Journal of Visual Languages and Computing (1997) **8,** 375–402

On the Classification of Visual Languages by Grammar Hierarchies

KIM MARRIOTT[*] AND BERND MEYER[†‡]

[*] *Department of Computer Science, Monash University, Clayton, Victoria 3168, Australia, marriott@cs.monash.edu.au,* [†] *Department of Computer Engineering, University of Colorado, Boulder, CO 80303, U.S.A., bernd.meyer@acm.org*

Received 1 December 1996; accepted 21 April 1997

Visual language specification has been investigated for more than two decades now and many different formalisms for specifying and parsing visual languages have been invented. However, there has been little attempt to develop a systematic and comprehensive hierarchy of visual languages based on their formal properties. Given the importance of the Chomsky hierarchy for the theory of textual languages and the difficulty of comparing these many different visual language formalisms, it is clear that there is a need for such a hierarchy. We develop a hierarchy for visual languages and investigate the expressiveness and cost of parsing for classes defined therein. Although the hierarchy is based on the constraint multiset grammar formalism, we sketch how other visual language specification formalisms can be mapped into constraint multiset grammars so that a comparison is made possible. One consequence of our work is that a large class of 'naturally occurring' visual languages are inherently context-sensitive, so that the core of such a hierarchy has to be built around different forms of context-sensitivity.

1. Introduction

MANY DIFFERENT formalisms for the specification of visual languages have been investigated for more than two decades now. Grammar-like formalisms range from early approaches like web and array grammars [1], and shape grammars [2] to recent formalisms like positional grammars [3], relation grammars [4], unification grammars [5–7], attributed multiset grammars [8], constraint multiset grammars [9] and several types of graph grammars [10]. There are also a variety of non grammar-like formalisms, including algebraic approaches [11] and logic-based approaches [12–14].

In order to understand the tradeoffs and comparative advantages of these different formalisms, there is a need for a systematic and comprehensive taxonomic hierarchy of visual languages. Such a taxonomy should specify properties of visual languages which allow us to classify them and the position of a particular class of languages in the taxonomy should correspond to the expressiveness of the class and the cost or

[‡] This research was undertaken when the author visited Monash University while supported by DFG Grant ME11/94.

1045-926X/97/040375+28 $25.00/0/vl970053

decidability of parsing and related problems. In the world of textual languages the Chomsky hierarchy serves this purpose. The importance of the Chomsky hierarchy to traditional formal language theory can hardly be overstated and it is difficult to imagine where textual compiler technology would be today without it. A similar classification for visual languages should serve to unite the different streams in visual language theory research and to facilitate the use of theoretic results in the implementation of visual language software environments. Here we present an approach to such a hierarchy.

Ideally, a taxonomic hierarchy for visual languages should be general enough to include all of the different visual language formalisms. This, however, seems unachievable due to the substantial differences between these formalisms. They not only differ in the selection of the basic framework like first-order logic, algebra or grammars, but even within these groups there is too much variation in the basic notion of what the building blocks of a visual language are. No matter which type of framework is selected, there are still a lot of important choices to be made:

- What are the atomic (basic) objects to work on? Some formalisms use only lines or points as atomic objects, while others allow almost arbitrary complex symbols, which means that the borders between lexical and syntactic recognition are shifting.
- Are these objects featureless or can they have attributes or sub-objects, etc.? Several formalisms do not use attributes at all, while others allow arbitrary complex attributes or sometimes even use nested object structures.
- If attributes are used, what type of computation on these attributes is allowed? Sometimes no computation is allowed, whereas other formalisms allow the use of any partial function to calculate values.
- How are spatial relations handled? In many formalisms they are explicitly given as uninterpreted, abstract relations, whereas other methods use implicit relations that are expressed as conditions on attribute values.
- Are spatial relations interpreted? An interpretation of a spatial relation can either be symbolic or arithmetic (in terms of the underlying geometry) or relations can be completely uninterpreted. Interpreting spatial relations allows us to express general geometric axioms (e.g. the transitivity of spatial inclusion) and thus enriches the expressiveness of the framework.
- What is the general mathematical structure used to represent a visual expression? In many approaches graphical objects are aggregated as arbitrary sets or multisets, whereas other methods use some tiling of the space (normally a grid-like structure) on which objects have to be arranged.

These choices represent only some of the basic design decisions to be made, yet it is clear that they deeply influence the expressive power and the computational complexity of the resulting specification method. A comparison of different formalism even within a single framework like grammars is thus very difficult. This is in stark contrasts to the case of textual languages in which it is clear that the basic object to be manipulated is a sequence of symbols.

Given this variation, we have to focus on a single specification framework to find some way of classifying visual languages. For our approach we have chosen to look at the most commonly used framework, grammatical methods. Within this framework, it

seems the only reasonable way to obtain a general classification of visual languages is to base the taxonomy on some particular but flexible grammar formalism and to clarify how the other formalisms can be mapped into this classification. These mappings in turn are likely to give rise to new, finer grained classes of languages. Since grammar formalism can more generally be regarded as a particular type of rewrite system, there is also hope that once a classification based on grammar systems is established, the links to logic and algebraic methods can be reestablished. This is the approach we have taken.

The approach is thus structured as follows. We have chosen one powerful grammar formalism as the 'core method' (called CCMGs) and have built a hierarchy of visual languages classified by restricted versions of this formalism. To characterize the positions of other types of formalisms [like positional grammars (PG), relational grammars (RG), atomic relational grammars (ARG), see Section 2.1] within this hierarchy, we have to map them to some subclass of the CCMG grammar formalism. One advantage of this structure is that we do not need n^2 but only n mappings to compare n formalisms.

Our main technical contributions are fourfold: first, we introduce a taxonomy for visual languages based on different classes of constraint multiset grammars (CMGs). Second, we support the choice of CMGs as the basis for the taxonomy by showing how several other representative formalisms—positional grammars, relation grammars and unification grammars—can be mapped into CMGs. Third, we study the relative expressiveness of classes in this taxonomy. Fourth, we characterize the cost of parsing with each class in the language. This paper is intended to provide an introduction to the CMG hierarchy and its properties. For this reason the majority of the technical and rather tedious proofs have not been included. In this paper we present example languages that characterize the differences between the classes in the hierarchy and outline the ideas of how to prove the theorems. For the full technical proofs the reader is referred to the book by Marriott and Meyer [15].

To our knowledge this is the first taxonomic hierarchy for visual languages apart from some work along these lines for early approaches like web grammars and array grammars [16, 1]. Array grammars, however, lack expressiveness. Later research based on web grammars can be found in the graph grammar community. A very systematic survey is by Courcelle [10], where connections between first-order logic, monadic second-order logic, algebraic specification and graph grammars are discussed. However, it is difficult to interpret the various graph grammar hierarchies in terms of visual languages. The main reason is that it is inherent in the graph grammar formalism that spatial relations are treated as uninterpreted, thus in a certain sense 'meaningless' relations. It seems natural to interpret spatial relations, be it arithmetically or symbolically, since otherwise their *natural spatial properties and interdependencies* are lost.

The remainder of the paper is structured as follows: the next section presents the constraint multiset grammar (CMG) formalism as well as mappings from other grammar methods to CMGs. Section 3 introduces a modified version of CMGs, copy-restricted CMGs (CCMGs), upon which we will build the hierarchy. A hierarchy of restricted CCMG classes is formally defined in Section 4. These classes are compared in regard to their expressiveness in Section 5 and their computational complexity in analyzed in Section 6. Section 7 concludes.

2. Constraint Multiset Grammars

We have chosen constraint multiset grammars (CMGs) as the basis for our taxonomy. CMGs have informally been introduced by Helm *et al.* [17] and have been used for a variety of reasonably complex tasks such as the parsing of state diagrams and mathematical equations. A precise formal treatment is given by Marriott [9] and we review only the basic notions here. CMGs are rewrite systems for multisets of typed objects. Spatial relations are handled by defining arithmetic constraints on object attributes, i.e. there is no explicit representation for spatial relations. A distinguishing feature of CMGs is that they can either be regarded as normal rewrite systems or as a special form of constraint logic programs. In fact, one parsing algorithm for CMGs is a variant of bottom-up derivation for constraint logic programs.

A concrete example of a CMG production taken from the specification of state diagrams is

```
TR:transition::= A:arrow, T:text
  where exists R:state, S:state where
  T.midpoint close_to A.midpoint,
  R.radius=distance(A.startpoint, R.midpoint),
  S.radius=distance(A.endpoint, S.midpoint)
  and TR.from=R.name, TR.to=S.name, TR.label=T.string.
```

This production defines that a transition in a state diagram consists of an arrow pointing from some state to some other state and being labeled by a text object. Since the *state* objects appear in the **where exists** part of the production, they are context objects, i.e. their existence is checked when the production is applied, but they are not reduced by the production application, whereas the *arrow* object and the text object are reduced. The geometric conditions that the arrow must connect the two states and that the text must be associated with the arrow are stated as arithmetical expressions using *distance* and *close_to*. These have to be thought of as a simple arithmetic function and a predicate which are applied to the arithmetic attributes of the objects. Finally, the last line of the production defines the attributes of the defined *transition* object.

In the more concise standard notation for CMGs each production has the form

$$x ::= X_1, \ldots, X_n \textit{ where exists } X'_1, \ldots, X'_m \textit{ where } C \textit{ and } \vec{\mathbf{v}} = E$$

meaning that the non-terminal x can be rewritten to the multiset $X_1, \ldots, X_n$ if the sentence contains symbols $X'_1, \ldots, X'_m$ (the context) such that the attributes of these symbols satisfy the constraint C. $\vec{\mathbf{v}}$ denotes the vector of attributes of x whose values are defined by the vector expression E over attributes of other objects in the production. In the above example we have $\vec{\mathbf{v}} = (TR.from, Tr.to, Tr.label)$ and $E = (R.name, S.name, T.string)$.

