Specifying Visual Languages with Conditional Set Rewrite Systems

Marc A. Najork* Simon M. Kaplan

Department of Computer Science
University of Illinois
1304 West Springfield Avenue
Urbana, IL 61801, U. S. A.
{najork,kaplan}@cs.uiuc.edu

Abstract

We propose Conditional Set Rewriting as a general mechanism for describing the syntax of multidimensional languages. We compare the approach with other existing methods, and give a number of examples that illustrate its strengths.

1 Introduction

This paper introduces the notion of a *Conditional Set Rewrite System (CSRS)*. Conditional Set Rewrite Systems are a generalization of Conditional Term Rewrite Systems [10]. Conditional Term Rewrite Systems deal with rewriting a single term, whereas Conditional Set Rewrite systems deal with rewriting a set of terms. Each conditional set rewrite rule specifies how to replace a subset of the set of terms by another set.

We suggest two uses for CSRS: they can be used to describe the set of all legal phrases (pictures) of a multidimensional (visual) language, and they can be used to translate a phrase from one language (e.g. a visual one) into another language (e.g. a textual one).

The ability of CSRS to translate visual programs into textual ones allows us to utilize many results from the semantics of textual languages. We can formalize the semantics of a visual language by giving a translation from the visual language to a textual one, and then give type inference rules and an operational or denotational semantics for the textual language.

There are a variety of grammar-based specification techniques for visual languages, among them Picture Processing Grammars [1], Relation Grammars [3], Positional Grammars [2], Fringe Relational Grammars [11], and Picture Layout Grammars.

Picture Layout Grammars [6, 5] are one of the most flexible of these grammars. They are based on *Attributed Multiset Grammars* [4], which are similar to textual context-free grammars, except that the right-hand side of a production is an unordered collection of symbols rather than a string. Picture Layout Grammars have been widely used to describe the syntax of two-dimensional visual languages.

Helm and Marriott [8, 9] have introduced a declarative specification technique for visual languages, based on the constraint logic programming language CLP($\mathcal{R}$) [7]. For lack of a better term, we will refer to their framework as to "*Logic Grammars*".

The remainder of this paper consists of three parts: first, we will define CSRS; then, we will argue that CSRS are a generalization of Picture Layout Grammars and of Logic Grammars; and finally, we will give a number of examples that demonstrate their expressiveness.

2 Conditional Set Rewrite Systems

Informally, a CSRS consists of an ordered sequence of rewrite rules, which are *guarded* by a condition. Conditions are predicate applications, closed over conjunction and disjunction. Predicates are defined through Horn clauses. The syntax of CSRS is as follows:

$t ::= x \mid k(t_1, \cdots, t_n)$	(term)
$\phi ::= P(t_1, \cdots, t_n) \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$	(formula)
$\delta ::= P(t_1, \cdots, t_n) \Leftarrow \phi$	(pred. def.)
$\rho ::= t_1, \cdots, t_m \rightarrow t'_1, \cdots, t'_n$ **if** ϕ	(rule)
$\Sigma ::= (\rho_1 \cdots \rho_m, \delta_1 \cdots \delta_n)$	(system)

A *term* is either a *variable* or a *constructor* applied to some terms. A *formula* is either a *predicate symbol* applied to some terms, or the *conjunction* or *disjunction* of two formulas. A *predicate definition* defines $P(t_1, \cdots, t_n)$ to hold if the formula ϕ holds. A *rewrite rule* replaces the terms $t_1, \cdots, t_m$ in a set σ by terms $t'_1, \cdots, t'_n$ if ϕ holds. A *conditional set rewrite system* consists of an ordered sequence of rewrite rules, and a set of predicate definitions.

