PARSING WITH RELATIONAL UNIFICATION GRAMMARS

Kent Wittenburg
Bellcore Visiting Researcher
Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, Texas 78759
Arpanet: Kent@mcc.com
Phone: (512)338-3626
Fax: (512)338-3600

ABSTRACT

In this paper we present a unification-based grammar formalism and parsing algorithm for the purposes of defining and processing non-concatenative languages. In order to encompass languages that are characterized by relations beyond simple string concatenation, we introduce relational constraints into a linguistically-based unification grammar formalism and extend bottom-up chart parsing methods. This work is currently being applied in the interpretation of hand-sketched mathematical expressions and structured flowcharts on notebook computers and interactive worksurfaces.

1. INTRODUCTION

In the MCC Interactive Work Surface Project, we have been applying a language perspective to the problem of connecting meaning to graphical and sketched media on both the input and the output side of human-machine interfaces. The technology is initially being applied to the problem of recognition of hand-sketched input through the "electronic paper" interface of notebook computers and worksurfaces (Avery 1988; Martin et al. 1990). Our first applications are interpreters for math expressions and structured flowcharts. Subsequent applications will include interpretation of sketched designs (e.g., engineering or architectural layouts or plats) in such a way that the semantic information can be made available for subsequent database update and querying, intelligent advising, creation of dynamic prototypes, etc. On the output side, we expect that the inverse connection of underlying data to a visual vocabulary will enable easy-to-use tools for connecting the semantics of underlying data to dynamic graphical displays.

Figure 1-1 shows a visualization of a derivation in two-dimensional space. Such derivations can be produced by grammars which describe languages whose sentences are objects situated in a two-dimensional space as long as the grammars can specify relational, in the most obvious case positional, constraints among the objects. Such constraints, and their resolution, are beyond the capacity of structurally-based unification grammars and parsing methods developed for languages of strings. The purpose of this paper is to present the framework of Relational Unification Grammar (RUG), which is capable of describing such languages, and then to extend bottom-up chart-parsing methods to work with these grammars. The algorithm presented here is motivated by the need to process input incrementally. We expect to derive benefit in our application domains from processing each symbol in the order in which it is created by a user. Such a parser allows for suggesting possible continuations for the user as well as determining correctness of the input so far. Temporal ordering imposes significant demands on our parser since we cannot enumerate the input based on spatial considerations, for example, a top-down left-to-right traversal. Such a normalization of ordering has usually been assumed in previous applications of grammar-based parsing to visual domains (e.g., Tomita 1989; Chang 1988).

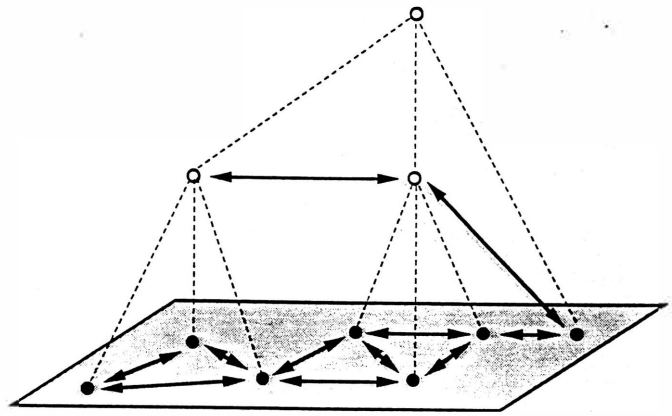


Figure 1-1: Derivation in 2D space

2. GRAMMARS

Parallel to Helm and Marriott (1990), who are investigating visual languages in the logic programming tradition, we adopt a unification-based grammar formalism that is augmented with constraints necessary to incorporate relations beyond string concatenation into the declarative specification of a language. Unification itself then must be expanded to incorporate some form of constraint solving, an area of active research in logic programming.

Our approach is to extend the family of PATR unification-based grammar formalisms (Shieber 1986, 1989). Instead of strings, we assume the terminals of our grammar to be what we will call icons, objects that are associated with a set of attributes such as <X,Y> coordinates, extent, and color, each of whose value ranges is finite. The rules of the grammar, besides specifying a variety of syntactic and semantic constraints for use in deriving sentences of the language, also specify relational constraints among icons that may require arbitrary computation to determine satisfaction.