In the present paper it will be convenient to represent productions in a CMG using a slightly different syntax. The above production will be represented by:

$$\begin{array}{l} transition, state_R, state_S \leftarrow arrow, text, state_R, state_S \| \\ text.midpoint\ close\_to\ A.midpoint, \\ state_R.radius = distance(arrow.startpoint, state_R.midpoint), \end{array}$$

$state_S.radius = distance(arrow.endpoint,\ state_S.midpoint) \wedge$
$transition.from = state_R.name,$
$transition.to = state_S.name,$
$transition.label = text.string.$

More generally productions are written as

$$x X'_1, \ldots, X'_m \leftarrow X_1, \ldots, X_n X'_1, \ldots, X'_m \,||\, C \wedge \vec{\mathbf{v}} = E$$

where x is a non-terminal symbol with attributes $\vec{\mathbf{v}}$, $X_1, \ldots, X_n$ and $X'_1, \ldots, X'_m$ are terminal or non-terminal symbols with $n \geq 0$ and the constraint C and expression E are over the attributes of $X_1, \ldots, X_n$ and $X'_1, \ldots, X'_m$. Throughout the paper we will use the different notations interchangeably.

Definition 1. *A constraint multiset grammar* (*CMG*) over a computational domain D is a quadruple (T_T, T_{NT}, S, P) consisting of a set of terminal symbols T_T, a set of non-terminal symbols T_{NT}, $T_T \cap T_{NT} = \emptyset$, a start symbol $S \in T_{NT}$ and a set of productions P of the above form.

A sentence, S, in a CMG is a multiset of tokens, i.e. terminal and non-terminal symbols. A single derivation step of a CMG G from sentence S to S', written $S \Rightarrow_G S'$, can be performed for some production in G,

$$x X'_1, \ldots, X'_m \leftarrow X_1, \ldots, X_n X'_1, \ldots, X'_m \,||\, C \wedge \vec{\mathbf{v}} = E$$

where $\vec{\mathbf{v}}$ are the attributes of x if there is an assignment θ of attributes to the tokens in the production such that

- $\{\theta(x), \theta(X'_1), \ldots, \theta(X'_m)\} \subseteq S$,
- θ satisfies $C \wedge \vec{\mathbf{v}} = E$, and
- $S' = (S \setminus \{\theta(x)\}) \cup \{\theta(X_1), \ldots, \theta(X_n)\}$.

Informally speaking, a production P can be applied if the current sentence contains all left-hand side tokens of P and if it is possible to find attribute values for all these tokens such that the constraints given in P can be satisfied. It is applied by removing the non-context symbols on the left-hand side of P from the sentence and adding the non-context symbols on the right-hand side of P under this assignment to the new sentence.

The language $\mathcal{L}(G)$ generated or recognized by a CMG, G, is the set of all multisets of attributed terminal symbols that can be derived from a distinguished non-terminal symbol with no attributes called the *start symbol*, T_S, by repeated application of productions in G:

$$\mathcal{L}(G) = \{S' \mid \{T_S\} \Rightarrow_G^* S'\}$$

2.1. Comparison to Other Formalisms

Above, we argued that we have to select one particular formalism to build the hierarchy upon. When such a formalism is chosen we have to make sure that other specification

methods can appropriately be mapped to this formalism so that the hierarchy can be used to analyze general properties of visual languages independently from a particular formalism.

Certainly any specification method can be mapped into the full form of CMGs, since their unrestricted form is computationally equivalent [9]. But it is important to know *how* these mappings can be done to ensure that these methods can reasonably be assigned to certain classes of the hierarchy. In this section we discuss three quite different types of grammar formalisms and show how they can be mapped to CMGs. Since the full proofs of equivalence are rather involved and do not contribute to the understanding of the main idea, we omit them here.

2.1.1. Positional Grammars

Positional grammars (PGs) [3] are rewrite systems working on sets of symbols which are located at a position on some grid. In this regard their basic approach is fairly different from CMGs that allow the specification of arbitrary spatial constraints. A PG production has the form

$$A \rightarrow x_1 R_1 x_2 R_2 \ldots R_{m-1} x_m$$

where A is a non-terminal, x_i is a terminal or non-terminal and R_i is a spatial relation of the form (dx, dy) indicating that x_{i+1} must be found at the position $(a+dx, b+dy)$ if x_i was located at (a, b).

Theorem 1. *For every PG, G, there is a CMG, G', with $\mathcal{L}(G) = \mathcal{L}(G')$.* Furthermore, *$G'$ does not use any context symbols.*

It is important to note that the above theorem implies that a CMG which is equivalent to some PG is always context-free. The production set of G' is easily obtained by replacing each PG production of the above form with a CMG production

$$\begin{aligned} A \leftarrow x_1 \ldots x_m \,||\, & p(dx_1, dy_1, x_1.x, x_1.y, x_2.x, x_2.y) \\ & \ldots \\ & p(dx_{m-1}, dy_{m-1}, x_{m-1}.x, x_{m-1}.y, x_m.x, x_m.y) \wedge \\ & A.x = x_1.x \wedge A.y = x_1.y \end{aligned}$$

where $R_i = (dx_i, dy_i)$ and $p(m, n, x_1, y_1, x_2, y_2) \Leftrightarrow x_1 + m = x_2 \wedge y_1 + n = y_2$. The terminal and non-terminal sets are the same in both grammars, but the symbols of G' have to be attributed with their x and y coordinates on the grid. It is easy to see that both grammars define the same language.

2.1.2. Relation Grammars

Relation grammars (RGs) [4] are rewrite systems which simultaneously work on a set of unattributed symbols and a set of relations over these symbols. The relations are purely

symbolic and are not interpreted arithmetically. There are two types of productions: s-rules for rewriting symbol sets and r-rules for rewriting relation sets. Each s-rule has the form

$$[n]A ::= \{m_1, \ldots, m_k\}, \{p_1(x_1, y_1), \ldots, p_l(x_l, y_l)\}$$

where n is a production index, A is a non-terminal, m_i are terminals or non-terminals, and $p_i(x_i, y_i)$ are relations over some m_i. Each r-rule has the form $r(x_1, x_2) ::= [n, j]\, P$ where $r(x_1, x_2)$ is a relation, n, j are indices, and P is a set of relations. The symbol set is rewritten by s-rules replacing A in the object set by $\{m_1, \ldots, m_k\}$ and adding $\{p_1(x_1, y_1), \ldots, p_l(x_l, y_l)\}$ to the relation set. Each time an s-rule with index n rewrites a symbol A, every relation $r(A, x)$ in the relation set is rewritten by the r-rule with index $[n, 1]$ and every relation $r(x, A)$ is rewritten by the r-rule with index $[n, 2]$. An RG is called a 1NS-RG if every production involves only relations containing at most one non-terminal [4]. Binary relation grammars are equivalent to edNLC graph grammars [18].

Speaking in less technical terms there are three important observations to be made: (1) RGs work with two interdependent rewrite systems. This makes their derivation structure different from normal grammar systems. (2) The meaning of a spatial relations as used in RGs is not necessarily independent of the derivation of a grammar, since the symbolic derivation of the spatial relation is dependent on which grammar productions are applied. It is thus difficult to give such a spatial relation an abstract, 'external' meaning. (3) This means that RGs are in some form inherently context-sensitive. Nevertheless, it is possible to map 1NS-RGs to CMGs.

Theorem 2. *For every 1NS-RG, G, there is a CMG, G', with $\mathcal{L}(G) = \mathcal{L}(G')$.*

To perform such a mapping we need to model the equivalent CMG in such a way that the symbols carry all the information about the spatial relations they are participating in. The normal way to express spatial relations in CMGs cannot be used, since spatial constraints in CMGs have an external meaning that is independent of the derivation structure. The basic idea of how to simulate the two coupled rewrite systems is to drop the set of relations and to use a list-valued attribute *prop* for each symbol instead. This attribute carries a symbolic representation of all spatial relations in which the symbol is involved. By rewriting this attribute in the productions of G', the coupled r-rules of G can be simulated.

Given a relation sentence (S, R) with symbol set S and relation set R, the corresponding CMG sentence is S such that for each symbol $s \in S$, *s.prop* contains exactly one item $p_1(X)$ for each $p(s, X) \in R$ and exactly one item $p_2(X)$ for each item $p(X, s) \in R$. For every s-rule an equivalent CMG productions is created that rewrites the object in the same way verifying the spatial relations by testing if the appropriate relations are contained in the attribute *prop*. Additionally, it computes the *prop* attribute for its left-hand side symbol by rewriting the *prop* attributes of $m_1, \ldots, m_k$ according to the r-rule with the appropriate index.

Of course, the CMG obtained in this way is rather unnatural, since it does not make any use of arithmetic constraints. The translation becomes much more natural, if the reasonable assumption is made that terminal symbols have geometric attributes and that all admissible relation symbols have a geometric interpretation such that they can be

tested by computations on these attributes. In this case, the r-rules of G would only be used to rewrite spatial relations on non-terminals consistently according to their meaning until only terminal symbols are involved in the relation representation. The same effect would be achieved in G' by percolating the attributes of these objects in their derivation and applying the constraint formula that defines the relation immediately. It is important that G' would then not need to use the aggregative attribute *prop*. This property of relations is, however, not guaranteed for arbitrary 1NS-RGs.

2.1.3. Unification Grammars

Visual unification grammars have been introduced by Wittenburg and Weitzmann [5]. Their successor, atomic relational grammars (ARGs), is presented by Wittenburg [6, 7]. ARGs are rewrite systems operating on a single set of symbols and (evaluable) relations. An ARG production has the form $A \rightarrow \alpha/\beta/F$ where A is a non-terminal, α a string of non-terminals and terminals, β a set of constraints of the form $r(x, y)$ where x and y refer either to a symbol in α or to an attribute of such a symbol. F is a set containing exactly one assignment $a = x$ for each attribute a of A where x is again either a symbol in α or an attribute of such a symbol. All arguments to relations have to be 'atomic objects' or 'individuals in the input set', that is terminal objects [6].