*Supported by the National Science Foundation under grant CCR-9007195

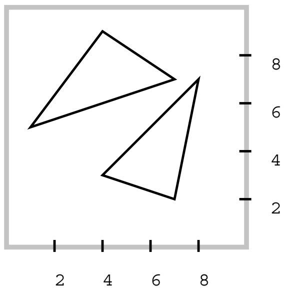

Figure 1: Triangles

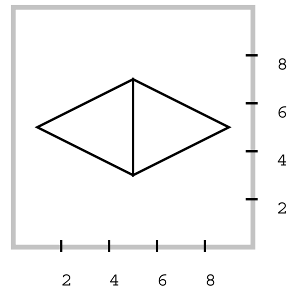

Figure 2: Ambiguous picture

Given a set of terms σ and a rewrite rule $t_1, \cdots, t_m \rightarrow t'_1, \cdots, t'_n$ **if** ϕ, the rule is *applicable* if σ contains terms matching $t_1, \cdots, t_m$, and ϕ holds. *Applying* an applicable rewrite rule means replacing $t_1, \cdots, t_m$ in σ by $t'_1, \cdots, t'_n$. A rewrite step $\sigma \rightarrow \sigma'$ results from applying the *first* applicable rewrite rule to σ. We say that σ_0 *rewrites* to σ_n ($\sigma_0 \rightarrow_* \sigma_n$) if there is a sequence of rewrite steps $\sigma_0 \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_n$. We say that σ is in *normal form* if there is no σ' s.t. $\sigma \rightarrow \sigma'$.

We allow for two notational simplifications: First, instead of $P(t_1, \cdots, t_n) \Leftarrow \mathsf{true}$, we simply write $P(t_1, \cdots, t_n)$ (and similar for $t_1, \cdots, t_m \rightarrow t'_1, \cdots, t'_n$ **if** true). Second, we allow an argument to be a simple function application instead of a term. For instance, we allow $\mathsf{t}(n) \rightarrow \mathsf{t}(n+1)$, which could be expanded to $\mathsf{t}(n) \rightarrow \mathsf{t}(n')$ **if** $\mathsf{plus}(n, 1, n')$.

Example 1: The following CSRS describes a 2D visual language where all the legal pictures contain only triangles:

<u>Rewrite Rules:</u>

$\mathsf{line}(u,v),\ \mathsf{line}(w,x),\ \mathsf{line}(y,z) \rightarrow \mathsf{tri}(a,b,c)$
  if $\mathsf{connected}(\mathsf{line}(u,v),\mathsf{line}(w,x),a)$
  $\wedge\ \mathsf{connected}(\mathsf{line}(u,v),\mathsf{line}(y,z),b)$
  $\wedge\ \mathsf{connected}(\mathsf{line}(w,x),\mathsf{line}(y,z),c)$

$\mathsf{tri}(a,b,c) \rightarrow$

<u>Predicate Definitions:</u>

$\mathsf{connected}(\mathsf{line}(x,y),\mathsf{line}(x,z),x)$

$\mathsf{connected}(\mathsf{line}(y,x),\mathsf{line}(x,z),x)$

$\mathsf{connected}(\mathsf{line}(x,y),\mathsf{line}(z,x),x)$

$\mathsf{connected}(\mathsf{line}(y,x),\mathsf{line}(z,x),x)$

In this particular CSRS, a picture is legal (belongs to the language) if it (or, more precisely, its set representation) can be rewritten to the empty set.

The picture shown in Fig. 1 is represented by the set of terms

$\{$ line(pt(1,5),pt(7,7)), line(pt(7,7),pt(4,9)),
line(pt(4,9),pt(1,5)), line(pt(4,3),pt(7,2)),
line(pt(7,2),pt(8,7)), line(pt(8,7),pt(4,3)) $\}$

One possible rewrite sequence is

$\{$ line(pt(1,5),pt(7,7)), line(pt(7,7),pt(4,9)),
line(pt(4,9),pt(1,5)), line(pt(4,3),pt(7,2)),
line(pt(7,2),pt(8,7)), line(pt(8,7),pt(4,3)) $\} \rightarrow$
$\{$ tri(pt(1,5),pt(7,7),pt(4,9)), line(pt(4,3),pt(7,2)),
line(pt(7,2),pt(8,7)), line(pt(8,7),pt(4,3)) $\} \rightarrow$
$\{$ tri(pt(1,5),pt(7,7),pt(4,9)), tri(pt(4,3),pt(7,2),pt(8,7)) $\} \rightarrow$
$\{$ tri(pt(4,3),pt(7,2),pt(8,7)) $\} \rightarrow \{\ \}$

But note that there are three other sequences which also lead to the empty set. This leads us to

Observation 1: CSRS can be non-deterministic.