Although we will not attempt to give a rigorous definition of the unification basis of our grammars here, it will nevertheless be useful to note properties of some of the attributes, values, and relational constraints appearing in the grammar. Besides the customary PATR grammar machinery consisting of a vocabulary of attribute labels L and constant values C, lexical and nonlexical productions P, and a start category (see Shieber 1989), a relational unification grammar RUG is distinguished by the tuple (N,Σ,I), where N is a finite set of (nonterminal) icon type symbols, Σ is a finite set of (terminal) icon type symbols, I is an infinite set of spatially located icons each of which has a type $\in N \cup \Sigma$, and R is a finite set of relations in I.

The rules of the grammar contain the following elements, whose left-hand sides we will consider unordered for the time being.

$$\text{Head } \text{Arg}_1 \ \ldots \ \text{Arg}_n \rightarrow \text{Result}$$

Since we are focusing on analysis here rather than generation, the arrow in the rule skeleton is interpreted as "reduces to" rather than "rewrites as". Each rule must have a head and a result, and there may be zero or more arguments. Although there is nothing essential from a formal point of view about our use of the functional terms *head*, *argument*, and *result* in rules, it is a convention to guide grammar construction that we find perspicuous.

Each of the elements of the production has at least the following structural constraints:

$$\text{syntax} \in N \cup \Sigma$$
$$\text{icon} \in I$$

The *syntax* attribute must take as value elements from the set $N \cup \Sigma$. Although in fact we allow for syntactic characterizations to be arbitrarily complex, we need a designated feature somewhere in the structure to be able to instantiate and refer to the types of icons associated with both terminal and nonterminal symbols. Here we will use the *syntax* attribute for this purpose. In addition, each of these rule elements has an *icon* attribute whose values are taken from I. In practice, the *icon* value may be a unique name for an icon instance.

Additionally, every rule that is not unary is required to have a set of relational constraints with certain additional conditions. Let us turn to an example in order to clarify the use of relational constraints. Following is an example of a rule from the domain of mathematics expressions. It is a rule that forms vertical infix expressions such as fractions, assigning an appropriate semantics for evaluation of the expression. The syntactic and semantic structural constraints appear first followed by the relational constraints.

Rule 1 Vertical infixation:

$$\text{Head} \quad \text{Arg}_1 \quad \text{Arg}_2 \quad \rightarrow \quad \text{Result}$$

$$\langle \text{Head icon} \rangle = X$$
$$\langle \text{Arg}_1 \text{ icon} \rangle = Y$$
$$\langle \text{Arg}_2 \text{ icon} \rangle = Z$$
$$\langle \text{Head syntax} \rangle = \text{Vert-infix-op}$$
$$\langle \text{Arg}_1 \text{ syntax} \rangle = \text{Formula}$$
$$\langle \text{Arg}_2 \text{ syntax} \rangle = \text{Formula}$$
$$\langle \text{Result syntax} \rangle = \text{Formula}$$
$$\langle \text{Head sem} \rangle = \langle \text{Result sem pred} \rangle$$
$$\langle \text{Arg}_1 \text{ sem} \rangle = \langle \text{Result sem arg1} \rangle$$
$$\langle \text{Arg}_2 \text{ sem} \rangle = \langle \text{Result sem arg2} \rangle$$

$$\langle \text{Result icon} \rangle = \text{composition}(X\ Y\ Z)$$
$$\text{above}(Y\ X)$$
$$\text{below}(Z\ X)$$
$$\text{wider-than}(X\ Y)$$
$$\text{wider-than}(X\ Z)$$

The first of the relational constraints, which involves *composition*, defines the icon of the rule mother as a function of the icons of the rule daughters. In practice, this relation may involve summation of the bounding boxes of the daughter icons for domains such as math expressions, or concatenation of line segments in diagram domains. The other relations impose positional and size constraints on the icons involved in the production. Suitable definitions of *above* and *below* in the math domain incorporate adjacency as well as position. The particulars of such relations will differ across grammars and domains.

In anticipation of the bottom-up parsing algorithm which we will present shortly, there is an additional requirement which we will impose on the form of grammar productions and their relational constraints. Given that there are no positional constraints implied by the rule skeletons, it is useful for the parser to be driven by appropriate relational constraints from individual rules for its basic rule matching operations. Thus we distinguish the class of relations that drive the matching action of the parser from those that operationally serve as constraints on proposed matches. Positional constraints such as *above* are more appropriate for driving parsing than size constraints such as *wider-than*. We will assume that each grammar distinguishes a class of positional constraints for this purpose.[1] Furthermore, we will refer to the maximal relational domain (R domain) over which positional relations hold. Many grammars will restrict their positional constraints to adjacent elements--in this case, adacency defines the R domain for that grammar.