Theorem 3. *For every ARG, G, there is a CMG, G', with $\mathcal{L}(G) = \mathcal{L}(G')$.*

As every relation on two atomic objects can be computed from some of their geometric attributes, each ARG rule of the form above can be replaced with a CMG production

$$A \leftarrow \alpha \| \beta' \wedge F'$$

ARGs use constraints on symbols, but a CMG must express them by constraints on attributes. Every ARG constraint $r(x, y)$ must have some computable interpretation depending on the attributes of x and y. Adopting the domain for the computation of ARG constraints as the constraint domain of the CMG it is easy to modify β into β' such that β' directly uses the attributes of x and y by unfolding the definition of r into the production. Most examples of ARGs have used arithmetical interpretations so that the standard domain of real numbers can be used. However, other domains could be used and would require the same constraint domain to be used for the CMG. The only remaining problem is that ARG attributes can be symbols (objects) as well as values. Objects cannot be used as attribute values in a CMG. So F has to be modified into F' such that each assignment $a = x$, where x references an object, is rewritten as a sequence of assignments such that all attributes of x are transferred to A.

As the rules for the application of productions in ARGs and CMGs are virtually the same, even the structure of the derivation is the same in both cases, and it is obvious that G and G' define the same language.

3. Copy-restricted Grammars

So far we have used the unrestricted CMG formalism. Unfortunately, it is not possible to base an interesting hierarchy directly on this definition of CMGs. This is because the

arbitrarily complex attribute computation allowed in CMGs is too powerful to allow us to build a hierarchy with sharp class borders. The problem is that we can simulate the derivation mechanisms by accumulating relevant attributes of potential context symbols. For instance, witness the encoding used for embedding relation grammars in CMGs. This difficulty is not surprising, since CMGs over the domain of integers with functions $\{+, -, =0\}$ are computationally adequate [9]. Therefore, we have to restrict the ways in which attributes may be used.

On the other hand, we cannot throw away attributes altogether. A hierarchy based on propositional multiset grammars, i.e. CMGs in which no symbols have attributes, has little direct relevance to visual languages since attributes are necessary to encode the visual aspects of the language. Therefore, we need to allow attributes but must not allow 'complex' computation over the attributes.

The most natural and radical restriction is not to allow computations at all—attributes of the LHS symbol can only be *copied* from the attributes of the RHS symbols.

Definition 2. *A copy-restricted CMG (CCMG)* is a CMG in which every production has the form

$$x X'_1 \ldots \mathrm{X}'_m \leftarrow X_1 \ldots X_n X'_1 \ldots X'_m \| C \wedge \vec{\mathbf{v}} = E$$

where $\vec{\mathbf{v}}$ is the vector of the attributes of x and E is a vector of attributes of $X_1 \ldots X_n, X'_1 \ldots X'_m$.

Copy-restricted CMGs give rise to slightly simpler parsing algorithms and lend themselves to a formal treatment as all computation is moved into the derivation process proper, rather than attribute computation. Keep in mind that some computations of attributes can still be done by using the derivation mechanism of the grammar. The copy restriction corresponds to limiting the computable attribute functions to those that can be done by the grammar formalism instead of some additional constraint formalism. It is, e.g. still possible to compute the bounding box of two objects by using the following grammar construction:

$$x \leftarrow yz \| \begin{array}{l} y.top \le z.top, y.left \le z.left, y.right \ge z.right, y.bottom \ge z.bottom \wedge \\ x.top = y.top,\ x.left = y.left,\ x.bottom = y.bottom,\ x.right = y.right \end{array}$$

$$x \leftarrow yz \| \begin{array}{l} y.top \le z.top, y.left \le z.left,\ z.right \ge y.right,\ z.bottom \ge y.bottom \wedge \\ x.top = y.top,\ x.left = y.left,\ x.bottom = z.bottom,\ x.right = z.right \end{array}$$

$$x \leftarrow yz \| \begin{array}{l} y.top \le z.top,\ z.left \le y.left, y.right \ge z.right,\ z.bottom \ge y.bottom \wedge \\ x.top = y.top,\ x.left = z.left,\ x.bottom = z.bottom,\ x.right = y.right \end{array}$$

$$\vdots$$

Though the copy restriction seems to be a severe limitation, virtually all of the examples that have been presented for CMGs before are, in fact, CCMGs. For example,

recall the production for transitions given above; here the attributes of the LHS symbol *TR* are directly copied from the RHS symbols *R* and *S*. So CCMGs are not only of theoretical value, but can directly be used for real specifications. Indeed the Picture Layout Grammars of Golin [19] are copy restricted.

In the light of copy restriction, we can revisit the mappings of other formalisms into the full CMG formalism that were discussed in Section (2.1). The mapping of PGs obviously already generates copy-restricted CMGs, since no attribute computations are used. For ARGs our mapping only generates a copy-restricted CMG, if the ARG itself assigns attributes by copying, that is, if the ARG adheres to the copy-restriction, too. In this case it, too, generates a CCMG that does not use context symbols. The mapping of RGs that we have discusses is more complex and obviously does not yield copy-restricted grammars, since it uses list attributes to simulate r-rules. However, we will later introduce a more complex class of copy-restricted CMGs ($CCMG_1^{\forall}$ s) to which RGs can be mapped. The basic idea of this new mapping, which is too involved to be discussed here, is that RGs can be regarded as a form of hyper-graph rewriting technique and that $CCMG_1^{\forall}$ s can be used to simulate hyper-graph rewriting.

For CCMGs we can make the simple but important observation that the amount of information that can pass through any node of a derivation tree is limited. This is in contrast to grammars that can use accumulating attribute data structures like lists and thus are able to pass an unlimited amount of information in the nodes of a derivation tree.

Lemma 1. *No non-terminal $n \in V_n$ occurring in any derivation for a copy-restricted CMG can accumulate the attributes of more than a constant grammar dependent number c of other terminals or non-terminals if their values are different.*

Proof. Since the grammar is copy-restricted, no accumulating data structures, like lists, can be used. Thus, every new value to be assigned to a LHS attribute requires a separate attribute. Since the number and types of attributes for every non-terminal is fixed in the productions of *G*, only a fixed number of attributes can be collected at any node. □

4. The Hierarchy of Copy-Restricted Grammars

Using CCMGs as the basis, we will now develop a hierarchy of visual languages based upon restricted versions of CCMGs. Originally, there were four dimensions along which the CMG formalism can be extended or restricted: (1) the admissible types of attribute computations; (2) the form of productions in the grammar; (3) the types of constraints that can be used and (4) general properties of the grammar like confluence. We have eliminated the first dimension by restricting our discussion to CCMGS. Three dimensions remain. In this section the form of productions in the grammar and the admissible constraint types will be used to build the hierarchy. Additional grammar properties will be discussed in the conclusion. To classify different forms of productions it seems most natural to consider the analogue of the restrictions used in the Chomsky hierarchy on the form and number of symbols in the LHS and RHS of the productions. We will thus

define and analyze forms of CCMGs analogue to regular, context-free, context-sensitive and unrestricted forms of conventional string grammars. We will then extend these grammar classes by defining different and new forms of contextual constraints; in particular, we will look at different forms of quantification that can be used to define contexts.

The production used in the definition of CCMGs given above is termed *type 1* (*context-sensitive*). The production for a transition is an example of a type 1 production.

Definition 3. A type x CCMG (denoted as CCMG_x) is a CCMG in which every production is of type x.

The first natural restriction is to disallow any context symbols:

Definition 4. A CCMG production is *type 2* (*context-free)* if it is a type 1 production of form $A \leftarrow v \| C \wedge F$ where $A \in T_{NT}$ and $v \in (T_T \cup T_{NT})^+$.

An example of a type 2 production is

```
A:labelledArrow::=Ar:arc, T:text where
  T.midpoint close_to Ar.midpoint
  and A.label=T.string, A.from=Ar.from, A.to=Ar.to.
```

where *arc* and *text* are terminal symbols.

This type can be further restricted to an even simpler class which is analogous to regular grammars:

Definition 5. A CCMG production is *type 3* (*regular*) if it is a type 2 production of one of the forms $A \leftarrow b \| C \wedge F$ or $A \leftarrow bB \| C \wedge F$ where $b \in T_T$ and $B \in T_{NT}$.

For example, the production above can easily be rewritten into two type 3 productions:

```
A:labelledArrow::=T:text, Ar:arrow where
  T.midpoint close_to Ar.midpoint
  and A.label=T.string, A.from=Ar.from, A.to=Ar.to
A1:arrow        ::=A2:arc
  and A1.to=A2.to, A1.from=A2.from, A1.midpoint=A2.midpoint.
```

It is also natural to generalize type 1 CCMGs as they are not expressive enough to allow the specification of all naturally occurring visual languages. For example, they cannot specify constraints that are quantified universally and so it is impossible to specify complete graphs using a type 1 CCMG. Analogous to the Chomsky hierarchy we can introduce type 0 grammars which drop the monotonicity criterion of type 1 grammars.

Definition 6. A CCMG production is *type 0* (*unrestricted*) if it is of the form $uA \leftarrow v \| C \wedge F$, where $u \in (T_T \cup T_{NT})^*$ and $A \leftarrow v \| C \wedge F$ is of type 1 or v is empty.