In the previous example, there were 4 different reduction sequences leading to a normal form, but they all led to the *same* normal form. The next example shows that this is not always the case:

Example 2: Consider a visual language where a legal picture contains one triangle and one V-shape.

<u>Rewrite Rules:</u>

$\mathsf{line}(u,v),\ \mathsf{line}(w,x),\ \mathsf{line}(y,z) \rightarrow \mathsf{tri}(a,b,c)$
  if $\mathsf{connected}(\mathsf{line}(u,v),\mathsf{line}(w,x),a)$
  $\wedge\ \mathsf{connected}(\mathsf{line}(u,v),\mathsf{line}(y,z),b)$
  $\wedge\ \mathsf{connected}(\mathsf{line}(w,x),\mathsf{line}(y,z),c)$

$\mathsf{line}(u,v),\ \mathsf{line}(w,x) \rightarrow \mathsf{vee}(a,b,c)$
  if $\mathsf{connected}(\mathsf{line}(u,v),\mathsf{line}(w,x),a)$
  $\wedge\ ((a{=}u \wedge b{=}v) \vee (a{=}v \wedge b{=}u))$
  $\wedge\ ((a{=}w \wedge c{=}x) \vee (a{=}x \wedge c{=}w))$

$\mathsf{tri}(a,b,c),\ \mathsf{vee}(d,e,f) \rightarrow \mathsf{pic}(\mathsf{tri}(a,b,c),\ \mathsf{vee}(d,e,f))$

and connected defined as in Example 1

In this CSRS, a picture is legal if it can be rewritten to the set $\{\mathsf{pic}(\mathsf{tri}(a,b,c),\mathsf{vee}(d,e,f))\}$.

Now consider the picture shown in Fig. 2, whose set representation is

{ line(pt(1,5),pt(5,3)), line(pt(5,3),pt(5,7)),
line(pt(5,7),pt(1,5)), line(pt(9,5),pt(5,3)),
line(pt(9,5),pt(5,7)) }

This set can be rewritten to two distinct normal forms,

{pic(tri(pt(1,5),pt(5,3),pt(5,7)),vee(pt(9,5),pt(5,3),pt(5,7)))}

and

{pic(tri(pt(9,5),pt(5,3),pt(5,7)),vee(pt(1,5),pt(5,3),pt(5,7)))},

each one identifying the picture as being legal. This leads us to

Observation 2: CSRS can be non-confluent.

In this respect, CSRS are similar to Picture Layout Grammars and to Logic Grammars, which both allow for ambiguous grammars. In all three frameworks, it is up to the user to ensure nonambiguity by carefully choosing and arranging the productions or rewrite rules. There are no known mechanical procedures to decide whether or not a given grammar or rewrite system is nonambiguous (such decision procedures exist for context-free textual languages!).

3 Generality of CSRS

In the following, we argue that CSRS can be viewed as a generalization of

- textual context-free grammars
- Picture Layout Grammars
- Logic Grammars

Claim 1: CSRS are at least as expressive as textual context-free grammars.

Proof Sketch: Any context-free grammar can be transformed into an equivalent grammar in Chomsky Normal Form (CNF). Each production of a grammar in CNF has the form $A \rightarrow BC$ or $A \rightarrow t$, where A, B, C are nonterminal symbols and t is a terminal symbol.

We replace each production of the form $A \rightarrow BC$ by a rewrite rule $B(b), C(c) \rightarrow A(a)$ **if** interval(a, b, c) and each production of the form $A \rightarrow t$ by a rewrite rule $t(a) \rightarrow A(a)$. We also introduce the predicate definition interval(iv(a,c),iv(a,b),iv($b+1$,c)).

A string $t_1 t_2 \cdots t_n$ in the textual grammar framework corresponds to the set

$\{t_1$(iv(1,1)),t_2(iv(2,2)),$\cdots$,t_n(iv(n,n))$\}$

in our framework. □

Claim 2: CSRS are at least as expressive as Picture Layout Grammars.