Our requirement on rule wellformedness is that there be some ordering of the daughter elements in productions as follows:

> **Condition 1:** An ordering of rule daughter elements is well-formed iff for every element but the first, a positional constraint exists between that element and an element appearing earlier in the ordering.

An intuitive understanding of the reason for Condition 1 can be reached by considering the ordering <Arg_1, Arg_2, Head>, corresponding to the order <numerator, denominator, divide-line> in a fraction expression, from Rule 1. Suppose the parser has matched the numerator element and is in the position of seeking candidates for its next match, the denominator element. Since there are no positional constraints in the rule that involve the icon associated with the instantiated numerator, the parser has no way to constrain the candidates for its next match. One would of course like to confine the search to only those objects which meet appropriate positional constraints from the grammar. On the face of it, the parser would have to consider every icon in the space as a possible instantiation of the denominator term in our example and could not rule any of these branches out on the basis of relational constraints until the divide-line had been matched.

[1]It will also simplify our exposition slightly if we assume that the icon variable for every argument appears as the domain term of at least one positional constraint. For this reason, we use both *above* and *below* in Rule 1, even though the same constraints could be stated with just one of these relations.

Fortunately, this condition on the form of rules can be determined off-line, and we assume that a particular ordering of the elements of a rule is prespecified that meets this condition. For the purposes of this paper, we will assume that daughter elements of rules are to be matched in the order in which they are given in the rule definitions, implying that rules will be matched head-first.[2]

As a final note, lexical productions are defined traditionally as in PATR grammars with the difference that instead of strings, the terminal vocabulary is taken from the set of icon type symbols Σ. That is, lexical entries are pairs of the form $\langle\sigma, \Phi\rangle$, where σ is a member of the set of terminal icon symbols Σ and Φ is an RUG formula containing structural attributes found in the individual elements of rules. For example, here is a possible lexical entry for a line representing division:

```
horizontal-line:

    <syntax> = vert-infix-op
    <icon> = X
    <sem> = divide
```

Note the that *icon* attribute is uninstantiated in the lexicon. It will be instantiated with an actual icon instance (or a reference to one) during lexical lookup.

An example of a simple grammar may be found in Section 4, where we show a parse trace.

3. PARSING

In this section we give an account of a data-driven, tabular parsing algorithm that uses the grammar formalism described above. The algorithm we describe is technically a recognition algorithm, though it is easy to extend it to a parsing algorithm through the standard methods available in the literature (Aho and Ullman 1972). Tabular parsing methods (e.g., Earley 1970) and closely related chart parsing methods (Kaplan 1973; Kay 1980) have in common the use of a grammar table (or chart) that stores all complete and partially matched constituents, indexing them

[2]Although the parsing algorithm we discuss matches the elements of rules deterministically, algorithms such as Satta and Stock's (Satta and Stock 1989), which match rules in variable orders starting with the head, could be adapted to these grammars if any and all orderings meet Condition 1. One would, however, have to add the overhead necessary to check for the possibility of achieving the same rule match in more than one order.

to spans of the input string. The tables are used both to merge equivalent constituents over the same input into a single entry, thus avoiding combinatorics, and also to propose candidates for rule applications, given that adjacent entries in the tables are tied directly to adjacent substrings in the input.

Two approaches have been employed previously to apply tabular parsers in visual language domains. The first is to convert visual input data into a one-dimensional string form and use conventional string-based parsing methods. According to Fu (1974), the linear-conversion approaches have "not been very effective in describing two- or three-dimensional patterns". The second approach is to extend conventional parsing tables to directly represent regions over a spatial domain rather than spans over an input string. Tomita (1989) has extended the Earley algorithm and his own LR methods in this manner. Such an option ties a parser to the particulars of the spatial concatenation operations allowed in the grammar since the makeup of the table itself will be affected by the set of relations permitted in the visual space.

In contrast, our approach is to redistribute the functions expected from a parsing table across two modules. One, discussed in detail here, incorporates the parsing table and its constituent entries. From these data structures one can determine the input which a constituent dominates in order to check for equivalent table entries and successful output; however, one cannot from these structures alone determine the candidates for extending constituent coverage through rule applications. The module called the spatial relations analyzer, which keeps its own set of data structures, is necessary to discover new icon candidates for incorporating into rule applications. We hope that this overall conceptual design will become clear in the descriptions and examples which follow.