We can, for example, use a type 0 production to handle segmentation of lines. The following production splits a T-shape consisting of two line segments into three separate segments:

```
L1, L2 : lineSegment : : = L3 : lineSegment
  where exists L4 : lineSegment where on(L4.end, L3.start, L3.end)
  and L1.start = L3.start, L1.end = L2.start = L4.end, L2.end = L3.end.
```

We will make use of this later when discussing the expressiveness of type 0 CCMGs.

The reader may also be wondering if there is a class between type 0 and type 1 CCMGs, namely the monotonic type 0 CCMGs. However, as for string grammars, the type 1 CCMGs are actually equivalent to the monotonic type 0 grammars since every non-shortening production can be replaced by a sequence of type 1 productions. For this reason we will not consider monotonic type 0 CCMGs further.

Lemma 2. *For every CCMG*$_0$ G *containing only productions of the form* $u \leftarrow v \| \varphi$ *where* $|u| \leq |v|$ *there exists a CCMG*$_1$ G' *with* $\mathcal{L}(G) = \mathcal{L}(G')$.

The other dimension in which we can extend CCMGs is to modify the form of the constraint in the production. This dimension is particularly interesting since it has no analogue for string grammars. In particular, negative contexts (i.e. contexts that must not be present in order for a certain production to be applicable) add expressive power [17]. We therefore investigate the following extended form of CCMGS:

Definition 7. A production is *type* $x\forall$ if it is of the form $u \leftarrow v \| C \wedge F$, such that $u \leftarrow v \| F$ is type x and C is a universally quantified formula $\forall x_1 \ldots x_l : \varphi$ where φ is constraint over the attributes of $x_1, \ldots, x_l$ and the symbols in v. A grammar is type $CCMG_x^{\forall}$ if all its productions are of type $x\forall$.

Note that we are assuming that the underlying constraints involving the attributes can be negated, so that a negative context can be rewritten to a universal context. For example, the following three productions are equivalent:

$$simpleState \leftarrow state \| \forall circle : outside(circle.region, state.region)$$
$$simpleState \leftarrow state \| \neg \exists circle : inside(circle.region, state.region)$$
$$simpleState \leftarrow state \| \neg \exists circle : \neg outside(circle.region, state.region)$$

More generally, we can consider the case in which the constraint is allowed to be an arbitrary first-order formula.

Definition 8. A production is *type* $xFOL$ if it is of the form $u \leftarrow v \| C \wedge F$, such that $u \leftarrow v \| F$ is type x and C is a first-order formula $Q_1 x_1, \ldots, Q_l x_l : \varphi$ where φ is a constraint

over the attributes of $x_1, \ldots, x_l$ and the symbols in v and $Q_i \in \{\exists, \forall\}$. A grammar is of type $CCMG_x^{FOL}$ if all of its productions are of type *xFOL*.

Note that it does not make a great deal of sense to consider type 3FOL and type 2FOL grammars as the extension to first-order formula allows existentially quantified symbols and so type 3FOL and type 2FOL CCMG grammars are essentially equivalent to type 1FOL CCMGs.

5. Expressiveness

In the previous section we have introduced a hierarchy of formalisms based on CCMGs. In this section we will investigate their expressiveness and, in particular, their relative expressiveness. It will turn out that even context-sensitive (i.e. type 1) CCMGs are not powerful enough to specify several interesting languages. The same applies to other multi-dimensional context-sensitive grammar formalisms, if they are restricted to copy-attributes. The full set of classes under investigation and their hierarchy is given in Figure 1. In the following $\mathcal{L}(G)$ denotes the language defined by a CCMG G or the class of languages defined by a CCMG grammar class G.

Strict inclusions hold between type 3, type 2 and type 1 CCMGs. Type 3 CCMGs can only specify languages in which the constraints are in a certain sense 'local'. More exactly, locality restriction means that for a language to be defined by a regular CCMG it

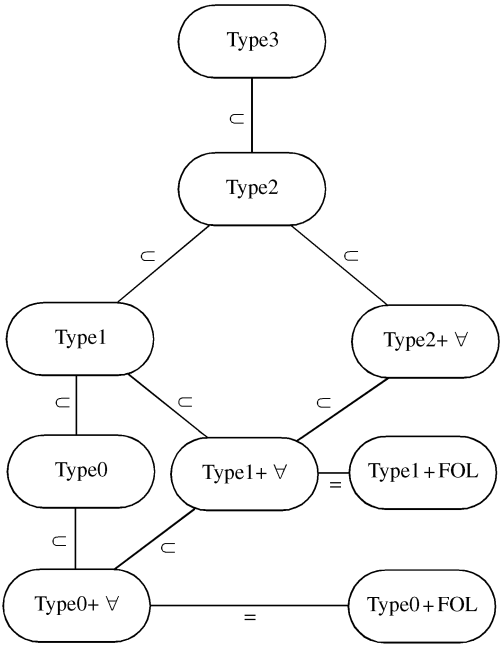

Figure 1. Complete CCMG hierarchy

must be possible to define a sequential ordering for the objects in every sentence of the language such that almost all constraints that have to be checked involve only objects whose distance way in the sequence is bounded by a constant. Only a constant number of exceptions is allowed. An example of a language for which this restriction does not hold is the language T of trees. This is because, no matter in what order we write the nodes of the tree, there is an arbitrary large distance between some node and its parent in this sequence. We now formalize this argument that type 3 CCMGs are strictly less expressive than type 2 CCMGs.

Theorem 4. $\mathcal{L}(CCMG_3) \subsetneq \mathcal{L}(CCMG_2)$.

Proof. The inclusion follows from the definition. We now prove that the inclusion is strict by showing that the language T of trees can be defined with a $CCMG_2$ but not with a $CCMG_3$.

We first demonstrate that T can be defined with a $CCMG_2$. For the sake of simplicity, we use a graphical tree representation without explicit nodes, i.e. nodes are assumed where edges meet. The following $CCMG_2$ generates the language of binary trees. The extension to trees of arbitrary degree is straightforward[a]. In this grammar and subsequent ones we follow a convention in which different occurrences of tokens of the same type in the same production are distinguished by indexing the type name.

$start \leftarrow tree$
$tree \leftarrow \varepsilon$
$tree_1 \leftarrow line,\ tree_2 \,\|$
$\quad tree_2.pos = line.end,\ tree_1.pos = line.start$
$tree_1 \leftarrow line_1,\ line_2,\ tree_2,\ tree_3 \,\|$
$\quad line_1.end = tree_2.pos,\ line_2.end = tree_3.pos,$
$\quad tree_1.pos = line_1.start = line_2.start$

We now demonstrate that the language T cannot be generated by a $CCMG_3$.

This follows from the fact that every $CCMG_3$ derivation, like a derivation by a regular string grammar, has the structure of a linear list rather than a tree structure. According to the node sequence in this list the derivation imposes a total ordering on its terminal symbols. We know that a constraint between two terminal symbols can only be stated at some node to which the relevant attributes of these symbols have been passed. Since a position constraint between every parent and its children has to be stated for language T, their position attributes have to be passed to a common node. By Lemma 1 we know that only a constant number of attributes c can pass through any node. For some attribute to be passed from node number i to node number j in some linear derivation the attribute has to pass through all nodes k, $i \leq k \leq j$, too.

For T to be recognizable by a $CCMG_3$, the grammar must therefore define an ordering of the tree in which all nodes only have a distance from their ancestors that is bounded by some constant d. If more than a constant number of nodes violates this condition, a subsequence of nodes $m \ldots n$ can be found in the ordered sequence of

[a] The ε-production is used for clarity only and can obviously be removed to obtain a proper $CCMG_2$.

nodes such that more than c attributes have to pass through m or n. Such an ordering cannot be given for trees independently of their size and degree. Therefore, T cannot be defined by a $CCMG_3$. □

Theorem 4 would intuitively be expected extrapolating from knowledge about string grammars. On the other hand, some languages in $\mathcal{L}(CCMG_3)$ may be unexpected:

1. The string language $L = a^i b^i$ is in $\mathcal{L}(CCMG_3)$ if the terminal symbols have an index attribute giving their position in the string.
2. $L = \{p \mid p$ is a set of lines forming a closed polygon$\}$ is in $\mathcal{L}(CCMG_3)$.

This is due to the fact that not only the structure of the grammar but also constraints on attributes provide means for limited forms of context-handling.

Context-free (type 2) CCMGs, in turn, are less expressive than context-sensitive (type 1) CCMGs, because they can only specify languages with a bounded number of spatial relations for each symbol. So they, as we shall now prove, cannot specify the language G_g of graphs:

Theorem 5. $\mathcal{L}(CCMG_2) \subsetneq \mathcal{L}(CCMG_1)$.

Proof. The inclusion follows from the definition. We now prove the inclusion is strict by showing that G_g is in $\mathcal{L}(CCMG_1)$ but not in $\mathcal{L}(CCMG_2)$. It is easy to give a $(CCMG_1)$ for G_g:

$$
\begin{array}{ll}
start & \leftarrow point,\ start \\
start, point_1, point_2 & \leftarrow start, line, point_1, point_2 \,||\, \\
 & \quad line.from = point_1.pos, \\
 & \quad line.to = point_2.pos \\
start & \leftarrow \varepsilon
\end{array}
$$

This grammar works by first generating an arbitrary set of points and then adding connecting lines between pairs of these points.

We now demonstrate that the language G_g that was just shown to be in $\mathcal{L}(CCMG_1)$ cannot be defined with a $(CCMG_2)$. We will show that for some graphs, in particular complete graphs, it is impossible to test whether all edges are properly connected to nodes, since in order to do this more than a constant number of attributes have to be passed in some node of the derivation tree.

By definition, a graph consists of a set of node markers (points) and a set of edge markers (lines) and each line has to be connected to exactly two points. Thus, each line must be constrained to end at two points. Let the graph G to be parsed be a complete graph with $2k$ nodes.

