Adventures in Multi-dimensional Parsing: Cycles and Disorders

Kent Wittenburg

Bellcore, 445 South St., MRE 2A-347,
Morristown, NJ 07962-1910, USA
email: kentw@bellcore.com

Abstract

Among the proposals for multidimensional grammars is a family of constraint-based grammatical frameworks, including Relational Grammars. In Relational languages, expressions are formally defined as a set of relations whose tuples are taken from an indexed set of symbols. Both bottom-up parsing and Earley-style parsing algorithms have previously been proposed for different classes of Relational languages. The Relational language class for Earley style parsing in Wittenburg (1992a) requires that each relation be a partial order. However, in some real-world domains, the relations do not naturally conform to these restrictions. In this paper I discuss motivations and methods for predictive, Earley-style parsing of multidimensional languages when the relations involved do not necessarily yield an ordering, e.g., when the relations are symmetric and/or nontransitive. The solution involves guaranteeing that a single initial start position for parsing can be associated with any member of the input set. The domains in which these issues are discussed involve incremental parsing in interfaces and off-line verification of multidimensional data.

1 Introduction

Relational Languages, those sets of expressions that are generable (or recognizable) by Relational Grammars, are characterized as relations on sets of symbols or, in practice, structured objects (Crimi et al., 1991; Golin — Reiss, 1990; Helm — Marriott, 1986, 1990; Wittenburg et al., 1991; Wittenburg, 1992a, 1992b, 1993). Sentential forms and elements of derivations are formally defined as sets of relations, each of which in turn is a set of ordered tuples, in the symbol (or object type) vocabulary set. This approach to defining multidimensional languages is compatible with unification-based approaches since the relations can be defined in unification-based grammars as constraints. Constraint Logic Programming or other approaches that enhance the usual notion of equality as the only structural constraint on terms can be employed in order to express the more free-form constraints required in multidimensional languages (see Crimi et al., 1991; Helm — Marriott, 1986, 1990; Wittenburg et al., 1991; Wittenburg, 1993). This approach to grammar and language definition generalizes over many other proposals for multidimensional grammars in the literature since arrays, graphs, and specialized spatial data structures can easily be modeled as sets of relational tuples.

Various parsers have been previously proposed for different classes of Relational languages. Bottom-up algorithms (Golin, 1991; Wittenburg et al., 1991) are the most straightforward and general, although they they are not suited to all applications. More efficient deterministic techniques proposed for graph grammars by Flasinski (1988; 1989) have been adapted by Ferrucci et al. (1991) to a constraint-based grammar framework, but the restrictions on relations and grammar productions make it unclear that the class of languages is widely useful. On the other hand, this algorithm has been shown to have a low polynomial bound on complexity. An Earley-style algorithm (Earley, 1970) has been proposed by

Wittenburg (1992a) for a larger class of nondeterministic languages whose relations are partial orders.

The limitations of these previously proposed algorithms have become evident in two domains for Relational Grammars under current investigation: on-line incremental parsing of visual language expressions (Wittenburg et al., 1991; Weitzman and Wittenburg, 1993) and off-line verification of multidimensional data. While bottom-up parsing may be employed, there are reasons to consider predictive parsing techiques. In the case of incremental parsing of visual languages consisting of connected diagrams or geometric layouts, one might use predictive parsing to detect errors as soon as they occur or to offer incremental directives (e.g., the analogue of completion in command input) to drawing or palette selections. However, to use deterministic algorithms one must use grammars that are often not expressive enough for the constraints desired. For example, no non left- or right-unique relations are allowed with the Flasinski approach and yet geometric layouts of objects of different sizes invariably include relations that, say, have two smaller objects both in some *below* relation to a larger object. The Earley-style algorithm of Wittenburg (1992a) is unsuited to incremental interface applications because more flexibility in a parser's scanning order is desired than is afforded by this algorithm. Unlike with one-dimensional text, it is not easy to anticipate in a two- or n-dimensional world exactly what orderings will seem most natural to users. It seems clear that for most diagramming languages, for example, interfaces that allow a good deal of flexibility in how users build up diagrams would be preferred over interfaces that insist that users build up diagrams in prespecified orders that happen to conform, say, to temporal control relations that are reflected in the arcs of the diagrams. Further, many visual languages contain relations that may be symmetric or nontransitive, in which case the Wittenburg (1992a) algorithm is not usable. One set of examples arises with languages whose primitives are directed line segments. Relations between line segments such as head-to-head or tail-to-tail are symmetric, precluding the use of algorithms depending on the intrinsic ordering properties of the relations to direct scanning. Cycles are of course generally common in flowchart diagramming languages to represent control loops.

In the case of off-line data verification and correction, the size of the input sets involved may preclude bottom-up parsing for efficiency reasons. The following scenario from a current Bellcore application domain is an example of the problem. Suppose one is verifying that all the line segments that comprise the border of each region in a mapping database in fact enclose that region and that any attributes of the data are in conformance. One approach is to define grammars that independently combine the line segments surrounding each region to yield a closed polygon of some form. But blind bottom-up parsing would take each and every line segment to be the start of the polygon constituent, rebuilding it many, many times over. In geographical data sets such as the U.S. Census Bureau's Tiger database, the number of line segments representing the border of a single state typically reaches into the thousands. It is not hard to see that such a redundant algorithm would be ill-advised. Not only is it inefficient, but the problem of error detection is, at least on the face of it, made more difficult than if the parsing enumeration were systematically ordered and based on systematic prediction.

The goal addressed in this work is to design a predictive, Earley-style algorithm for Relational Grammars without relying on the relations themselves to provide scanning orderings. Such an algorithm allows predictive parsing methods to be employed with higher-dimensional languages whose relational graphs contain cycles, i.e., one or more of the relations are not partial orders. To design such an algorithm requires solving the problem of finding a start element for the parser to begin its predict and scan operations. The Wittenburg (1992a) algorithm uses minimal elements of the relations to initialize the parser at possibly multiple starting positions. The goal here is to allow the parser to start with a single arbitrary member of the input set and still guarantee completeness.