We assume an unordered set of spatially located icons as input to the parser. The correctness and completeness of the algorithm will not be affected by any temporal ordering of the input, but, for that reason, we can process the icons incrementally, in the order in which they appear through the interface.

> **Definition 1:** A *cover*, defined with respect to entries (partial or complete grammatical constituents) in the parse table, is the subset of icons in the input set that an entry dominates.

Covers are necessary for determining equivalence of constituents and success for the parse. The goal of parsing will be to produce any and all consitutents covering the initial input set that are labeled with the start symbol of the grammar.

Note that a cover need not be contiguous in a temporally determined input sequence. However, contiguity of a cover in the two-dimensional space will be enforced to the extent that the grammar uses spatial relations that subsume adjacency.

> **Definition 2:** A *category* is defined to be a PATR formula that is either a (partially) instantiated production as defined in Section 2 or else a PATR formula with instantiated features *syntax* and *icon*.

Categories are (partial) instantiations of rules, rule results, or lexical categories. In the algorithm descriptions which follow, we will assume the convention of referring to relevant features of categories with the notation *[head, arg_1...arg_n, result]* in the case of partial rule instantiations and *[syntax, icon]* in the case of rule result or lexical instiantions. We will also refer to individual rule elements at times with the convention *[syntax, icon, rels]*, where rels is a sorting of all relational constraints in the rule that contain the element's icon in either the domain or range term.

> **Definition 3:** A *state* is defined to be a triple [category, next-arg, cover], where next-arg refers to an arg i...j of category, possibly empty.

States are the parser's representation of a constituent. States are said to be *active* if next-arg is nonempty, implying that the category is an incomplete constituent, or *inactive* if next-arg is empty, implying that the category is a complete constituent. Active states will have partial rule instantiations as categories; inactive states will have instantiations of rule results or lexical items.

> **Definition 4:** A *trigger*, defined with respect to active states, is any (instantiated) icon appearing in the range term of a positional constraint whose domain term is the next-arg's icon variable.

That is, consider an active state that fits the category schema

$$\text{Head} \ldots \text{Arg}_j \ldots \rightarrow \text{Result}$$

$$\begin{array}{l} \langle \text{Head icon} \rangle = \text{Icon}_1 \\ \langle \text{Arg}_j \text{ icon} \rangle = X \\ \quad --- \\ (\text{Rel}_1\ X\ \text{Icon}_1) \end{array}$$

and whose next arg is Arg_j. $Icon_1$ will be a trigger for this active state since it appears in the range term of a positional constraint together with the next-arg's icon as domain. Note that we do define triggers to be instantiated icon instances, not icon variables.

In some cases there may be more than one trigger icon defined for an active state. Consider an active state whose category matches the following schema

```
Head ... Arg_j ... Arg_k ... → Result
  <Head icon> = Icon_1
  <Arg_j icon> = Icon_2
  <Arg_k icon> = X
       ---
  (Rel_h X Icon_1)
  (Rel_i X Icon_2)
```

and whose next-arg is Arg_k. Both $Icon_1$ and $Icon_2$ are triggers for this state. In such a situation the parsing algorithm, which uses triggers to index active states in the parse table, needs only one trigger. We arbitrarily choose among them.

Before turning to the parsing algorithm itself, we need one final definition. Lexical lookup, which produces states with instantiated categories associated with incoming icons, is defined next.

Definition 5: The function Lex(ical lookup), from the set of icons to a set of pairs consisting of a state and its icon index, is defined as follows:

```
Lex(X)={(s=[category, nil, {X}], i=X) |
  category = a lexical entry indexed by
  icon-type(X) and whose <icon> is
  unified with X}
```

This function represents lexical lookup and state instantiation. Given an icon, it uses the icon's type symbol to consult the lexicon. With the set of categories the lexicon produces, it then initializes the data structures for placing inactive states onto the parse table. In so doing, it unifies the icon itself with the icon variable of the category. The cover of the category will be the unary set consisting of the icon again. The index is an icon which will be used to index the state in the parse table. In the algorithm presented here, all lexically instantiated states will be inactive--the index for inactive states will be the icon for which the state represents a complete constituent.