Lemma 3. *Every $CCMG_2$ can be rewritten such that all productions have one of the forms $x \leftarrow yz \,||\, C \wedge F$ or $x \leftarrow a \,||\, C \wedge F$ where $x, y, z \in V_N$* and *$a \in V_T$. (This can be done by unfolding analogous to string grammars*).

Let T be a context-free derivation tree generated by a $CCMG_2$ in this normal form. It must be of one of the forms given in Figure 2. The reader should ignore the dotted lines for now, since they are only used in context-sensitive derivations. L and R in Figure 2 are not necessarily separate: one can be a subtree of the other. In this case we refer by R only to the area without L. First consider the simpler derivation type on the left side where L and R are disjoint: to state a constraint between two nodes $n_1 \in L$ and $n_2 \in R$ the relevant attributes of at least one node n_1, n_2 must obviously pass through the root node of the subtree containing n_1 and n_2. Now consider the derivation type on the right side: still the attributes have to be passed through the node where L and R join.

Select L and R such that L contains the leftmost k nodes (points) and R the rightmost k nodes of G in the derivation tree. Since G is complete, each node i in L must be connected to each node j in R by an unique line l_{ij}. (a) If l_{ij} is in L then the coordinates of p_j must obviously be passed from L into R in order for l_{ij} to be checked, for p_j is in R. (b) Analogously, if l_{ij} is outside of L the coordinates of p_i have to be passed from L into R. This contradicts Lemma (1), since at least $O(k)$ coordinates have to be passed and k is linear in the input size. □

Adding universal quantification to type 2 CCMGs increases their power significantly. With type 2 CCMGs we cannot specify any language that requires conditions to be checked which are quantified over tuples. This becomes possible with universal quantification. Examples of such languages can easily be found, for instance, the language C consisting of all sets of pairwise disjoint circles.

Theorem 6. $\mathcal{L}(CCMG_2) \subsetneq \mathcal{L}(CCMG_2^{\forall})$.

Proof. As usual, the inclusion follows from the definition. We now prove that the inclusion is strict by showing that C can be defined with a $\forall$ type 2 CCMG but not with a type 2 CCMG without universal quantification.

The following $CCMG_2^{\forall}$ generates C:

$$start \leftarrow diag$$
$$diag_1 \leftarrow circle_1, diag_2 \,||\, (\forall circle_2) : disjoint(circle_1.area, circle_2.area)$$
$$diag \leftarrow \varepsilon$$

The same language C cannot be generated by a $CCMG_2$, since the quantification ranging over tuples $(\forall C_1, C_2 : circle) : disjoint(C_1, C_2)$ cannot be expressed because of bounds on the amount of information that can be passed between sub-derivations in a derivation.

Again, by Lemma 3, we can rewrite any type 2 grammar such that its derivations have the structure given in Figure 2. Let the sentence of C to be parsed consist of $2k$ circles and select L and R such that they contain k circles each. By definition of C it has to be checked for every circle in L (R) that it is disjoint from every circle in R (L). All circles in L must have different coordinates, so $O(k)$ attributes must be passed between L and R.

Since k depends only on the size of the input sentence and thus has no bound, we face the same contradiction to Lemma 1 as in the proof of Theorem 5, so there cannot be a $CCMG_2$ that defines C. □

Adding universal quantification to type 1 CCMGs also extends their power. It allows the specification of the language G of complete graphs. This language is not in $CCMG_2^{\forall}$ ibecause the connectedness of edges cannot be expressed despite of universal quantification.

Theorem 7. $\mathcal{L}(CCMG_2^{\forall}) \subsetneq \mathcal{L}(CCMG_1^{\forall})$.

Proof. The inclusion follows from the definition and we shall show that $CCMG_1^{\forall}$ s are expressive enough to define G while $CCMG_2^{\forall}$ s are not. The following $CCMG_1^{\forall}$ specifies the complete graphs, G:

$$
\begin{array}{ll}
start & \leftarrow point, \overline{start} \\
\overline{start} & \leftarrow \overline{point}, cont \,||\, \overline{point}.pos = cont.pos \\
\overline{start} & \leftarrow \varepsilon \\
point, cont & \leftarrow \overline{point}, cont, line \,||\, point.pos = \overline{point}.pos,\ line.start \\
 & \quad = \overline{point}.pos,\ line.end = cont.pos \\
cont & \leftarrow res \,||\, \nexists \overline{point} \\
\overline{point}, res & \leftarrow point, res \,||\, point.pos = \overline{point}.pos \\
res & \leftarrow \overline{start} \,||\, \nexists \overline{point}
\end{array}
$$

This works in the following way: in every derivation stage the current graph contains a number of nodes (objects of type *point*). Only the first point is added directly. Every subsequent point is first added as an object of a different type (a $\overline{point}$). For each new node ($\overline{point}$) added, the productions of *cont* iterate through all nodes and insert lines connecting them to the new node whose coordinates are stored in the attributes of *cont*. Once this is done for all nodes and thus all *point*s have been changed into $\overline{point}$s, the productions of *res* change all nodes back to their original type (*point*). Then a new cycle can start.

Universal quantification alone does not facilitate the expression of constraints that define the connectedness of edges and thus the language G of complete graphs cannot even be defined with a context-free CCMG if we add universal quantification.

In the proof of Theorem (5) we have shown that the connectedness of edges to nodes cannot be tested in context-free form. We now show that we also cannot use a universally quantified formula in a $CCMG_2^{\forall}$ to test whether proper connections exist. Assume this could be done; then this condition has to be tested in some formula of the form $\forall X{:}\overline{type}\ldots{:}\varphi(\ldots)$, where $\overline{type}$ is the type of some ancestor of an edge or a node to which the relevant coordinates have been passed. The coordinates of these nodes and the coordinates of the lines to be tested must be given as arguments to φ. No two lines in the sentence have the same pair of adjoining nodes. Since G is complete and the number of arguments to φ is fixed, for some instance $X{:}\overline{type}$ such that $\varphi(X.attrs, x_1, \ldots, x_n)$ there is an instance $X'{:}\overline{type}$ such that $\neg\varphi(X'.attrs, x_1, \ldots, x_n)$. So the connectedness of lines in G cannot be specified by a universally quantified formula. □

Both type 1 and ∀ type 1 CCMGs, can define the connectedness criteria for a graph, but a $CCMG_1$ cannot specify the completeness criteria, because it corresponds to a universally quantified condition over tuples ($\forall x \forall y : connected(x, y)$).

Theorem 8. $\mathcal{L}(CCMG_1) \subsetneq \mathcal{L}(CCMG_1^{\forall})$.

Proof. The inclusion follows from the definition. We have already shown that for the language G of complete graphs $G \in \mathcal{L}(CCMG_1^{\forall})$. The universal quantification is needed in order to test the connectedness of edges since even a $CCMG_1$ still has a grammar-dependent bound on the information that can be used at any node in a derivation. This is because the non-local references allowed by existential quantification can only add a *constant* number of additionally accessible attributes to a node. Thus, G cannot be defined by a $CCMG_1$.

Figure 2 shows a context-sensitive derivation tree. The dotted lines indicate the context connections added by the existential quantification. The amount of information passed through a certain node is still bounded by a grammar-dependent constant k. In addition, there can be up to e existentially quantified symbols used in every production, adding up to e dotted lines to a node. Thus, the total amount of information used at a node cannot be larger than $O((1 + e)k)$. Given a complete graph of size n, we know that each line in L has to be checked against n points in R. (If all the lines are contained in R we swap L and R). Since n is not bounded by the grammar, this cannot be done. □

Theorem 9. $\mathcal{L}(CCMG_1) \not\supset \not\subset \mathcal{L}(CCMG_2^{\forall})$.

Proof. In Theorem 7 we have shown that complete graphs are not in $\mathcal{L}(CCMG_2^{\forall})$. This directly extends to unrestricted graphs, as well. Thus $\mathcal{L}(CCMG_1) \not\subset \mathcal{L}(CCMG_2^{\forall})$. The proof that complete graphs are not in $\mathcal{L}(CCMG_1)$ used in Theorem 8 directly shows that no condition quantified over tuples can be specified with a $CCMG_1$ when the number of tuples is bounded only by the size of the input. Thus, the language of disjoint circles is not in $\mathcal{L}(CCMG_1)$. Since it is in $CCMG_2^{\forall}$, $\mathcal{L}(CCMG_2^{\forall}) \not\subset \mathcal{L}(CCMG_1)$. □

Adding universal quantification to type 0 CCMGs greatly increases their power, in fact they become *computationally adequate*. Thus, we have the following.

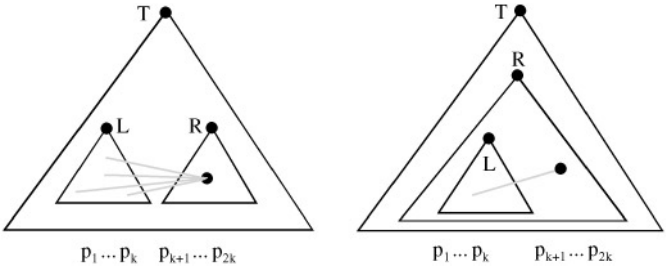

Figure 2. Two $CCMG_1$ derivation structures

Theorem 10. $\mathcal{L}(CCMG_0) \subsetneq \mathcal{L}(CCMG_0^{\forall})$ *and* $\mathcal{L}(CCMG_1^{\forall}) \subsetneq \mathcal{L}(CCMG_0^{\forall})$.

We conjecture that we gain further expressiveness, when the monotonicity criterion for type 1 grammars is dropped, because derivations do not have to be bounded in the input length for type 0 grammars. We believe that, as for conventional string grammars, there are languages that require derivations of unbounded length and which can be defined by a $CCMG_0$. However, it is not easy to find such languages. Assuming they exist, $\mathcal{L}(CCMG_1^{\forall})$ and $\mathcal{L}(CCMG_0)$ are incomparable classes of languages:

Conjecture 1. $\mathcal{L}(CCMG_1^{\forall}) \not\supset \not\subset \mathcal{L}(CCMG_0)$.

