 \rightarrow x1\ x2$

$$\left[\begin{array}{ll} x0 & \left[\begin{array}{ll} cat & s \end{array}\right] \\ x1 & [1]\left[\begin{array}{ll} cat & np \end{array}\right] \\ x2 & \left[\begin{array}{ll} cat & vp \\ subcat & \left[\begin{array}{ll} first & [1] \\ rest & end \end{array}\right] \end{array}\right] \end{array}\right]$$

(2) $x0 \rightarrow x1\ x2$

$$\left[\begin{array}{lll} x0 & \left[\begin{array}{ll} cat & vp \\ subcat & [1] \end{array}\right] & \\ x1 & [1] & \left[\begin{array}{ll} cat & vp \\ subcat & \left[\begin{array}{ll} first & [2] \\ rest & [1] \end{array}\right] \end{array}\right] \\ x2 & [2] & \end{array}\right]$$

The first step in parsing a sentence with this grammar is to find a rule whose left hand side unifies with the DAG described by the path equation $\langle cat \rangle = s$

(i.e. the start DAG). Since the rule in (1) satisfies this requirement, the next step is to make a prediction for the $x1$ daughter. In Earley's algorithm as it was originally formulated (Earley 1970), the prediction for $x1$ would simply be its category label (i.e. np). In this unification style grammar, however, category labels are just features like any other feature. Since the DAGs associated with each of the non-terminals ($x0, x1, \ldots, xn$) in a rule may express just partial information about that non-terminal, it is possible that some non-terminals (such as $x2$ in the second rule) will not be associated with any category label at all. The natural solution, then, would be to make a prediction using the entire DAG associated with a given non-terminal. Suppose, now, that we have parsed the np in rule (1) and we're ready to parse $x2$. The DAG associated with $x2$ would be (3).

(3) $$\begin{bmatrix} cat & vp \\ subcat & \begin{bmatrix} first & \begin{bmatrix} cat & np \end{bmatrix} \\ rest & end \end{bmatrix} \end{bmatrix}$$

When this DAG unifies with the category on the left hand side of (2) we get the rule shown in (4).

(4) $$x0 \rightarrow x1\ x2$$
$$\begin{bmatrix} x0 & \begin{bmatrix} cat & vp \\ subcat & \boxed{2} \end{bmatrix} \\ x1 & \begin{bmatrix} cat & vp \\ subcat & \begin{bmatrix} first & \boxed{1} \\ rest & \boxed{2} \begin{bmatrix} first & \begin{bmatrix} cat & np \end{bmatrix} \\ rest & end \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ x2 & \boxed{1} \end{bmatrix}$$

Now, following the same procedure, the predictor would next make a prediction for the non-terminal $x1$ in (4). It can easily be seen that when the DAG associated with $x1$ unifies with the left hand side of rule (2) the predicted rule is almost the same as (4) except that the value for ⟨*subcat rest*⟩ in (4) becomes the value for ⟨*subcat rest rest*⟩ in the new predict ı. In fact, the predictor step can continue making such predictions ad infinitum and, crucially, the new predictions will not be subsumed by previous predictions.

To solve this problem Shieber proposes that the predictor step should not use all of the information in the DAG associated with a non-terminal, but rather it should use some limited subset of that information. Of course, when some nodes of the DAG are eliminated the predictor step can overpredict, but this does not affect the correctness of the algorithm since these spurious predictions will not be completable. Shieber's proposal is basically that before the predictor step is applied, a RD should be created which contains just the information associated with a finite set of paths (i.e. a restrictor). [3] In this way, Shieber's algorithm allows an infinite number of categories to be divided into a finite number of equivalence classes. Since the number of possible RDs is finite it becomes impossible to make the kind of infinite cycle of predictions illustrated above.

Primarily for notational reasons, I will define restriction in a slightly different manner from Shieber (1985). For our purpose here we can define the RD D' of DAG D to be the least specific DAG D' $\sqsubseteq$ D such that for every path P in the restrictor if the value of P in D is atomic then the value of P in D' is the same as the value of P in D and if the value of P in D is complex then the value of P in D' is a variable. This differs from Shieber's definition in that reentrancies are eliminated in the RD. Thus the RD is not really a DAG but rather is a tree and hence it can be represented more easily by a simple list structure. For example,given the restrictor [⟨a b⟩, ⟨d e f⟩, ⟨d i j f⟩], the RD for the DAG in (5) (from Shieber 1985) will be represented by the indented list shown in (6), in which variables are indicated by [].[4]