Algorithm 1 *Main loop*

Assume an input set of spatially located icons W and an agenda set A, initially empty. Develop a table T whose entries are state sets indexed by icons.

```
while A nonempty or there exist icons
remaining to be processed in W do:
 choose one of the two following actions:
    for some icon X in W do              (1)
        for each pair (s,i) in Lex(X) do
            add (s, i) to set A
    extract any pair                     (2)
      (s=[category, next-arg, cover], i)
        from the set A;
        insert s in table T_i;
        apply each of the following
        procedures, in any order, to s:
            propose(s)
            expand(s)
            complete(s)
end while;
if there exists an s =
[category, next-arg, cover] in T such that
cover = W,  next-arg = empty, and the label
of category = start,
   then succeed;
   else fail.
```

The basic algorithm chooses arbitrarily among two actions as long as there are remaining data to do either action. Action (1) processes a single arbitrary icon from the input set. It creates state-index pairs to be placed in set A--this set, in chart parsing, corresponds to an agenda of pending actions. Action (2) chooses an arbitrary state-index pair from the agenda. It inserts the state into the parse table and then generates more pairs for the agenda by applying the three procedures *propose*, *expand*, and *complete* in any order. The indices for states in the parse table are icons. As will become evident in the procedures that follow, the index for active states is a trigger icon for that state; for inactive states, it is the icon associated with the highest dominating nonterminal.

Procedure 1 *Propose*

```
If state s=[category, nil, cover] is inactive,
then for every production p in P
   such that the category of s
   unifies with head element of p,
     create a new pair
     (s'=[category', next-arg, cover'], index)
     as follows:
       if there are no arguments in p
          then category' ::= result of p
               next-arg ::= nil
               cover' ::= cover
               index ::= icon of category;
          else category' ::= p
               next-arg ::= arg_1 of p
               cover' ::= cover
               index ::= a trigger icon of s';
     add pair to A unless an equivalent pair
     already exists.
```

The *propose* procedure applies to inactive states. It proposes new constituents through trying to unify the category of an inactive state against the head terms of the rule set. Successful unifications will result in new states that will be active or inactive depending on whether the rule is unary or not. Active states have a next-arg pointing to the first argument of the rule to be matched; inactive states have a null next-arg. The index of a new state will be the icon associated with the

newly unified category if the state is inactive, or a trigger icon if the state is active.

The condition that new states are added only if there is not an equivalent state already in A is a necessary (but not sufficient) condition for keeping the algorithm polynomial. This is a familiar move for all On^3 bounded context-free parsing algorithms. We will not elaborate here on questions of computational complexity, but suffice it to say we assume a definition of equivalence of <state, index> pairs--they are equivalent if their covers and indices are equal and if their categories and advancement are equivalent. A parsing algorithm, rather than just a recognition algorithm such as the one we are discussing here, would need to keep track of these equivalent states in order to recover the full set of parse trees.[3]

The fact that this algorithm proposes new rules for matching only when head elements of rules are discovered is part of the formula for making this algorithm "head-driven". It would be possible to use the predictive power of the partially matched headed constituents to filter out useless argument constituents. In the basic data-driven algorithm we discuss here, however, we do not actually make use of such top-down predictive machinery.

Procedure 2 *Expand*

```
If state s=[category,next-arg,cover]
   with next-arg=[syntax,Y,rels] is active,
then for some trigger icon X in a relation
  (rel Y X) in rels,                          (1)
  for every icon Z in the space such that
  (rel Z X)=True,                             (2)
      then for every inactive state
      s'=[category', nil, cover'] indexed
      by Z,
      if category' unifies with next-arg      (3)
         then (advance s s'),
         add resulting pair (s'' i) to set A
         unless an equivalent state exists.
```

The *expand* procedure is used to advance an active state across its next argument by finding inactive states that match the constraints of that argument as specified in the partially instantiated rule. Finding the candidate inactive states is the crux of the matter. They must (a) be associated with icons that meet the relational constraints of the argument, and (b) have categories that unify with the structural constaints of the rule's next argument. We use the partially instantiated relational constraints, relying on our spatial relations module, as a means of finding the icons that meet the spatial requirements. This particular feature of the algorithm is necessary given that we are not relying on our parse table to provide us with, say, adjacent icons.