Our reasons for believing this conjecture follow. Let G be the language of complete graphs; then $\mathcal{L}(CCMG_0) \not\ni G \in \mathcal{L}(CCMG_1^{\forall})$. We believe that the other direction of the conjecture can be shown using the language R that consists of all sets of axis-parallel lines which form a rectangle that is completely filled by other rectangles. The lines in the input sentence may be, but need not be, segmented at points where two segments meet in a right angle. $R \in \mathcal{L}(CCMG_0)$. R can obviously be generated by the $CCMG_0$ sketched in Figure 3. We believe that $R \notin \mathcal{L}(CCMG_1^{\forall})$. This is because some derivations need to 'construct' a number of merged line segments that exceeds the size of the input. Therefore, it is highly unlikely that R can be specified by a $CCMG_1^{\forall}$.

The following conjecture essentially completes the hierarchy. Its validity depends only on Conjecture 1. Assuming that Conjecture 1 holds, there must be languages in $\mathcal{L}(CCMG_0)$ that are not in $\mathcal{L}(CCMG_1)$ and thus $\mathcal{L}(CCMG_0)$ must be a proper superclass of $\mathcal{L}(CCMG_1)$.

Conjecture 2. $\mathcal{L}(CCMG_1) \subsetneq \mathcal{L}(CCMG_0)$.

Proof. $\mathcal{L}(CCMG_0) \supset \mathcal{L}(CCMG_1)$ by definition. $\mathcal{L}(CCMG_1^{\forall}) \not\ni R \in \mathcal{L}(CCMG_0)$. $\mathcal{L}(CCMG_1^{\forall}) \supset \mathcal{L}(CCMG_1)$. Thus $R \notin \mathcal{L}(CCMG_1)$. □

All that remains is to consider grammars in which the context can be specified with a full first-order logical formula. Surprisingly this does not lead to greater expressiveness.

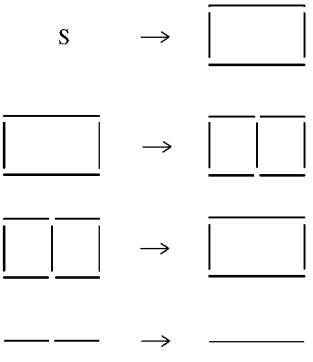

Figure 3. Tiling grammar

Theorem 11. $\mathcal{L}(CCMG_0^{FOL}) = \mathcal{L}(CCMG_0^{\forall})$.

Proof. We observe that *leading existential quantification* is almost identical to using context symbols. The only difference is that context symbols must be distinct from other symbols in the production, while existentially quantified variables can refer to symbols that are already used. It is therefore possible to remove a leading existential quantifier from a production $v \leftarrow w \| (\exists X{:}t)\ \varphi$ by replacing this production with the production $X{:}t,\ v \leftarrow X{:}t,\ w \| \varphi$ and productions $v \leftarrow w \| \varphi\{X/Y_i\}$ for all $Y_1, \ldots, Y_k$ that are the symbols of type t in v. *Leading universal quantification* in a production can be removed by replacing such a production with a 'proof procedure' that iteratively tests the universally quantified property for every symbol of the appropriate type.

This leads to the following schema of grammar transformation: (1) transform the constraint formula in each production into prenex normal form, (2) rewrite leading existential quantifiers as context symbols, (3) replace leading universal quantifiers by a 'proof procedure' and (4) iterate these steps until no existential quantification within the scope of a universal quantification is left.

We will illustrate the mechanism of this proof procedure only with a simplified example, since the full transformation is technically rather involved. The following example should suffice to understand the general transformation schema.

Consider the production $g{:}graph \leftarrow g{:}graph,\ p_1{:}point \| (\forall p_2{:}point\ \exists l{:}line)\ connected(p_1, p_2, l)$ which might be used in a grammar for complete graphs. We eliminate the leading universal quantifier by removing this production and substituting it in the following way:

1. Add $g_1{:}graph \leftarrow g_2{:}graph,\ p_1{:}point \| g_2.state = p_1.state = 1 \wedge p_1.marked_1 = false \wedge g_1.state = 0 \wedge \omega(g_1, g_2)$ to the production set where ω is a function transferring all the attributes except for *state* between its arguments. All productions in the grammar are stopped from interfering with this production by adding a constraint to them which tests that for all symbols s, $s.level \neq 0$. This production is only used to set the derivation into a state in which exclusively the subsequent productions apply. $marked_1$ is a new attribute used as a flag to indicate whether the quantified property has already been tested for the respective symbol.
2. Add $g_1{:}graph,\ p_1, p_2{:}point \leftarrow g_1{:}graph,\ p_1, p_3{:}point \| (\exists l{:}line)\ connected(p_1, p_2, l) \wedge p_3.marked_1 = true \wedge p_2.marked_1 = false \wedge g_1.state = p_1.state = 1 \wedge \omega(p_2, p_3)$ to the production set. This is the core of the proof procedure, which iterates over all points explicitly to test the quantified property.
3. Add $g_1{:}graph,\ p_1{:}point \leftarrow g_2{:}graph,\ p_2{:}point \| (\nexists p_3{:}point)\ (p_3.marked_1 = false \wedge p_3.state \neq 1) \wedge g_2.state = p_2.state = 2 \wedge g_1.state = p_1.state = 1 \wedge \omega(g_1, g_2) \wedge \omega(p_1, p_2)$ to the production set. This production terminates the proof after all relevant symbols have been tested. Augment all productions with a constraint which tests that for all symbols s, $s.level \neq 2$.
4. Add $g_1{:}graph,\ p_1, p_2{:}point \leftarrow g_1{:}graph,\ p_1, p_3{:}point \| p_3.marked_1 = false \wedge p_2.marked_1 = true \wedge g_1.state = p_1.state = 2 \wedge \omega(p_2, p_3)$ to the production set.
5. Add $g_1{:}graph,\ p_1{:}point \leftarrow g_2{:}graph,\ p_2{:}point \| (\nexists p_3{:}point)\ p_3\ marked_1 = true \wedge g_1.state = p_1.state = 2 \wedge g_2.state = p_2.state = 0 \wedge \omega(g_1, g_2) \wedge \omega(p_1, p_2)$ to the production set. The last two productions reset the state of all symbols to the original state.

Table 1

	$\mathcal{L}_3$	$\mathcal{L}_2$	$\mathcal{L}_1$	$\mathcal{L}_0$	$\mathcal{L}_{2\forall}$	$\mathcal{L}_{1\forall}$	$\mathcal{L}_{0\forall}$
$\mathcal{L}_3$	$=$	$\subset / T4$	$\subset$	$\subset$	$\subset$	$\subset$	$\subset$
$\mathcal{L}_2$	$\supset / T4$	$=$	$\subset / T5$	$\subset$	$\subset / T6$	$\subset$	$\subset$
$\mathcal{L}_1$	$\supset$	$\supset / T5$	$=$	$\subset / C2$	$\not\supset \not\subset / T9$	$\subset / T8$	$\subset$
$\mathcal{L}_0$	$\supset$	$\supset$	$\supset / C2$	$=$	$-$	$\not\supset \not\subset / C1$	$\subset / T10$
$\mathcal{L}_{2\forall}$	$\supset$	$\supset / T6$	$\not\supset \not\subset / T9$	$-$	$=$	$\subset / T7$	$\subset$
$\mathcal{L}_{1\forall}$	$\supset$	$\supset$	$\supset / T8$	$\not\supset \not\subset / C1$	$\supset / T7$	$=$	$\subset / T10$
$\mathcal{L}_{0\forall}$	$\supset$	$\supset$	$\supset$	$\supset / T10$	$\supset$	$\supset / T10$	$=$

Note that in subsequent transformation steps the integers assigned to *state* and $marked_i$ would change according to the level of the inductive transformation. Additional productions are used to remove these attributes from the terminal tokens in a sentence, in order to generate the original sentences.

Strictly speaking the new productions above are not copy-restricted, since they assign the new attribute values *true*, *false* and the integer values representing levels to some attributes. However, this is a fixed number of different attribute values determined by the grammar and not by the input sentence. It is therefore possible to unfold the attribute values directly into the token types by replacing, for example, a token of the new type x_{state0} for every token of type x with an attribute $state = 0$ and changing the productions accordingly. All productions at the end of the transformation have the form required in a $CCMG_0^{\forall}$. □

Theorem 12. $\mathcal{L}(CCMG_1^{FOL}) = \mathcal{L}(CCMG_1^{\forall})$.

Proof. The same technique as in the proof for Theorem 11 above can be used. Since the modifications do not introduce any shortening productions if the original productions are non-shortening, the transformation generates a $CCMG_1^{\forall}$ from a $CCMG_1^{FOL}$. □

We finally summarize our expressiveness results for the CCMG hierarchy in Table 1. $\mathcal{L}_x$ is used as an abbreviation for $\mathcal{L}(CCMG_x)$. The symbols $\subset$ and $\supset$ indicate *proper* set inclusions. An entry for row x and column y of the form op/Tn or op/Cn has to be interpreted as $x\ op\ y$ by Theorem n (or by Conjecture n). If no theorem or conjecture is indicated, the result follows immediately from the other entries.

6. Complexity of Parsing

Given a sentence, S, and grammar, G, the *membership problem* is to determine if $S \in \mathcal{L}(G)$. In this section we investigate the inherent complexity of the membership problem for the various classes in the hierarchy.

For a particular class of grammars $\mathcal{G}$, there are really two complexity questions we can ask:

- What is the complexity of the membership problem for all $G \in \mathcal{G}$ and for all sentences S?
- For a fixed grammar $G \in \mathcal{G}$, what is the complexity of the membership problem for G for all sentences S?