The remainder of this paper is structured as follows. First, as a review and refinement of work to date, a Relational Grammar formalism for practical applications developed and implemented at

Bellcore is summarized, followed by relevant aspects of Earley-style parsing from Wittenburg (1992a). We then turn to the question of how to define a suitable subclass of Relational Grammars for ensuring that parsing may be initiated from an arbitrary starting point. An Earley-style parsing algorithm and example trace are presented next. The conclusion includes remarks regarding the extensions to this work necessary to fully solve the problems of incremental, predictive parsing for higher-dimensional languages.

2 A Relational Grammar Formalism

Relational Grammars are motivated by domains in which the sets of objects to be generated or analyzed can not comfortably be represented as strict linear orders of symbols. Examples include expressions in 2-D such as mathematics notation, flowcharts, or schematic diagrams; 2-D or 3-D graphical layouts and displays, perhaps with time or additional media as added dimensions; and n-dimensional data to be found in empirical data collections or various types of databases. One must generalize the data type of language expressions from, say, one-dimensional arrays (strings) to 2-, or perhaps n-dimensional arrays or, more generally, to graphs of relations where the nodes represent data in unrestricted formats. Notions of replacement in derivations have to be generalized also, and, as attested by the literature on array and graph grammars, there are many variations on how to define grammar productions and the notion of replacement (called the embedding problem in the graph-grammar literature).

The Relational Language (RL) framework provides for an abstraction over the particular data structures used to hold the object sets being processed. The grammar productions make mention of the relations expected to hold in the data but there is flexibility in choosing exactly how the data is represented and stored as well as the implementation of how the relations are checked or queried. Such an approach accommodates many kinds of data representations, including array or graph structures, K-D trees, or even commercial databases in which indexings may be precisely tuned for efficiency.

Combining input objects during the parsing process can be characterized neutrally through set operations — set union then being a very general analog of string concatenation. It is natural to think of a grammar rule as providing a definition of a composite (nonterminal) object as a set (usually nonunary) whose type is the symbol on the left-hand-side of the rule and whose parts are the union of the parts of the objects whose types are the symbols on the right-hand-side of the rule. Derivations are defined as a sequence of replacements that are headed by a type that is a root symbol of the grammar and terminate in a set of objects whose types are taken from the vocabulary of terminal symbols. An important characteristic of the RL approach is that derivations are trees, as is the case in conventional context-free grammars. An effect of this restriction is that no object may be used more than once per derivation.

The most significant new requirement for the grammar formalism is that it has to provide a means for specifying possible combining relations explicitly. Specifying combining relations in Relational Grammars is done by stating, for each element in the right-hand-side of a non-unary rule, what relation it has to stand in with respect to at least one other right-hand-side element. Operationally, a parser finds relevant input for combination by executing queries formed from the relational constraints.

Besides the unification-based approaches to Relational Languages mentioned above, a more efficient grammar compiler has also been implemented (Wittenburg, 1992b) that incorporates a form of "pseudo-unification" (see Tomita, 1990). One may consider the pseudo-unification-based formalism as syntactic sugar for an underlying unification system. From this point of view, the results of this paper generalize to the full family of extended unification-based approaches and thus it is not the case that we are dealing here with yet another special-case grammar formalism. Nevertheless, we will use the pseudo-unification-based formalism in this dicussion since it is less verbose than the full unification-based specification. It also of course affords the possibility that specialized algorithms may be found that can be proved more efficient than general unification.

Example 1.

```
(defrule (Subtree-rule example-grammar)
    (0 Subtree)
    (1 prim)
    (2 Row)
    :expanders (below 2 1)
    :predicates (centered-in-x 1 2))
```

Example 1 shows a defining form for a simple rule that states that an object of type Subtree can be composed out of two objects of type prim and Row, as long as they stand in the stated *below* and *centered-in-x* relations. The integers in the textual rule definition act as references to rule elements: the left-hand-side of a rule is conventionally marked as 0; the one or more right-hand-side elements are numbered 1... n. The backbone of this rule thus could be written as Subtree → prim Row. A relational constraint such as (below 2 1) is to be interpreted as a requirement that the object matching rule element 2 (of type Row in this case) must stand in the *below* relation to the object matching rule element 1 (of type prim).

During parsing, relational constraints either have the effect of generating possible candidates for rule element matches or filtering candidate matches that have been proposed. Relational constraints immediately following the keyword :expanders act as a generators. These relations must be binary, and the parser will execute a query based on these relational expressions as it explores candidates to match rule elements. For example, the query (below :? i) would be executed when matching rule Subtree-rule in order to expand the match to the second right-hand-side element, assuming i is an index to the input matching the first right-hand-side element. We call the binary, generating relations *expander* relations, since their main role is to expand rule matches. In addition, one may include further relational constraints (of any arity). These non-initial constraints will be executed as predicates. In Example 1, (centered-in-x 1 2) is the only predicate.

Figure 1 shows a graphical depiction of the rule in Example 1 in which composition is represented as spatial enclosure. Thus one sees that the Subtree object is composed of the prim and Row objects. The arrows represent the required spatial relations.

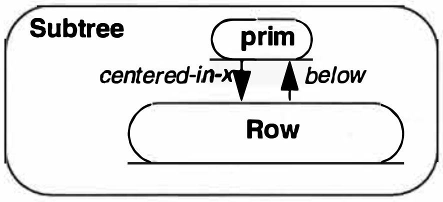

Figure 1: A simple layout rule

At a definitional level, the right-hand-side elements of RG rules are unordered — what really matters are the relational constraints, which only partially determine the order in which a parser might match the rule elements. But a parser is going to have to match rule elements in some order or other. In bottom-up parsing, it is possible to choose only a single ordering of the right-hand-side elements of each production. Wittenburg et al. (1991) discuss the constraints in choosing such an ordering. The following constraint, which we refer to as the connectedness constraint, must hold of an order for right-hand-side rule elements in RGs:

Restriction 1. For an ordering of rhs rule elements $D_1 \ldots D_n$, there must exist at least one expander constraint between each element D_j , $1 < j$, and an element D_i where $i < j$.