The procedure begins with a trigger icon for an active state. Recall Definition 4 for triggers; line (1) of the procedure essentially restates it. Given a trigger icon, line (2) looks in the physical space for any icons in the triggering relation. The ones it finds will then be candidates for the icon to be associated with the next-arg. The remaining steps find any inactive states associated with the icon in question and then check that the structural features of these states as well as any remaining relational constraints are consistent with the rule's requirements. (Both conditions are implied by the unification step in line (3).) Any states that satisfy these conditions will be combined with the orginal active state, producing a new state that covers more territory.

Procedure 3 *Complete*

```
If state
   s=[category=[syntax, Y], nil, cover]
   is inactive,
then for every icon X falling within the
   local R domain w.r.t. icon Y,          (1)
   for every active state
   s'=[category',next-arg,cover'] that is
   indexed by X as trigger,               (2)
     if next-arg of s' unifies with category,
          then (advance s s'),
          add resulting pair (s'' i) to set A
          unless an equivalent state exists.
```

The *complete* procedure is defined with respect to inactive states. The basic operation is to look for active states for which this new inactive state can serve as a next argument, and then advance any such active states with respect to the inactive state. It operates just like the *expand* procedure once the candidate states are found. The differences lie in how one finds candidate active states given an inactive state, rather than the reverse.

As is indicated in line (1), the procedure depends on a notion of locality in the space in order to find the initial set of icons that is used to begin the search. If the R domain were characterized by adjacency, the procedure would map over each of the icons that were adjacent to the icon associated with the new inactive state. We do not, however, rule out the possibility that the locality of spatial relations may be defined otherwise.

Line (2) then consults the parsing table to find active states indexed by the locally related icons. Recall that active states are indexed by trigger icons. Thus these states will be the ones which the original inactive state may combine with. Further steps are the same as in *expand*.

[3]The issue of equivalence and state merging is nontrivial for unification grammars. See Shieber (1985).

Procedure 4 *Advance*

```
Given active state s=[category,next-arg,cover]
  with next-arg=[syntax, Y,(rel X)]
  and inactive state
  s'=[category', nil, cover'],

create a state s'' with index i as follows:

 case1: if category has no further arguments,
  then create a pair
  (s''=[category'', nil, cover''], i)
  where category'' ::= result of category,
        cover'' ::= cover' Union cover.
        i ::= icon of category''.

 case2: if category has further arguments,
  then create a pair
  (s''=[category'', next-arg', cover''], i)
  where category'' ::= category,
        next-arg' ::= next-arg + 1,
        cover'' ::= cover' Union cover,
        i ::= icon trigger for s''.
```

Advance takes an active state s and an inactive state s' which has already been unified as the next-arg for s, and it returns a new state/index pair. The new state resulting from advancement will be either inactive or active, depending on whether the final argument of the active state has been matched or not. The creation of an inactive state, shown in case1, sets the new state's category to the result-category of the active state. Its index will be the icon newly formed from the composition relation that holds between the icon of the result and the icons of the rule daughters. The creation of active states involves an advancement of the next-arg pointer. These states are indexed by a trigger icon. In both cases, the cover for the new state will be the union of the covers of the original states.

4. EXAMPLE

Here we give a example of a grammar for simple fractions and a parse trace of the bottom-up algorithm described above. Rule 1 is repeated for convenience. The trace will refer to the rules and lexical entries by number and omit the details of the internal rule elements. When nil appears in the next-arg position of a state, it is an indication that the category of the state corresponds to the instantiated result element of completed rules or the categories of lexical entries.

Rules

1 Vertical infixation:

Head Arg_1 Arg_2 → Result

<Head icon> = X
<Arg_1 icon> = Y
<Arg_2 icon> = Z
<Head syntax> = Vert-infix-op
<Arg_1 syntax> = Formula
<Arg_2 syntax> = Formula
<Result syntax> = Formula
<Head sem> = <Result sem pred>
<Arg_1 sem> = <Result sem arg1>
<Arg_2 sem> = <Result sem arg2>

<Result icon> = composition(X Y Z)
above(Y X)
below(Z X)
wider-than(X Y)
wider-than(X Z)

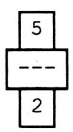
Lexicon

2 floating-point-no:
<syntax> = Formula
<icon> = X

<sem> = (numerical-value X)

3 horizontal-line:
<syntax> = vert-infix-op
<icon> = X
<sem> = divide

Let us assume the input to be the icons



arranged as shown. We will note them as <5>, <h-line>, and <2>, respectively, in the trace which follows. We have to pick an order for processing these input icons. Arbitrarily, we will process the icons top to bottom. We also will be faced with the choice between Actions 1 or 2 of *main loop*. Again, arbitrarily, we'll choose Action 2 (processing items in set A) over action 1 (processing another input icon) whenever there are are items in set A to process. Lastly, the algorithm gives us the freedom of ordering the items we choose from set A. We will process each of these items in the order in which they were put into A.