The first question investigates the complexity in terms of the size of the grammar and sentence, while the second is only concerned with the size of the sentence. In this paper we will consider the first question, since this is arguably the more fundamental question. Throughout this section we assume that constraint testing takes polynomial time.

It is well known that the membership problem for regular and context-free string grammars has polynomial complexity. Somewhat surprisingly, it is much more expensive to solve the membership problem for regular and context-free CCMGs. Indeed, most classes in the hierarchy display the same (high) complexity which is the same as the complexity of the membership problem for context-sensitive string grammars. Indeed, we prove this by encoding a context-sensitive string grammar as a regular CCMG.

Theorem 13. *Let $\mathcal{G}$ be one of $CCMG_3$, $CCMG_2$, $CCMG_1$, $CCMG_1^{\forall}$, or $CCMG_1^{FOL}$. The complexity of the membership problem for $\mathcal{G}$ is PSPACE-complete.*

The proof follows from the following two lemmas. We first show that the membership problem for $CCMG_3$ is PSPACE-hard. It follows that the membership problem for the other classes is also PSPACE-hard.

Lemma 4. *For $CCMG_3$ the membership problem is PSPACE-hard.*

Proof. The proof is based on the membership problem for context-sensitive string grammars which is known to be PSPACE-hard. We show how to encode an arbitrary context-sensitive string grammar G' and string S' into a $CCMG_3$ G and a sentence S such that $S \in \mathcal{L}(G)$ iff $S' \in \mathcal{L}(G')$. Since the membership problem for arbitrary context-sensitive string grammars is PSPACE-hard and the size of G and S is polynomial in the size of G' and S', it follows that the membership problem for $CCMG_3$ is PSPACE-hard.

The encoding of the (arbitrary) string S' is as a single token called *Str* which essentially has an attribute encoding each symbol in the string. Let S be the singleton multiset containing the token of type *Str* which encodes the string S'.

The encoding of the (arbitrary) productions in G' is to productions in G which reduce a *Str* token to another *Str* token.

The precise encoding is as follows. Let the initial string S' have n symbols. There are two types of tokens in G:

- The start token *start* which has no attributes.
- A *Str* type which represents a string of length n or less. Str has $3n$ attributes. For $i = 1, \ldots, n$ there is a $type_i$ attribute which is the name of the ith symbol in S and a lhs_i

and rhs_i attribute which is an integer such that adjacent tokens have the same *lhs* and *rhs* value. Conventions are that the name of the start symbol in G' is 1 and that if $type_i = lhs_i = rhs_i = 0$ this represents a null in the string.

For instance, the string aaX might be represented by the token *Str* where

$$\begin{array}{lll} Str.type_1 = 1 & Str.lhs_1 = 3 & Str.rhs_1 = 4 \\ Str.type_2 = 0 & Str.lhs_2 = 0 & Str.rhs_2 = 0 \\ Str.type_3 = 2 & Str.lhs_3 = 1 & Str.rhs_3 = 2 \\ Str.type_4 = 2 & Str.lhs_4 = 2 & Str.rhs_4 = 3 \end{array}$$

where X the start symbol is represented by 1 and a by 2.

Given the initial string $S' = a_1 \ldots a_n$ we therefore encode it as the terminal sentence $\{Str\}$ where for each $i = 1, \ldots, n$, $Str.type_i = a_i$, $Str.lhs_i = i$ *and* $Str.rhs_i = i + 1$.

The productions in the grammar G are as follows. The first production generates the *Str* representing the start symbol.

$$\begin{array}{l} start \leftarrow Str \| \\ Str.type_1 = 1, \\ Str.lhs_1 = 1, \\ Str.rhs_1 = n + 1 \end{array}$$

The next $n-1$ productions allow us to swap the representation of two symbols in *Str*. Note that this swap does not change the actual string represented by *Str*:

$$\begin{array}{ll} Str' \leftarrow Str \| & Str' \leftarrow Str \| \\ true \wedge & true \wedge \\ Str'.type_1 = Str.type_2, & Str'.type_{n-1} = Str.type_n, \\ Str'.lhs_1 = Str.lhs_2, & Str'.lhs_{n-1} = Str.lhs_n, \\ Str'.rhs_1 = Str.rhs_2, & Str'.rhs_{n-1} = Str.rhs_n, \\ Str'.type_2 = Str.type_1, & Str'.type_n = Str.type_{n-1}, \\ Str'.lhs_2 = Str.lhs_1, \quad \ldots & Str'.lhs_n = Str.lhs_{n-1}, \\ Str'.rhs_2 = Str.rhs_1, & Str'.rhs_n = Str.rhs_{n-1}, \\ \forall i \in \{2, \ldots, n\}, & \forall i \in \{1, \ldots, n-2\}, \\ \quad Str'.lhs_i = Str.lhs_i, & \quad Str'.lhs_i = Str.lhs_i, \\ \quad Str'.rhs_i = Str.rhs_i, & \quad Str'.rhs_i = Str.rhs_i, \\ \quad Str'.type_i = Str.type_i & \quad Str'.type_i = Str.type_i \end{array}$$

Finally, for each production P' in G', of form $L_1, \ldots, L_m \leftarrow R_1, \ldots, R_m, R_{m+1}, \ldots, R_p$ say, there is a production

$$\begin{array}{l} \quad Str' \leftarrow Str \| \\ \forall i \in \{1, \ldots, p-1\}, \\ \quad Str.rhs_i = Str.lhs_{i+1} \\ \forall i \in \{1, \ldots, p\}, \\ \quad Str.type_i = R_i \wedge \\ \forall i \in \{1, \ldots, m-1\} \\ \quad Str'.type_i = L_i \end{array}$$

$$
\begin{aligned}
&\quad Str'.lhs_i = Str.lhs_i \\
&\quad Str'.rhs_i = Str.rhs_i \\
&Str'.type_m = L_m \\
&Str'.lhs_m = Str.lhs_m \\
&Str'.rhs_m = Str.rhs_p \\
&\forall i \in \{m+1, \ldots, p\} \\
&\quad Str'.type_i = 0 \\
&\quad Str'.lhs_i = 0 \\
&\quad Str'.rhs_i = 0 \\
&\forall i \in \{p+1, \ldots, n\}, \\
&\quad Str'.lhs_i = Str.lhs_i \\
&\quad Str'.rhs_i = Str.rhs_i \\
&\quad Str'.type_i = Str.type_i
\end{aligned}
$$

The production mimics the operation of P' on a string. The first part of the production (the first two for loops) ensures that *Str* contains the right-hand side of P', the second part ensures that *Str'* contains the left-hand side of P together with the appropriate padding of nulls, while the last part (the final for loop) ensures that the remainder of the string is left unchanged.

As an example, consider the case when S' is the string aa and G' is the context-sensitive string grammar with two productions $X \to aX$ and $aX \to aa$ where a is a terminal symbol and X the start symbol. Since S' has two symbols the grammar will be defined over the single token *Str* with 6 attributes. As before, we let 1 represent X and 2 represent a. The four productions in the grammar G simulating G' are:

$$
\begin{aligned}
&&&Str' \leftarrow Str || \\
&&&true \wedge \\
&start \leftarrow Str || &&Str'.type_1 = Str.type_2, \\
&\quad Str.type_1 = 1, &&Str'.lhs_1 = Str.lhs_2, \\
&\quad Str.lhs_1 = 1, &&Str'.rhs_1 = Str.rhs_2, \\
&\quad Str.rhs_1 = n+1 &&Str'.type_2 = Str.type_1, \\
&&&Str'.lhs_2 = Str.lhs_1, \\
&&&Str'.rhs_2 = Str.rhs_1
\end{aligned}
$$

$$
\begin{aligned}
&Str' \leftarrow Str || &&Str' \leftarrow Str || \\
&\quad Str.rhs_1 = Str.lhs_2 &&Str.rhs_1 = Str.lhs_2 \\
&\quad Str.type_1 = 2 &&Str.type_1 = 2 \\
&\quad Str.type_2 = 1 \wedge &&Str.type_2 = 2 \wedge \\
&\quad Str'.type_1 = 1 &&Str'.type_1 = 2 \\
&\quad Str'.lhs_1 = Str.lhs_1 &&Str'.lhs_1 = Str.lhs_1 \\
&\quad Str'.rhs_1 = Str.rhs_2 &&Str'.rhs_1 = Str.rhs_1 \\
&\quad Str'.type_2 = 0 &&Str'.type_2 = 1 \\
&\quad Str'.lhs_2 = 0 &&Str'.lhs_2 = Str.lhs_2 \\
&\quad Str'.rhs_2 = 0 &&Str'.rhs_2 = Str.rhs_2
\end{aligned}
$$

The top-left production generates the start symbol, the top-right production swaps the representation of tokens in the string, the bottom-left production encodes $X \to aX$

and the bottom-right production encodes $aX \rightarrow aa$. The initial string aa will be encoded as the sentence $\{Str\}$ where

$$Str.type_1 = 2 \quad Str.lhs_1 = 1 \quad Str.rhs_1 = 2$$
$$Str.type_2 = 2 \quad Str.lhs_2 = 2 \quad Str.rhs_2 = 3$$

The careful reader may have noticed that our grammar is not, in fact, strictly copy-restricted, since we assign a 0 to a $Str'.type_i$, $Str'.lhs_i$ and $Str'.rhs_i$. However, since this is a single fixed constant, it is easy to transform this into a copy-restricted grammar, say with the convention that $Str'.type_0 = Str'.lhs_0 = Str'.rhs_0 = 0$ which allows us to copy the 0 from this attribute. □

We now prove that the membership problem is in PSPACE for $CCMG_1^{FOL}$ and hence for the other classes.