That is, considered as a graph with the expander relations as arcs, the right-hand-side of a rule must be connected and, when ordered, each element in turn must be connected to some other element earlier in the ordering. This requirement implies that this class of Relational Grammars can generate only connected relation graphs since, for every production, we assume that there is at least one ordering that meets this condition. Since the expander constraints are going to provide the parser with a query function that can find the candidates to combine with next at each step, this restriction ensures that an expander query can fire at each stage in the rule match. Once an ordering is found, a grammar compiler can place an expander or predicate expression with the first rule element in which all the arguments of the relational expression will be bound.[1]

[1]Ordering the constraints in a grammar compiling operation as discussed here obviates the need for residuation in unification operations, the basis for the unification extensions discussed in Wittenburg (1993).

In remaining examples in this section, we will assume that the rules are ordered as shown, and we will associate the relational constraints with the relevant rule elements rather than list them at the end of the rule definition.

A definition useful in subsequent sections on parsing follows:

Definition 1. We know, given Restriction 1, that for every ordered production, an expander constraint of the form (rel x y) must exist for every ordered daughter at position $j > 1$ where either x or y will be grounded by matching a daughter at position j and either x or y (whichever doesn't satisfy this previous condition) is already grounded by a daughter at a position $i < j$. These arguments are defined as the to-be-bound argument and the already-bound argument, respectively, at position j.

A fundamental issue in Relational Grammar representation is whether to allow nonterminals to appear as direct arguments to relational constraints. The natural interpretation of a nonterminal in such a constraint is that it is a reference to the set of input objects in its derivational yield. Let us presume a grammar having the rule in Example 1 has other rules expanding the nonterminal category Row in such a way that a bottom-up parser can build up a horizontally aligned set of primitive objects. Consider the effect of executing the query (below :? i), mentioned earlier, when parsing. If this query is executed against the original input only, it will never find a Row object since the existence of such an object is an artifact of parsing. A solution to this problem requires dynamic updating of what we call an object store, which starts out as a collection of input objects. Composite objects are introduced as the parser finds them through rule matches. A grammar thus must define derived nonterminal objects in terms of terminal objects. For example, if input data is characterized as rectangular regions (which is reasonable for many graphical applications), then composites introduced through rule matches might be defined as the summation of the rectangular regions of the rule's daughters.

Including relational constraints directly on composites is reasonable when using bottom-up parsing, but it complicates the definition of Relational Grammars as generative systems since the composition-of relation must in principle be reversible. Further, significant problems are introduced for Earley-style parsing, or any other form of predictive parsing, as discussed in Wittenburg (1992a). The alternative is to write grammars that state relational constraints only on individuals in the input set and use feature percolation to pass up bindings of these individuals as attribute values in derivations. We will refer to this latter subclass of Relational Grammars as Atomic Relational Grammars (ARGs), noting that the most significant restriction is that the arguments of relational constraints must be atomic.

As an illustration, consider Example 2, the rule set of a flowchart grammar fragment. The root symbol for this grammar is Flowchart.

Example 2: Flowchart Grammar.

```
(defrule (flowchart flowchart-grammar)
    (0 Flowchart (setf (in 0) 1
                       (out 0) 3)
    (1 oval)
    (2 P-block (connects-to 1 (in 2)))
    (3 oval (connects-to (out 2) 3))))

(defrule (conditional flowchart-grammar)
    (0 P-block (setf (in 0) 1
                     (out 0) 3))
    (1 diamond)
    (2 P-block (Y-connects-to 1 (in 2)))
    (3 circle (connects-to (out 2) 3)
              (N-connects-to 1 3)))

(defrule (basic-p-block flowchart-grammar)
    (0 P-block (setf (in 0) 1
                     (out 0) 1))
    (1 rectangle))
```

A graphical depiction of the is shown in Figure 2. A visual indication that these relations do not hold of composite sets directly can be seen as the arcs (representing relations) cross the enclosing perimeters of nonterminal objects. All relations in this example are taken as constraints on individual members of the input set. Consider, for example, the relational constraint (connects-to 1 (in 2)) appearing in rule flowchart. The first argument, 1, is a direct reference to a terminal object with lexical type oval. The second argument, (in 2), is an indirect reference to the value of the in attribute of an object of (nonterminal) type P-block. This value will be bound to a terminal object during parsing. We call the set of attributes expander attributes that appear in any of the arguments to expander relations in a grammar. In this grammar, *in* and *out* are the expander attributes.

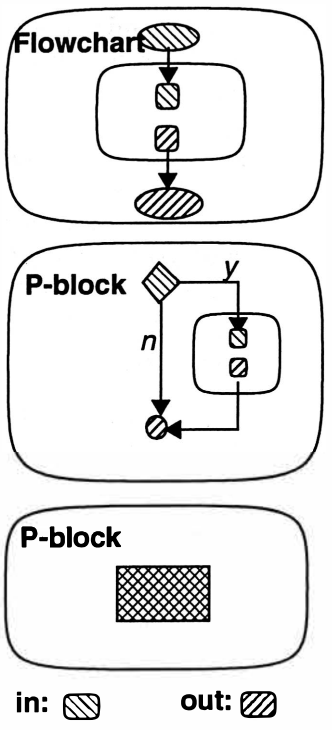

Figure 2: Graphical views of flowchart productions.

The rules must percolate references to individual members of the input as feature values from the right-hand-side of each rule to its left-hand-side. In the pseudo-unification formalism, assignments are made through setf forms. (We will also use forms in the text such as $(\mathrm{attr}_x\ 0) = (\mathrm{attr}_y\ i)$ to represent feature percolation. They are intended to be operationally equivalent to the setf forms.) Unlike general attribute passing, we assume equality as the only relation between values in feature percolation.[2] In the rule basic-p-block, one can see how the values of features in and out are linked directly to terminal input, in this case, an individual input object of lexical category rectangle.

We propose the following restriction for Atomic Relational Grammar productions, whose utility will become most evident when we consider predictive parsing later in this paper:

Restriction 2. Each production must percolate a value for every expander attribute used in the grammar.

In the grammar of Example 2 we can see that this condition is met since *in* and *out* are the only expander attributes used in the grammar and every production associates the value of each of these attributes in its left-hand-side with some value on its right-hand-side.

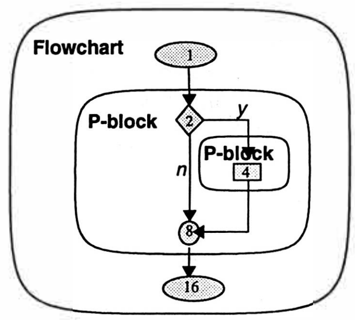

Figure 3: A derivation.