The algorithm will produce states in the order shown below:

1. s_1=[2,nil,{<5>}] is added at $T_{<5>}$ through Action 1 of main loop.
2. s_2=[3,nil,{<h-line>}] is added at $T_{<5>}$ through Action 1 of main loop.
3. s_3=[1,arg_1,{<h-line>}] is added at $T_{<h\text{-}line>}$ through procedure *propose*.
4. s_4=[1,arg_2,{<h-line>,<5>}] is added at $T_{<h\text{-}line>}$ through procedure *expand*, advancing s_3 with s_1.
5. s_5=[2,nil,{<2>}] is added at $T_{<2>}$ through Action 1 of main loop.
6. s_6=[1,nil,{<h-line>,<5>,<2>}] is added at $T_{<<5><h\text{-}line><2>>}$ through procedure *complete*, advancing s_4 with s_5.
7. The procedure halts with success, s_6 satisfying the conditions.

5. RELATED WORK

We first compare related work in grammar formalisms followed by related approaches to parsing visual languages.

Of other visual grammar frameworks we are aware of, our proposal differs in the following two respects:

1. The functional role of heads and arguments. Characteristic of the linguistic roots of our approach, we assign the functional roles of head and arguments to elements in the rule body. What motivates this move? First, we assume that these syntactic roles bear a close, if not one-to-one, relationship to predicates and arguments in the semantics. In our opinion such a commitment makes it easier to coordinate incremental syntactic and semantics processing important in the parsing of visual interface languages, and it also tends to produce grammars that have more meaningful and transparent syntactic and semantic constituents. We are not aware of any such committment in competing visual grammar approaches that do discuss semantics. Second, assuming that heads of phrases tend to offer constraints on the syntactic and semantic properties of their arguments, it becomes possible to take advantage of the pruning power of these constraints through the use of head-driven parsing and generation algorithms (Kay 1989; Satta and Stock 1989; Shieber et al. 1989).

2. The domain of spatial relations. As with Helm and Marriott (1990), our formalism allows the grammar to state any number of relational constraints among any elements within the domain of a single rule. While the formalism used by Anderson (1968) differs in several other respects, he too allows spatial relations to be stated over such a domain. Unlike Golin and Reiss (1989), we do not presume that it is possible to state constraints among elements arbitrarily distant in a derivation tree. Unlike the most recent grammars of the SIL-ICON system (Crimi et al. 1989), we do not confine the expression of spatial constraints to a single relation among pairs of elements that are adjacent in a rule body. In our opinion, most visual languages in practice, complex mathematics formulae among them, need the additional expressiveness of our formalism over the latter group of proposals.

As for parsing, the algorithm we have outlined is unique among visual language parsers, as far as we know, in allowing for maximally flexible enumeration. We have motivated this design feature in the context of our goal to provide parsing tools and help facilities for interface languages, where temporal ordering of the input cannot be assumed to match systematic spatial enumeration procedures.

The other distinguishing feature of the parsing algorithm is its disassociation of the parse table from any particular set of spatial relations used by the grammar. We take this to be a strength in that the algorithm is thus extremely general, although we concede that without exploring the spatial component more fully we cannot provide a complete solution to any particular visual language domain nor can we determine the computational complexity of our algorithm. The crux of our approach is to propose a particular form of indexing of the grammar table that makes use of icons and icon sets (covers). In future work, we will explore the complexity of this algorithm when paired with sets of assumptions regarding the spatial relations assumed by the grammar.

6. CONCLUDING REMARKS

This paper concentrated on basic rule proposing and combining methods rather than on particular treatments of visual relations and representations. We expect to have more to say on these topics in future work. Other areas we expect to follow up on include the problem of nonmonotonicity inherent in allowing users to edit or alter their input, the problem of offering help to users in an incremental parsing situation, and various problems associated with reversing the grammars shown here in connection with generation of visual output from the semantics of underlying data.