Lemma 5. *For $CCMG_1^{FOL}$ the membership problem is in PSPACE.*

Proof. It suffices to show that the membership problem is in NPSPACE since PSPACE = NPSPACE. Imagine we are trying to determine if $S \in \mathcal{L}(G)$. Our nondeterministic algorithm just guesses each sentence in the derivation from S to the start symbol together with the production used and the mapping from symbols in the production to those in the sentence. At each step the algorithm verifies that this is a valid derivation by checking the constraints. For universally quantified tokens it does this by guessing an assignment to the tokens which does not satisfy the constraint. At each stage only two sentences need to be remembered. As the size of each of these sentences is less than or equal to S and the size of the production is less than or equal to G, the algorithm takes only polynomial space. □

Now, we consider the complexity of the membership problem for unrestricted CCMGs. Surprisingly, it is decidable which contrasts with unrestricted string grammars, for which it is undecidable. Unsurprisingly, it is a very hard problem—at least EXP-SPACE hard.

A constraint multiset grammar in which no symbols have attributes is said to be *propositional*. Clearly, any propositional CMG is also copy-restricted. The membership problem for propositional $CCMG_0$ has been previously studied under the name of the word problem for commutative Semi-Thue Systems. This problem is known to be decidable and EXP-SPACE hard [20]. This result can be used to prove the following theorems.

Theorem 14. *The membership problem for $CCMG_0$ is decidable.*

Proof. Consider the sentence S and $CCMG_0$ grammar G. Let the attributes occurring in S be A. Let S' be the set of tokens which can be constructed from the types in G using the attributes in A. Clearly S' is finite. We can map each element t of S' to a unique propositional token $\iota(t)$. Now consider a production P in G. We can map it to a set of productions $\iota(P)$ which rewrite a sentence of propositional tokens to a new sentence of propositional tokens iff the corresponding tokens in S' can be rewritten with G. Note

that $\iota(P)$ is finite. Let G' be the set of $\iota(P)$ s where P is a production in G and S' the propositional tokens encoding S. Then we have $S \in \mathscr{L}(G)$ iff $S' \in \mathscr{L}(G')$. As the membership problem for propositional $CCMG_0$ grammars is decidable, it is decidable if $S' \in \mathscr{L}(G')$, so $S \in \mathscr{L}(G)$ is decidable. □

Theorem 15. *The membership problem for $CCMG_0$ is EXP-SPACE hard.*

Proof. This follows from the EXP-SPACE hardness of the membership problem for propositional $CCMG_0$ grammars. □

The following theorem holds because the membership problem for propositional $CCMG_0^{\forall}$ grammars is undecidable—it is easy to simulate a two-counter machine with them.

Theorem 16. *Let $\mathscr{G}$ be either $CCMG_0^{\forall}$ or $CCMG_0^{FOL}$. The membership problem for $\mathscr{G}$ is undecidable.*

7. Conclusion and Extensions

We have developed a hierarchical classification of visual languages based on CMGs. We have investigated the expressiveness and the cost of parsing for the classes in the hierarchy. We have also illustrated how other formalisms can be mapped into CMGs and hence into the hierarchy. Our hierarchy thus provides a ways to classify visual languages according to their computational properties and compare the expressiveness of visual language specification methods.

One consequence of our results concerning expressiveness is that context-sensitive specifications are important for visual languages. We note that, although the majority of textual computer languages are context-free, this is *because* they have been *designed* to be context-free. For the same reason, although many of the visual languages that have been designed for computer usage are context-free, we feel this is because most current visual language specification methods primarily cater for context-free languages, causing language designers to use only context-free syntactic constructions. However, a prime objective of visual language research is to allow intuitive, easy-to-use visual languages, and to make existent visual notations that have been developed outside of computer science, e.g. in architecture, available in computer-based environments. We feel this necessarily includes visual languages that are not context-free. Some compelling examples of such visual languages are graph-based languages such as Petri nets and state-transition diagrams. These are widely used, yet since they are based on graphs, cannot be specified by any copy-restricted context-free grammar.

Our results on complexity seem somewhat depressing as they indicate that parsing for even the most restrictive class in the hierarchy is very expensive. This, however, is not necessarily true since a theoretical $\mathscr{NP}$-hardness result does not automatically render a method unusable in practice. After all, humans manage to understand diagrams very quickly.

One promising approach is to consider restrictions which make the underlying formalism radically more efficient, yet still allow the specification of 'natural' visual languages. The main reason for the high complexity of parsing is non-determinism. If grammars are required to be confluent (in the standard sense of confluence for rewrite systems), then, under reasonable assumptions, the cost of parsing for a given monotonic CCMG becomes polynomial. Prototypes of applications that use confluent context-sensitive CCMGs [21] have shown that real-time editing behavior is possible for a variety of visual languages.

The other complexity result which may be of concern is that parsing for the universally quantified type 0 case is undecidable. This is because we can introduce multiple occurrences of the same symbol with the same attribute values. In a sense this is rather artificial, since it allows us to have multiple objects which are geometrically indistinguishable. If instead we do not allow copies of a symbol with the same attribute values, then the membership problem is decidable for all elements of the hierarchy.

The use of copy-restricted grammars as the basis for our hierarchy is apparently a rather severe restriction. Though most practical CMGs actually adhere to the copy restriction, it sometimes makes specifications harder to write and to understand. We are therefore looking for ways to reintroduce limited forms of attribute computation without flattening the hierarchy. This, however, requires great care since simple extensions like introducing addition and subtraction can make parsing undecidable, and allowing accumulating data types such as sets or lists can make it possible to simulate arbitrary positive and negative contexts.

Nonetheless, in practical applications, attribute computation may well be useful and inexpensive to integrate. It seems particularly natural to look at the integration of intervals and interval operations like *union*, *intersection*, *difference*, since these operations are commonly used in bounding-box calculations. The first observation to be made is that one- and two-dimensional continuous intervals with the usual interval operations do not change our hierarchy at all. They can obviously already be simulated with copy-attributes, as we have demonstrated for bounding-boxes in Section 3. The analogous case for non-continuous regions is less immediate, but it can also be simulated if the underlying domain is finite. However, it is not clear what happens to the hierarchy.

The investigation of various kinds of attribute computations seems to bring the analysis of visual languages closer to the intuitively perceived visual phenomenon, since they lie at the heart of handling spatial relationships. By making spatial properties such as, e.g the aggregation of regions, directly expressible we come closer to the normal way of thinking about diagrams. The most important domain to explore in more depth are continuous regions (point sets) with union and intersection operations serving as a natural non-rectangular generalization of bounding-boxes.

References

1. A. Rosenfeld (1976) Array and web grammars: an overview. In: *Automata*, *Languages*, *and Development* (A. Lindenmayer & G. Rosenberg, eds). North-Holland, Amsterdam, pp. 517–529.
2. J. E. Gips (1974) Shape grammars and their uses. Ph.D. thesis, Stanford University.

3. G. Costagliola, S. Orefice, G. Polese, G. Tortora & M. Tucci (1993) Automatic parser generation for pictorial languages. In: *IEEE Symposium on Visual Languages*, pp. 306–313.
4. F. Ferrucci, G. Tortora, M. Tucci & G. Vitiello (1994) A predictive parser for visual languages specified by relation grammars. In: *IEEE Symposium on Visual Languages*, pp. 245–252.
5. K. Wittenburg & L. Weitzmann (1990) Visual grammars and incremental parsing for interface languages. In: *IEEE Symposium on Visual Languages*, pp. 111–118.
6. K. Wittenburg (1993) Adventures in multidimensional parsing: cycles and disorders. In: *International Workshop on Parsing Technologies*, pp. 333–348.
7. K. Wittenburg (1996) Predictive parsing for unordered relational languages. In: *Recent Advances in Parsing Technologies*, to appear.
8. E. Golin & S. P. Reiss (1989) The specification of visual language syntax. In: *IEEE Symposium on Visual Languages*, pp. 105–110.
9. K. Marriott (1994) Constraint multiset grammars. In: *IEEE Symposium on Visual Languages*, pp. 118–125.
10. B. Courcelle (1990) Graph rewriting: an algebraic and logic approach. In: *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.). Elsevier, Amsterdam.
11. S. Üsküdarli (1994) Generating visual editors for formally specified languages. In: *IEEE Symposium on Visual Languages*, pp. 278–287.
12. R. Helm & K. Marriott (1991) A declarative specification and semantics for visual languages. *Journal of Visual Languages and Computing* **2,** 311–331.
13. B. Meyer (1992) Pictures depicting pictures. In: *IEEE Symposium on Visual Languages*, pp. 41–47.
14. V. Haarslev (1995) Formal semantics of visual languages using spatial reasoning. In: *IEEE Symposium on Visual Languages*, pp. 156–163.
15. K. Marriott & B. Meyer (eds) (1997) *Theory of Visual Languages*. Springer, New York, forthcoming.
16. N. Abe, M. Mizumoto, J.-I. Toyoda & K. Tanaka (1973) Web grammars and several graphs. *Journal of Computer and System Sciences* 7.
17. R. Helm, K. Marriott & M. Odersky (1991) Building visual language parsers. In: *ACM Conf. Human Factors in Computing*, pp. 105–112.
18. M. Tucci, G. Vitiello, G. Pacini & G. Tortora (1992) Graphs and visual languages for visual interfaces. In: *Advanced Visual Interfaces* (*AVI 92*). World Scientific, Singapore.
19. E. J. Golin (1991) A method for the specification and parsing of visual languages. Ph.D. thesis, Brown University.
20. E. W. Mayr (1984) An algorithm for the general petri net reachability problem. *SIAM Journal of Computing* 13.
21. S. S. Chock & K. Marriott (1995) Automatic construction of user interfaces from constraint multiset grammars. In: *IEEE Symposium on Visual Languages*, pp. 242–249.