A derivation tree is shown in Figure 3. The input set is indicated as filled shapes indexed by integers representing binary numbers. Relations between input elements are graphed as arcs. The derivation tree (omitting ordering of daughter elements) is shown through the convention of spatial enclosure with dominating nonterminals represented as enclosing rounded rectangles.

3 Earley-style Parsing for Partial Orderings

Wittenburg (1992a) proposes a subclass of Atomic Relational Grammars amenable to Earley-style parsing. The class is called Fringe Relational Grammars (FRGs), where fringes are defined to be the minimal and maximal elements of Relational language expressions. The definition of Fringe Relational Grammars guarantees that any expression generable by the grammar

[2]This restriction to equality is significant only for expander attributes in percolation statements. The grammar formalism supports the use of additional features whose values may be tied to arbitrary computation using other features.

can be (partially) ordered. The motivation for such a requirement is that if we can partially order the input, then starting positions can be defined as the minimal elements and the parser can be given a partial order for scanning the input.

This is a summary of the restrictions on Atomic Relational Grammars that define the Fringe Relational Grammar subclass.

Restriction 3. Each relation used in an expander constraint in the grammar must independently be a partial order.

Restriction 4. For each expander relation used in the grammar, a pair of minimal/maximal expander attributes must be declared and every production must percolate values to these attributes in a manner that retains the partial orderings.

Restriction 5. Each production in the grammar must have an ordering variant of its right-hand-side such that, for each expander attribute, the right-hand-side element percolating the value to that expander attribute appears first.

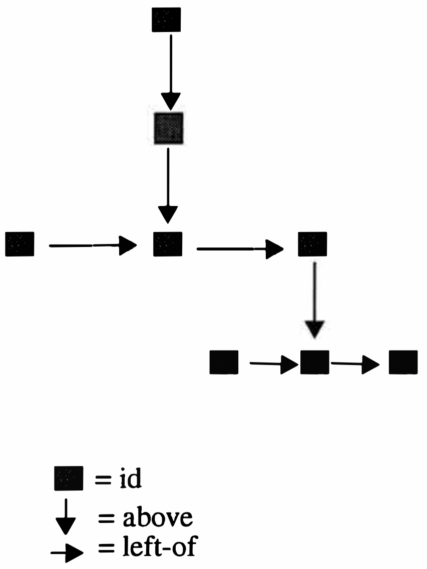

Figure 4. An expression in L(FRG-gram)

The rule set in Example 3 for the grammar we'll call FRG-gram generates Relational language expressions such as that graphed in Figure 4.

Example 3: FRG-gram rule set.

```
(defrule (S-rule FRG-gram)
    (0 S (setf (above-min 0)(above-min 1)
               (above-max 0)(above-max 1)
               (left-of-min 0)(left-of-min 1)
               (left-of-max 0)(left-of-max 1)))
    (1 Subtree))

(defrule (Subtree-rule FRG-gram)
    (0 Subtree (setf (above-min 0) 1
                     (above-max 0) (above-max 2)
                     (left-of-min 0) 1
                     (left-of-max 0) 1))
    (1 id)
    (2 Row)
    :expanders
     (above 1 (above-min 2))))

(defrule (Row-rule FRG-gram)
    (0 Row (setf (above-min 0)(above-min 2)
                 (above-max 0) (above-max 2)
                 (left-of-min 0) (left-of-min 1)
                 (left-of-max 0)(left-of-max 3)))
    (1 Subtree)
    (2 Subtree)
    (3 Subtree)
    :expanders
    (left-of (left-of-max 1)(left-of-min 2))
    (left-of (left-of-max 2)(left-of-max 3)))

(defrule (Basic-subtree FRG-gram)
    (0 Subtree (setf (above-min 0) 1
                     (above-max 0) 1
                     (left-of-min 0) 1
                     (left-of-max 0) 1))
    (1 id))
```

The *left-of* relation is used to compose horizontally aligned rows via topmost elements and *above* is used to vertically align mother elements with daughter rows via the daughter element at the row's center.

It is easy to see that both the *left-of* and *above* relations can be defined such that they independently will be partial orders on any input. This takes care of Restriction 3 for FRGs. As for the Restriction 4, note how each production sets the value of all four expander attributes *above-min*, *above-max*, *left-of-min*, and *left-of-max* in its left-hand-side category. The linkings that are defined in all cases are consistent with the partial orderings that the relations induce on the right-hand-sides of each rule. In order to meet Restriction 5, we must spawn ordering variants for some of the rules. Table 1 summarizes such an expanded ruleset that can be generated automatically. Each ordering variant is indicated by rule name and right-hand-side sequence as defined in Example 3. From Table 1 we can see that three rule variants would need to be added over the orderings implicit in the base grammar of Example 3. Such a table is used on-line in the parsing algorithm in order to provide a mapping from expander attributes (used in prediction) to rule variants that percolate (or bind) that expander attribute first.

Table 1: Expanded FRG-gram ruleset

Rule	**Expander Attribute**	**RHS Ordering**
S-rule	*all*	<1>
Subtree-rule	above-min	<1, 2>
	above-max	<2, 1>
	left-of-min	<1, 2>
	left-of-max	<1, 2>
Row-rule	above-min	<2, 1, 3>
	above-max	<2, 1, 3>
	left-of-min	<1, 2, 3>
	left-of-max	<3, 2, 1>
Basic-subtree	*all*	<1>

The intuitive idea of the parsing algorithm is as follows. The parser is initialized in Earley style with dotted productions expanding root symbols of the grammar at each of the minimal elements of the input. Extended predict, scan, and complete steps carry over from Earley's algorithm, so the parser will build up item sets through enumerations of the nodes of the input relation graph originating at the minimal nodes. Note that no matter where the parser starts in the input, a successful derivation starting from that position will have to visit all the nodes of the input graph at least once. The main goal in designing the algorithm is to minimize the number of enumerations while still ensuring completeness. When two or more enumeration sequences converge, the algorithm is able to detect when a prediction has been made before so that the same prediction (and all items that ensue) don't have to be re-created. In order to be able to absorb the products of a previous prediction into a converging enumeration sequence, a fourth step has to be added to Earley's predict, scan, and complete. It is called inverse-complete. Where complete is given an inactive state and asked to extend active states that end where the inactive state starts, inverse-complete is given an active state and tries to extend it with any inactive states that start where the active state ends.