(5)
$$\begin{bmatrix} a & \begin{bmatrix} b & c \end{bmatrix} \\ d & \begin{bmatrix} e & \boxed{1}\begin{bmatrix} f & \begin{bmatrix} g & h \end{bmatrix} \end{bmatrix} \\ i & \begin{bmatrix} j & \boxed{1} \end{bmatrix} \\ k & l \end{bmatrix} \end{bmatrix}$$

(6)
$$\begin{array}{l} [[a,[[b,c]]], \\ \;[d,[[e,[[f,[]]]], \\ \quad\;\; [i,[[j,[[f,[]]]]]]]]] \end{array}$$

3 Restriction in the Subsumption Test

The first use of restriction I will discuss involves the subsumption check on new predictions. In the original Earley's algorithm (Earley 1970), a check was made on each new prediction to see that an identical prediction had not already been made in the same state set. Of course, if duplicate predictions are retained the parser can fall into the left recursion trap. In Shieber's adaptation, however, this identity check is changed to the more general notion of a subsumption check. If a new DAG is predicted that is subsumed by a previous (more general) DAG,

[3]The question of how to select an appropriate restrictor for greatest efficiency must remain a question for further research. See the conclusion of this paper for further discussion.

[4]Eliminating reentrancies from RDs may also be a reasonable thing to do from a computational point of view. Judging from the particular restrictors used in Shieber (1985,1986) it would appear that reentrancies rarely occur in RDs. However, for some purposes it may be desirable to include more information in RDs. A possible example would be the use of parsing algorithms for generation, in which it would be desirable to use as much top down information as possible.

the new DAG is not retained since any DAGs that could be predicted on the basis of the new DAG could already have been predicted on the basis of the more general DAG. Clearly, the move from an identity check to a subsumption check is the right sort of move to make, but a subsumption check on arbitrarily large DAGs can be an expensive operation. This seems to be an ideal area in which restriction could be used to optimize the algorithm.

The move I propose is the following. Initially, new predictions are made in the manner suggested by Shieber; i.e. make a RD for the category "to the right of the Dot" and then collect all the rules from the grammar whose left hand side category unifies with this RD–these rules then constitute the new predictions. At this point I suggest that the RD used to find these predictions should be retained along with the new predictions; that is, a list of RDs that have been used to make predictions should be kept for each state set. I will call this list the RD_List. Then, the next time the parser enters the predictor step and creates a new RD from which to make new predictions, a subsumption check can be made directly between this RD and the RD_List. If the new RD is subsumed by any member of the RD_List then we can immediately give up trying to make any new predictions from this RD. Any predictions made from th RD would necessarily already have been made when the predictor encountered the more general RD in the RD_List. Thus we avoid both the expense of making new predictions and the expense of applying the subsumption test to weed these new predictions out. Moreover, since RDs are typically very small (at least given the sample restrictors given in Shieber (1985,1986)), the subsumption test that is performed on them can be applied very quickly.

As an example, suppose that some set of predictions has already been made using the RD, [[cat, np]], then there is no point in making predictions using [[cat, np],[num, sing]] since any such predictions would necessarily fail the subsumption check; i.e., rules expanding singular noun phrases are more specific than (or subsumed by) rules expanding noun phrases unspecified for number. This particular case probably does not arise often in actual parsing, but cases of left recursion do arise for which this optimization can make a very significant difference in processing speed. In fact our experience with the UNICORN natural language processing system (Gerdemann and Hinrichs 1988), has shown that for grammars with a large amount of left recursion, this simple optimization can make the difference between taking several minutes of processing time and several seconds of processing time.

4 Restriction in the Completer Step

The next use of restriction I propose involves the completer step. The completer applies, in Earley's algorithm, at the point where all of the right hand side of a rule in some state has been consumed, i.e., the point at which the "Dot" has been moved all the way to the right in some rule. At this point the completer goes back to the state set in which the state to be completed was originally predicted and searches for a prediction in this state set which has a category "to the right of the Dot" which can unify with the mother node of the rule in the state to be completed. This search can be quite time consuming since the completer must attempt to perform a unification for each state in this state set.

In each state, there is a variable F which indicates in which state set that state was predicted so the completer can immediately go back to the Fth state set in order to make the completion. But there is no variable which indicates which state in the Fth state set could have been responsible for making that prediction. And, in fact, it would be quite difficult to implement such a direct backpointer since in many cases a particular state is really only indirectly responsible for some prediction in the sense that it would have been responsible for the prediction if it had not been for the subsumption check. For example, suppose we try to implement a system of backpointers as follows. Each state will be a quintuple ⟨Lab,BP,Dot,F,Dag⟩ where Lab is an arbitrary label, BP is a kind of backpointer which takes as its value the label of the state that was responsible for predicting the current state and Dot, F, and Dag are as in Shieber's adaptation of Earley's algorithm; i.e., Dot is a pointer to the current position in the rule represented by Dag, and F is the more general kind of backpointer which only indicates in which state set the original prediction was made. To illustrate the problem with this scheme, consider the partial state set in (7), in which the subscripted i indicates that this is the ith state set.