Although we have been applying Relational Unification Grammars in graphical domains, there is reason to suppose that such extensions of unification grammars may prove

useful for natural languages as well. In particular, using relations such as case and gender agreement in place of left- and right-adjacency as the foundation for grammatical description may prove superior for so-called free word order languages. We expect that the parsing algorithm presented here would apply in such cases.

7. ACKNOWLEDGEMENTS

This work has been carried out under the sponsorship of the MCC Human Interface Laboratory, directed by Bill Curtis. The paper is to a large extent a revision and extension of an earlier paper coauthored with Louis Weitzman (Wittenburg and Weitzman 1990), who together with Jim Talley has worked closely with the author in developing the concepts and building the systems discussed here. Other colleagues I wish to thank include Rich Cohen for his comments on early versions of this paper and for his support of the HITS blackboard technologies used in our implementations, Chinatsu Aone for discussions on the rule formalism, and Gale Martin and Jay Pittman of the neural net character recognition group for getting me involved in this project in the first place.

REFERENCES

Aho, Alfred V., and Jeffrey D. Ullman. 1972. "The Theory of Parsing, Translation, and Compiling," Prentice Hall.

Anderson, Robert H. 1968. "Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics," in M. Klerer and J. Reinfelds (eds.), Interactive Systems for Experimental Applied Mathematics, Academic.

Avery, James. 1988. "Interactive Worksurface: An Interface Paradigm for Sketchable Things," MCC tech report no. ACA-HI-127-88.

Chang, Shi-Kuo. 1988. "The Design of a Visual Language Compiler," in Proceedings of the 1988 IEEE Workshop on Visual Languages, October 10-12, Pittsburgh, PA.

Crimi, C., A. Guercio, G. Tortora, and M. Tucci. 1989. "An Intelligent Iconic System to generate and to interpret Visual Languages," in Proceedings of the 1989 IEEE Workshop on Visual Languages, October 4-6 1989, Rome, Italy.

Fu, K.S. 1974. Syntactic Methods in Pattern Recognition. Academic.

Golin, Eric J., and Steven P. Reiss. 1989. "The Specification of Visual Language Syntax," in Proceedings of the 1989 IEEE Workshop on Visual Languages, October 4-6 1989, Rome, Italy.

Helm, Richard, and Kim Marriott. 1990. "Declarative Specification of Visual Languages," in Proceedings of the 1990 IEEE Workshop on Visual Languages, October 4-6, Skokie, Illinois.

Kay, Martin. 1980. "Algorithm Schemata and Data Structures in Syntactic Processing," Xerox Palo Alto Research Center, tech report number CSL-80-12.

Kay, Martin. 1989. "Head-Driven Parsing," in Proceedings of the International Workshop on Parsing Technologies, 28-31 August 1989, Pittsburgh, PA, Carnegie Mellon.

Kaplan, Ronald. 1973. "A General Syntactic Processor," in R. Rustin (ed.), Natural Language Processing, pp. 193-241, New York: Algorithmics.

Martin, Gale, James Pittman, Kent Wittenburg, Richard Cohen, and Tom Parish. 1990. Sign Here, Please: State of the Art, Computing without Keyboards. BYTE magazine, July 1990.

Satta, Georgio, and Oliviero Stock. 1989. "Head-Driven Bidirectional Parsing: A Tabular Method," in Proceedings of the International Workshop on Parsing Technologies, 28-31 August 1989, Pittsburgh, PA, Carnegie Mellon.

Shieber, Stuart. 1989. Parsing and Type Inference for Natural and Computer Languages, Technical note 460, SRI International.

Shieber, Stuart. 1986. An Introduction to Unification-Based Approaches to Grammar. Center for the Study of Language and Information, Stanford University.

Shieber, Stuart. 1985. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In Proceedings of the 23rd Meeting of the Association for Computational Linguistics, 8-12 July 1985, University of Chicago.

Shieber, Stuart, Gertjan van Noord, Robert Moore, and Fernando C. N. Pereira. 1989. "A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms," in Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, 26-29 June 1989, Vancouver.

Tomita, Masaru. 1989. "Parsing 2-Dimensional Language," in Proceedings of the International Workshop on Parsing Technologies, 28-31 August 1989, Pittsburgh, PA, Carnegie Mellon.

Wittenburg, Kent, and Louis Weitzman. 1990. "Visual Grammars and Incremental Parsing for Interface Languages," in Proceedings of the 1990 IEEE Workshop on Visual Languages, October 4-6, Skokie, Illinois.