As suggested in the concluding remarks of Wittenburg (1992a), there is still research to be done to minimize enumeration sequences further. One strategy is look for ways of reducing the number of requisite start points. While we address a different problem in the remainder of this paper, the solution is relevant to this question since the parser need only initialize its search at a single position.

4 Preliminaries to a Parser

Here we present the preliminaries necessary to define an Earley-style algorithm for Atomic Relational Grammars that can be initialized from any start element. As is discussed in the concluding remarks, the parsing algorithm solves only part of the problem for defining incremental, predictive parsing for Relational Grammars, but it is of interest in its own right. We begin with the following observations.

The existence of an Earley predictive state (an active state that covers no input) in a parse table implies that a derivation headed by the nonterminal on the left-hand-side of the dotted rule may "begin" at that positional index. A scanning action is valid, given some positional index, only if the terminal symbol at that position is a valid left-corner of a possible derivation subtree predicted to start at that index. For a parser to adquately predict top-down expansions of the grammar's root symbol from any position in the input and only that position, it follows that every element of any parsable input sets must be a possible left-corner of a valid derivation tree headed by the root symbol. As a condition on a grammar, this implies that for every nonterminal, there must be variants of any production expanding that nonterminal such that any member of its terminal yield can be ordered first in the production's right-hand-side.

Part of our proposal then is to require ordering variants of the right-hand-sides of every production so that every rhs-element appears first in at least one variant. It is not hard to see that such a mechanism carried through will satisfy the any-start requirement. Intuitively, if for any local subtree in a derivation admitted by the base grammar, there exists another derivation subtree reordered such that an arbitrary rhs-element appears first, then, by induction, any derivation tree can be transformed by a series of local reorderings such that an arbitrary terminal node appears as a left-corner in at least one (other) derivation tree.

Then there is the question of remaining ordering variations once a parser has chosen a start position. Note here that it is *not* a requirement that once the parser has chosen its first element, that the next choice for scanning may arbitrarily be any of the remaining input elements. A natural requisite for ordering variants once the algorithm has been initialized at a start position is to allow for any variations of input elements that are connected through some relation to the input already scanned. Such an approach would be consistent with the parsing algorithms discussed previously since queries formed from relational constraints could serve as the basis for expanding rule matches. To fully realize even this set of ordering variations would require extensions beyond the scope of the present paper, however. Here we concentrate on the problem of an arbitrary starting point and choose to direct the parser to scan remaining input nondeterministically in orders (not necessarily all orders) consistent with the connectedness constraint just mentioned. A simple extension allows for all connected ordering variations within local rules.

In Wittenburg (1992a), the predict subroutine makes use of the function F-permute to find all candidate rules for prediction. This function maps from an expander attribute and a predicted nonterminal to rule variants appropriately ordered that can expand that nonterminal. Appropriately ordered here implies that the production can provide a possible percolation path such that, as the left-branch of an eventual derivation subtree bottoms out, terminal elements scanned at that position can ground the expander attribute used in prediction. Such a mapping is extracted from what was called an F-permute table, as exemplified in Table 1. To get an intuitive grasp of this relationship of percolation paths and prediction, which will be carried over here, consider again the input in Figure 3 with the grammar in Example 2. Assume that an Earley-style parser has scanned the topmost oval, indexed as 1 in the figure. This would imply the existence of an item that incorporated a dotted production of rule Flowchart, repeated here.

```
[flowchart-1: Flow -> start . P-block end]

    (0 Flowchart (setf (in 0) 1
                       (out 0) 3)
    (1 oval)
    (2 P-block (connects-to 1 (in 2)))
    (3 oval (connects-to (out 2) 3))
```

Note now that the expander constraint (connects-to 1 (in 2)) is of relevance in predicting the next input to be scanned. Given just this item, only those input elements that are candidates to bind the in attribute of a P-block constituent need be considered. In particular, we need not consider ordering variants of P-block rules in which the initial right-hand-side element cannot serve to bind this attribute. These observations lead us to the following definition.

Definition 2. A triple $< N, \text{attr}, P >$, N a nonterminal category, 'attr' an expander attribute, and P an ordered production of Atomic Relational Grammar $\mathcal{G}$ is in the starts-by-binding relation iff the left-hand-side category of $P = N$ and there exists a feature assignment statement of the form (attr 0) = 1 or (attr 0) = (attr′ 1) in P.

The starts-by-binding relation forms the basis for precompiling a Prediction Table. A rule variant in the table indexed by some nonterminal category X and some expander attribute attr implies that the production expands nonterminal X and that there is an assignment of the form (attr 0) $= \ldots 1 \ldots$ in the feature percolations of that production, i.e., the cover of the first right-hand-side element grounds attr if it is a terminal or it carries an attribute whose value is linked to attr if it is a nonterminal.

The derivation of a table representing this relation is in two steps. We first generate rule ordering variants. Here we assume an algorithm that, for each production, forms one ordering variant per right-hand-side element such that the right-hand-side element is ordered first. Ordering the remaining elements is done arbitrarily subject to the connectedness condition in Restriction 1.[3] Step two generates a table representing the starts-by-binding relation in the grammar and, in addition, we include in the table the special "attribute" start, whose entry includes a set of rule variants such that for each production expanding that nonterminal category, there is at least one variant ordering each right-hand-side element first, regardless of feature percolations.

Example 4 shows the Atomic Relational Grammar appearing previously in Example 2 with the addition of named ordering variants for non-unary productions.