(7) $_i[\ldots[Lab1, BP1, Dot1, F1, Dag1], [Lab2, BP2, Dot2, F2, Dag2], \ldots]$

Now suppose the RD for Dag1 is [[cat,np]] and that the RD for Dag2 is [[cat,np],[num,sing]]. When the predictor looks at state Lab1 it will make some number of predictions with backpointers to Lab1 as in (8) (For example, [Lab3,Lab1,0,i,Dag3] is a new state with an arbitrary label, Lab3, a backpointer to state Lab1, the Dot set at 0 indicating the beginning of the left hand side, F set to i indicating that the prediction was made in state set i, and Dag3 representing the new rule).

(8) $_i[\ldots[Lab1, BP1, Dot1, F1, Dag1], [Lab2, BP2, Dot2, F2, Dag2],$
$[Lab3, Lab1, Dot3, Dag3], [Lab4, Lab1, Dot4, F4, Dag4], \ldots]$

But when the predictor looks at Lab2 no predictions will be made since its RD is subsumed by the RD of Lab1. Thus even though (without the subsumption check) Lab2 could have been responsible for the predictions Lab3 and Lab4, no backpointers are created for Lab2.

It is at this point that RDs can again help us out. The idea is that when the predictor attempts to make predictions on the basis of some state it adds a RD to that state and to all predictions made from that state as a kind of marker (or coindexing between a state and the predictions resulting from that state). The RD used for this coindexing will be either 1.) the RD used to make the predictions or 2.) if no predictions were made because a more general RD had already been used to make predictions, then this more general RD is used as the marker. Now the completion step is greatly simplified. The completer can go back to the Fth state set and attempt unification only on states that have identical RD-markers. Clearly this move eliminates many attempted unifications that would be doomed to failure. To implement this idea states will be defined as quintuples ⟨BP,FP,Dot,F,Dag⟩ where BP is a RD acting as a backpointer, FP is a RD acting as a forward pointer and F,Dot, and Dag are as before. Now the analog of (7) will be (9).

(9) $_i[\ldots[BP1, FP1, Dot1, F1. Dag1], [BP2, FP2, Dot2, F2, Dag2], \ldots]$

In (9) BP1 and BP2 will each be instantiated to the value of the RD responsible for the prediction which created their respective state. FP1 and FP2, however will be uninstantiated variable since these two states have not yet been responsible for creating any new predictions. Now assuming that the RDs for Dag1 and Dag2 are as in (7) then when the predictor applies to the first state shown in (9), the result will be the state set shown in (10).

(10)
$$\begin{aligned}&{}_i[\ldots[BP1, [[cat, np]], Dot1, F1, Dag1],\\ &[BP2, FP2, Dot2, F2, Dag2],\\ &[[[cat, np]], FP3, Dot3, F3, Dag3],\\ &[[[cat, np]], FP4, Dot4, F4, Dag4], \ldots]\end{aligned}$$

Then when the predictor looks at the second state in (10), no predictions will be made as before, however the predictor will register the attempt to make a prediction by instantiating the variable FP2 as in (11).

(11)
$$\begin{aligned}&{}_i[\ldots[BP1, [[cat, np]], Dot1, F1, Dag1],\\ &[BP2, [[cat, np]], Dot2, F2, Dag2],\\ &[[[cat, np]], FP3, Dot3, F3, Dag3],\\ &[[[cat, np]], FP4, Dot4, F4, Dag4], \ldots]\end{aligned}$$

Now whenever the descendants of states 3 and 4 are ready to be completed, it will be easy to go back to this state set and find the states whose forward pointers are identical to the backpointers of the states to be completed. Thus many candidates for completion are immediately ruled out.

5 Restriction Used in Lookahead

The final use for restriction that I propose involves lookahead. Lookahead is one aspect of Earley's algorithm which clearly needs modification in order to be used efficiently with unification grammars or natural language grammars in general. In the original algorithm, a calculation of lookahead was performed as part of the prediction step. A simple example can show the problem with Earley's version of this procedure. In the S → NP VP rule, when the predictor makes a prediction for NP, it is required to add a state for each possible lookahead string that can be derived from the VP. But given the large number of verbs or adverbs that can start a VP in a natural language this would require adding a huge number of states to the state set. Clearly we don't want to simply list all the possible lookahead strings, but rather the correct approach would be to find what features these strings have in common and then add a smaller number of states with feature based lookaheads.