Example 4: Extended Flowchart grammar.

```
(defbaserule (flowchart flowchart-grammar)
    (0 Flowcart (setf (in 0) (in 1)
                      (out 0) (out 3)))
    (1 start)
    (2 P-block)
    (3 end)
    :expanders
    (connects-to (out 1) (in 2))
       (connects-to (out 2) (in 3)))

    Ordering variants:
    flowchart-1 <1, 2, 3>
    flowchart-2 <3, 2, 1>
    flowchart-3 <2, 1, 3>

(defbaserule (conditional flowchart-grammar)
    (0 P-block (setf (in 0) (in 1)
                     (out 0) (out 3)))
    (1 decision)
    (2 P-block )
    (3 junction)
    :expanders
    (Y-connects-to (out 1) (in 2))
    (connects-to (out 2) (in 3))
    :predicates
    (N-connects-to (out 1) (in 3)))

    Ordering variants:
    conditional-1 <1, 2, 3>
    conditional-2 <3, 2, 1>
    conditional-3 <2, 1, 3>

(defrule (basic-p-block flowchart-grammar)
     (0 P-block (setf (in 0) (in 1)
                      (out 0) (out 1)))
     (1 procedure))
```

Table 2: Flowchart-grammar Prediction Table

Nonterm	Expander Attribute	Production Variant
Flow	in	flowchart-1
	out	flowchart-2
	start	flowchart-1 flowchart-2 flowchart-3
P-block	in	conditional-1 basic-p-block
	out	conditional-2 basic-p-block
	start	conditional-1 conditional-2 conditional-3 basic-p-block

Table 2 shows a Prediction Table for the grammar in Example 4. We assume the existence of a function starts-by-binding(N, A) that for some Atomic Relational Grammar $\mathcal{G}$ returns the set of production variants in the Prediction Table at nonterminal N and attribute A.

Definition 3. The first-attrs of an ordered production p in Atomic Relational Grammar $\mathcal{G}$ is $\{\text{attr}_x\}|$ there exists an assignment statement $(\text{attr}_y\ 0) = (\ \text{attr}_x\ 1)$ in p where $\text{attr}_{x,y}$ are expander attributes.

For example, the first-attrs of ordered production conditional-1 is the set {in}. The first-attrs of ordered production basic-p-block is the set {in out}. The first-attrs of a production are those expander attributes associated with the first rhs element of a production that provide bindings for expander attributes in the left-hand-side. They are used in the recursive predict step of the Earley-style algorithm to follow.

5 Earley-style Recognition for ARGs

We now turn to an Earley-style algorithm for the full class of Atomic Relational Grammars in

[3]If a scanning algorithm is desired that will allow for all orderings consistent with Restriction 1, then more variants can simply be produced here.

which the starting position is arbitrary. We will summarize the essential points here, and confine our attention to a recognition algorithm.

Definition 4. A multidimensional multiset (md-set) is an n-tuple $(I, R_1 \dots R_n)$ such that $R_1 \dots R_n$ are binary relations on the multiset I.

Even though $R_1 \dots R_n$ may in general be n-ary, expander relations must be binary and that is all we will concern ourselves with here.

Definition 5. An indexed md-set $\mathcal{C}$ is an md-set $(I, R_1 \dots R_n)$ and a one-to-one and onto function from the set of integers $1 \dots |I|$ to members of I.

Now we define the states, or items, used in a parse table given an indexed md-set $\mathcal{C}$. Inactive states are representative of completely matched rules and thus they include a category as well as the feature values assocated with the rule's left-hand-side.

Definition 6. Inactive states relative to an indexed md-set $\mathcal{C}$ are a triple $[cat, f, c]$ where cat is a nonterminal or terminal symbol, f is a vector of features (especially, expander attributes with values in $\mathcal{C}$), and c is a logical bit vector representing a subset of $\mathcal{C}$ (the state's terminal yield, or cover).

Inactive states will be indexed in the parse table by every binding of an expander attribute in f (all of which must be individuals in $\mathcal{C}$). Intuitively, we consider inactive states to begin as well as end at every terminal that is bound by an expander attribute. However, this parser doesn't make the distinction between beginning and ending so we need only a single indexing array.

We next turn to active states, which represent partially matched or unmatched rules. We assume the Earley algorithm convention of a set of dotted productions, with the dot representing a position in the ordered right-hand-side elements.

Definition 7. Active states are a triple $[p, i, (d_1 \dots d_n)]$ where p is a production; i, the Earley positional dot, is an integer ranging from 1 to the length of the right-hand-side of p representing the next daughter to match; and $(d_1 \dots d_n)$ is an ordered list of pointers to inactive states of daughters matched so far.

The cover, or terminal yield, of an active state is derivable from the covers of the inactive states that have already been matched.

Definition 8. The cover of an active state $[p, i, (d_1 \dots d_n)]$ is defined to be the union of the covers of the inactive states $(d_1 \dots d_n)$.

As with inactive states, active states are indexed by individual members of $\mathcal{C}$. The intuition is that active states are indexed by individuals in the input that are candidates to be used in the next advancement of that active state. For any daughters but the first, one can make use of the expander constraint at that positional dot to find such candidates. For active states that have not yet matched any daughters, their input indices are derived from higher predictions, initialized with the input element at the starting position.[4]

The following definition is useful in the recursive predict step of the Earley-style algorithm. It is necessary to pick out the expander attributes that can acquire bindings through matching the next daughter.

Definition 9. The to-be-bound-attrs of an active state $s = [p, i, (d_1 \dots d_n)]$ are defined to be first-attrs(p) if $i = 1$, else $\{\text{attr}_x\}$ where attr_x is the attribute of the to-be-bound argument of the expander constraint at position i.

Agenda items are defined next.

Definition 10. An agenda item is a pair [state, keys] where state is an active or inactive state and keys is a set of state indices (individuals in an indexed multidimensional set $\mathcal{C}$).

[4]In Wittenburg et al. (1991), active states were indexed by the already-bound argument. The change is necessary to make indexing of predictive states (which have no cover, and thus no bound argument) consistent with the indexing of other active states.

Agenda items represent states and their indices relative to some input. As in chart parsing, the agenda will hold a list of items to be potentially added to the parse table. There is flexibility in its management. Here we assume a FIFO queue.

Procedure 1. `Advance(a-state, i-state, a-index)`

Input: An active-state a-state $= [p, i, (d_1, \ldots d_n)]$, and an inactive-state i-state $=$ [cat, f, c].[5]

Output: A list of agenda items, possibly null, that are the result of advancing `a-state` with `i-state`.

Method:

Case 1: If i = length of rhs of p (new state will be inactive), then let c' = union-covers$((d_1 \ldots d_n), c)$ and let f' = percolate$(p, (d_1 \ldots d_n), c, f)$. Return an agenda item $[\text{i} - \text{state}' = [\text{cat}', f', c'], \text{keys}]$ where cat$'$ = lhs of p and keys is the list of inactive state indices of i-state$'$.

Case 2: (New state will be active.) Let a-state$' = [p, i+1, c' = (d_1, \ldots d_n, c)]$. Let e = expander-at-position$(p, i+1)$. Let q = query(expander, c', Q). If q is non-null, return an agenda item $[\text{a} - \text{state}', q]$.

The advance procedure is called by scan, complete, and inverse-complete, to be defined shortly. As we have pointed out, active states are indexed at an element in the terminal yield of any potential next daughters to be matched (rather than the last one to have been matched), so they will be added to the parse table only if some tuple in the required relation can be shown to exist in the input. This question is satisfied by the query(expander, c', Q) form. Here we assume that given an expander constraint and the daughters matched so far, a subroutine can dereference the arguments to the expander constraint, binding the already-bound one, and then execute a query that will return the members of the input that can bind the to-be-bound argument.

Algorithm 1. Membership in L(ARG)

Input: An Atomic Relational Grammar = $\mathcal{G}$, an indexed multidimensional set $\mathcal{C}$, and a starting element q that is an arbitrary member of $\mathcal{C}$.

Output: A parse table of state sets S_j.

Auxiliary data structures:

Agenda: a FIFO list of states to process, initially null.

Init-states: the set of starting predictive states, initialized as follows: For every p in starts-by-binding(S, start), add a state $[p, 1, \text{null}]$ to init-states. For every state $s = [p', 1, \text{null}]$ in init-states, if the rhs symbol X at rhs position 1 of p' is a nonterminal, then let init-states = union(init-states, starts-by-binding(X, start)).

Parse table: a hash table of state sets S_j, where j is an index to individuals in $\mathcal{C}$.

Method: We initialize as follows:

- For every s in init-states, add an agenda item $[s, \{q\}]$ to the Agenda.
- Until Agenda is empty, do:
  - Remove one item = [state, keys] from Agenda.
  - For k in keys, if an equivalent state is not already at S_k, add state at S_k. Then do:

    Scanner: If state $= [p, i, (d_1 \ldots d_n)]$ at k is active and the rhs symbol x at position i of p is terminal, if the terminal symbol of input item $= y$ at k matches x and k does not intersect cover(state), then add any item in advance(state, y, k) to Agenda.

[5]A variant of this procedure must also accommodate input items from C directly in place of the inactive-state argument when called by the scan procedure. It is straightforward to form a (transient) inactive state from an input item.

Predictor: If state = $[p, i, (d_1 \ldots d_n)]$ is active and the rhs symbol X at position i of p is a nonterminal, then for every attribute a in to-be-bound-attrs(state), for every production p' in starts-by-binding(X, a), add item $[\text{state}' = [p', 1, \text{null}], k]$ to Agenda.

Completer: If state = $[\text{cat}, f, c]$ is inactive, then for every a-state = $[p, i, (d_1 \ldots d_n)]$ in union$(S_k, \text{init} - \text{states})$, if cat matches the rhs symbol at position i of p, the intersection of c and cover(a-state) is null, and k satisfies the expander constraint at position i of p, then add any item in advance(a-state, state, k) to Agenda.

Inverse-completer: If state $[p, i, (d_1 \ldots d_n)]$ is active, then for every i-state = $[cat, f, c']$ at S_k, if cat matches the symbol at position i of p, the intersection of cover(state) and c' is null, and k satisfies the expander constraint at position i of p, then add any item in advance(a-state, state, k) to Agenda.

- If there is an inactive state of the form $[X, f, u]$ in the parse table such that X = root-category of $\mathcal{G}$ and $u = \mathcal{C}$, then succeed. Else fail.

6 Parse trace

The following trace of a parsing run uses the grammar from Example 4 together with the input shown in Figure 3. This trace picks the rectangle, indexed as 4, as the start element. Each step of the trace represents a state added to the table and includes the following information:

- <rule> or <category / feature vector>,
- <cover>, <indices>, and
- <source>.

Active and predictive states show the rule-name and dotted production in the form [<rule-name>: <dotted-rule>]. Inactive state categories and feature vectors are shown as #S(<cat> :<feat-1> <val-1> ... :<feat-n> <val-n>). The <cover> is shown as an integer representing a logical bit vector. For example, 15 represents the logical bit vector 1111, which in turn is the union of items indexed with integers 1, 2, 4, and 8. <Indices> is a list of input covers by which the inactive or active state is indexed in the parse table. The <source> information explains which subroutine produced the state.

1. [flowchart-2: Flow -> . P-block start end], 0, (4), init.
2. [flowchart-3: Flow -> . end P-block start], 0, (4), init.
3. [flowchart-1: Flow -> . start P-block end], 0, (4), init.
4. [conditional-1: P-block -> . decision P-block junction], 0, (4), init.
5. [conditional-3: P-block -> . junction P-block decision], 0, (4), init.
6. [conditional-2: P-block -> . P-block decision junction], 0, (4), init.
7. [basic-p-block: P-block -> . procedure], 0, (4), init.
8. #S(P-block :in 4 :out 4), 4, (4), scan 7.
9. [conditional-2: P-block -> P-block . decision junction], 4, (2), complete 6 with 8.
10. [conditional-2: P-block -> P-block decision . junction], 6, (8), scan 9.
11. #S(P-block :in 2 :out 8), 14, (8 2), scan 10.
12. [flowchart-2: Flow -> P-block . start end], 14, (1), complete 1 with 11.
13. [flowchart-2: Flow -> P-block start . end], 15, (16), scan 12.
14. #S(Flow :in 1 :out 16), 31, (16 1), scan 13.

7 Conclusion

The algorithm presented here is the first predictive, Earley-style algorithm that we know of for the full class of Atomic Relational Grammars. This class of grammars appears to be widely useful and is easily implemented through unification-based approaches; more specialized implementations closer in spirit to attribute grammars are also afforded. The primary problem addressed here is allowing for initialization at an arbitrary starting position in the input. The solution to this problem should carry over to other predictive parsers for multidimensional grammars as, for example, extended LR algorithms (see Costagliola et al., 1991).