Aside from the question of what kind of lookahead to calculate, there are two other questions that need to be considered: first the question of when to calculate lookahead and second how to calculate it. Beginning with the when question, it is clear that unification grammars require lookahead to be calculated at a later point than it is in Earley's approach. The reason for this is illustrated by rules like (2) repeated here as (12)

(12)

$$x0 \rightarrow x1\ x2$$

$$\begin{bmatrix} x0 & & \begin{bmatrix} cat & vp \\ subcat & [1] \end{bmatrix} \\ x1 & [1] & \begin{bmatrix} cat & vp \\ subcat & \begin{bmatrix} first & [2] \\ rest & [1] \end{bmatrix} \end{bmatrix} \\ x2 & [2] & \end{bmatrix}$$

According to Earley's approach, when a prediction is made for x1, the lookahead for $x2$ should be calculated. But in this case, no features for $x2$ will be specified until after x1 is parsed. This is an extreme situation, but it illustrates a general problem. It is the normal case in a unification grammar for the result of parsing one category to affect the feature instantiations on its sister. Clearly, what needs to be done in this case is to parse x1 and *then* perform a lookahead on $x2$. Thus, lookahead should be calculated for a category immediately before the predictor applies to that category; i.e., lookahead can be considered a quick check to be made immediately before applying prediction. Unlike Earley's original algorithm, then, it is not necessary to put a lookahead string into a state to be checked at a later point.

The question, then, is how to calculate lookahead. In Earley's version of the algorithm, there is a function, H_k which when applied to a category C returns a set of k-symbol strings of terminals which could begin a phrase of category C. When applied to unification grammars, however, the problem of having an infinite number of categories again appears. We certainly cannot list possible strings of preterminals that can begin each category. It is clear, then, that some form of restriction is again going to be necessary in order to implement any kind of lookahead. One, relatively simple, way of implementing this idea is as follows. When the predictor applies to a category C, the first thing it does is make a RD for C. Then a table lookup is performed to determine what preterminal categories could begin C. Since there are potentially infinite preterminal categories, restriction must be applied here too. So more precisely, the table lookup finds a set of RDs that could unify with whatever actual preterminal could begin a phrase of category C. Let us call these RDs the preterminal RDs. Then before the predictor can actually make a prediction a check must be performed to verify that the next item in the input is an instance of a category that can unify with one of the preterminal RDs. If the check fails, then the prediction is abandoned. All that remains is to specify how the lookup table is constructed. One way such a table might be constructed would be to run the parser in reverse for generation as in Shieber (1988) . Thus, for each possible RD (given a particular restrictor), the generator is used to determine what preterminal RDs can begin a phrase of this category.

6 Conclusion

I have argued here that restriction can be used in unification parsing to effect three optimizations. First, it can be used to greatly speed up the subsumption test for adding new predictions to the state set, second it can be used to speed up the searching used in the completer step, and finally it can be used to implement a form of lookahead. The first two of these uses have been fully implemented within the UNICORN natural language processing system (Gerdemann and Hinrichs 1988). The use of restriction with lookahead is still under development.

In general, the fact that unification grammars may have categories of indefinite complexity necessitates some way of focusing on limited portions of the information contained in these categories. It seems quite likely, then, that restriction would be useful even in other parsing algorithms for unification grammars. The primary question that remains is what portion of the information in complex DAGs should be used in these algorithms; that is, the question is how to choose a restrictor. Up to now, no general principles have been given for choosing a restrictor for greatest efficiency. Given the proposals in this paper, it becomes even more critical to find such general principles since restriction can affect the efficiency of several steps in the parsing algorithm.

References

[1] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 1970.

[2] Dale Gerdemann and Erhard Hinrichs. UNICORN: a unification parser for

attribute-value grammars. *Studies in the Linguistic Sciences*, 1988.

[3] Carl Pollard and Ivan Sag. *An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamentals. CSLI Lecture Notes No. 13*, Chicago University Press, Chicago, 1987.

[4] Roland Seiffert. Chart-parsing of unification-based grammars with ID\LP-rules. In Ewan Klein and Johan van Benthem, editors, *Categories, Polymorphism and Unification*, pages 335–54, CCS/ILLI, Edinburgh/Amsterdam, 1987.

[5] Stuart Shieber. *An Introduction to Unification-Based Approaches to Grammar. CSLI Lecture Notes No. 4*, Chicago University Press, Chicago, 1986.

[6] Stuart Shieber. A uniform architecture for parsing and generation. In *COLING-88*, pages 614–9, 1988.

[7] Stuart Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *ACL Proceedings, 23rd Annual Meeting*, pages 145–52, 1985.

[8] Terry Winograd. *Language as a Cognitive Process: Syntax.* Ablex, Norwood, 1983.