Although the Earley-style algorithm presented here is of interest in its own right, there are remaining issues in exploring incremental, predictive parsing of visual language interfaces. To carry out the goal of providing an analogue of command completion in visual language interfaces requires at least two extensions beyond the work reported on here. First, more variations in scanning order are likely to be desired than what can be provided for here. Note that with the current algorithm, the ordering variants relative to a single global scanning order are restricted to local permutations within rules. What may be desired is the multidimensional analogue of predictive parsing of free-word-order languages that can scramble not only within grammatical constituents but also across constituents (subject to the connectness constraint). Further, algorithms more akin to island-based parsing (from a single island out) are likely to be preferable for interface parsing than the Earley-style algorithm presented here. Note that by following all permitted scanning orders reachable from a given start position, the Earley-style algorithm expands multiple islands in parallel, each of which may cover only part of the input globally processed so far.

Finally, a few short remarks on related literature. In the theoretical graph grammar literature, there have been recent results suggesting a natural class for a useful and general class of graph grammars, namely, context-free hypergraph grammars of bounded degree (Englefreit, 1992). An interesting line of research would be to investigate the relationship between Atomic Relational Grammars and Hypergraph grammars. The role of features and percolation in Atomic Relational Grammars seems to be quite similar to hyperedges and hyperedge replacement in Hypergraph Grammars. Elsewhere in the graph grammar literature, an active chart parsing algorithm for flowgraphs has been proposed (Lutz, 1989; Wills 1990) that is related to the parsing algorithm discussed here and in Wittenburg et al. (1991). Again, the exact relationship between flowgraphs and Atomic Relational Languages is worthy of investigation.

The most closely related parsing algorithms from the visual language literature are to be found in Tomita (1990) and Costagliola — Chang (1991), both of which extend Earley-style parsing into multidimensional domains. Although there is commonality at the level of parsing subroutines, indexing methods differ substantively. These differences arise in part because of different assumptions regarding the nature of the input and the allowable relations. Both these other proposals assume that the input is held in a grid of some kind with elements of equal size. Relations are defined accordingly. There are no such assumptions here.

Acknowledgements

This research was carried out in the Bellcore Computer Graphics and Interactive Media research group, directed by Jim Hollan. Thanks to Steve Abney, Bernard Lang, David Weir, and Louis Weitzman for discussions related to the topic of this paper.

References

Costagliola, G. — S.K. Chang (1991) "Parsing 2-D Languages with Positional Grammars." In: *Proceedings IWPT-91, Second International Workshop on Parsing Technologies*, 13-15 February 1991. 235–243. Pittsburgh, Pennsylvania: Carnegie Mellon University, School of Computer Science.

Costagliola, G. — M. Tomita — S.K. Chang (1991) "A Generalized Parser for 2-D Languages." In: *1991 IEEE Workshop on Visual Languages* (Kobe, Japan). 98–104.

Crimi, C. — A. Guercio — G. Nota — G. Pacini — G. Tortora — M. Tucci (1991) "Relation Grammars and their Application to Multidimensional Languages." In: *Journal of Visual Languages and Computing* 2(4), 333 - 346.

Earley, J. (1970) "An Efficient Context-Free Parsing Algorithm." In: *Communications of the ACM 13*, 94 - 102.

Engelfreit, J. (1992) "A Greibach Normal Form for Context-free Graph Grammars." In: Kuich, W. (Ed): *Automata, Languages and Programming: 19th International Colloquium*, Wien, Austria, July 1991, Lecture Notes on Computer Science 623, 138 - 149. Springer-Verlag.

Ferrucci, F. - G. Pacini - G. Tortora - M. Tucci - G. Vitiello (1991) "Efficient Parsing of Multidimensional Structures." In: *1991 IEEE Workshop on Visual Languages* (Kobe, Japan). 104 - 110.

Flasinski, M. (1988) "Parsing of edNLC-Graph Grammars for Scene Analysis." In: *Pattern Recognition* 21, 623 - 629.

Flasinski, M. (1989) "Characteristics of edNLC-Graph Grammar for Syntactic Pattern Recognition." In: *Computer Vision, Graphics, and Image Processing* 47, 1 - 21.

Golin, E. J. (1991) "Parsing Visual Languages with Picture Layout Grammars." In: *Journal of Visual Languages and Computing* 2, 371 - 393.

Golin, E.J. — S.P. Reiss (1990) "The Specification of Visual Language Syntax." In: *Journal of Visual Languages and Computing* 1, 141 - 157.

Lutz, R. (1989) "Chart Parsing of Flowgraphs." In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 116 - 121.

Helm, R. — K. Marriott (1986) "Declarative Graphics." In: *Proceedings of the Third International Conference on Logic Programming*, Lecture Notes in Computer Science 225, 513 - 527. Springer-Verlag.

Helm, R. — K. Marriott (1990) "A Declarative Specification and Semantics for Visual Languages." In: *Journal of Visual Languages and Computing* 2(4), 311 - 331.

Tomita, M. (1990) "The Generalized LR Parser/Compiler V8-4: A Software Package for Practical NL Projects." In: *Proceedings of COLING-90*, Volume 1, 59 - 63.

Tomita, M. (1991) "Parsing 2-Dimensional Language." In: Tomita, M. (Ed): *Current Issues in Parsing Technology*, 277 - 289. Kluwer Academic.

Weitzman, L. — K. Wittenburg (1993) "Relational Grammars for Interactive Design." In: *Proceedings of 1993 IEEE/CS Symposium on Visual Languages*, August 24-27, Bergen, Norway.

Wills, L. (1990) "Automated Program Recognition: A Feasibility Demonstration." In: *Artificial Intelligence* 45, 113 - 171.

Wittenburg, K. — L. Weitzman — J. Talley (1991) "Unification-Based Grammars and Tabular Parsing for Graphical Languages." In: *Journal of Visual Languages and Computing* 2(4), 347 - 370.

Wittenburg, K. (1992a) "Earley-style Parsing for Relational Grammars." In: *Proceedings of the 1992 IEEE Workshop on Visual Languages*, Seattle, Washington, Sept 15-18, 1992. 192 - 199.

Wittenburg, K. (1992b) *The Relational Language System*. Bellcore Technical Memorandum TM-ARH-022353.

Wittenburg, K. (1993) "F-PATR: Functional Constraints for Unification-based Grammars." In: *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 216 – 223.