Proof Sketch: Picture Layout Grammars use productions of the form

$$A \rightarrow op(B_1, \cdots, B_m, \underline{B_{m+1}}, \cdots, \underline{B_n})$$
$$A.attr = func(B_1.attr, \cdots, B_n.attr)$$

where

$$pred(B_1.attr, \cdots, B_n.attr)$$
$$\vdots$$

A is a nonterminal symbol, the B_i are terminal or nonterminal symbols. op is a *production operator*, which expresses a spatial relationship between the B_i (such as over, left_of, ...). Underlined symbols, called *remote symbols*, are not actually part of the production, but must occur somewhere else in the parse tree. Attached to each symbol are attributes. *func* computes a new attribute for A from the attributes of the B_i. Attached to each production is a list of predicates that have to be met in order for the production to be applicable.

Each PLG production can be replaced by a rewrite rule

$$B_1(b_1), \cdots, B_n(b_n) \rightarrow A$$
$$(func(b_1, \cdots, b_n)), B_{m+1}(b_{m+1}), \cdots, B_n(b_n) \textbf{ if}$$
$$pred(b_1, \cdots, b_n)$$
$$\vdots$$

□

Picture Layout Grammars are very flexible for describing the syntax of 2D visual languages. The production operators, however, are “hardwired” into the formalism, whereas CSRS allow the definition of arbitrary predicates for describing spatial relationships. Moreover, Picture Layout Grammars are currently restricted to 2D languages, whereas CSRS also allow the definition of 3D languages (see examples 4 – 6).

Claim 3: CSRS are at least as expressive as “Logic Grammars”.

Proof Sketch: A “Logic Grammar” rule has the form

$$P(\overline{s}) \Rightarrow R_1 \wedge \cdots \wedge R_m \ [] \ P_1(\overline{s_1}) \ \& \cdots \& \ P_n(\overline{s_n})$$

$P(\overline{s})$ is a complex picture, the $P_i(\overline{s_i})$ are its less complex components. $R_1, \cdots, R_m$ are additional constraints relating the pictures.

Each Logic Grammar rule can be replaced by a rewrite rule

$$P_1(\overline{s_1}) \ \& \cdots \& \ P_n(\overline{s_n}) \rightarrow P(\overline{s}) \textbf{ if } R_1 \wedge \cdots \wedge R_m$$

□

We see that CSRS and Logic Grammars are very similar. There is, however, one key difference: Logic Grammars can have only one symbol on the left-hand side of a production. This means that they lack the “remote-symbol” feature of Picture Layout Grammars. It would be quite hard to use Logic Grammars to describe directed graph

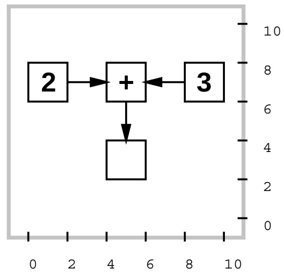

Figure 3: Dataflow diagram

structures (such as data flow languages or state charts). CSRS, on the other hand, allow for an arbitrary number of terms on either side of the rewrite rule.

4 More Examples

Example 3: Consider a 2D visual language based on directed data flow diagrams. A picture belonging to this language consists of a set of boxes, either empty or filled with a number or an operator, and a set of arcs connecting the boxes. Any number of arcs may leave a box. Each box must be

1. filled with a number and having no incoming arc, or
2. empty and having one incoming arc, or
3. containing an operator and having two incoming arcs

For simplicity, let us assume that + is the only legal operator. Fig. 3 shows a picture belonging to this language.

First, let's assume we simply want to decide whether or not a given picture belongs to the language. This is accomplished by the following CSRS:

Rewrite Rules:

$\mathsf{box}(b),\ \mathsf{num}(n,p) \longrightarrow \mathsf{fbox}(b)$ **if** $\mathsf{inside}(p,b)$

$\mathsf{box}(b_1),\ \mathsf{plusop}(p),\ \mathsf{fbox}(b_2),\ \mathsf{fbox}(b_3),\ \mathsf{arc}(p_1,p_2),\ \mathsf{arc}(p_1',p_2') \longrightarrow$
$\mathsf{fbox}(b_1),\ \mathsf{fbox}(b_2),\ \mathsf{fbox}(b_3)$
if $\mathsf{inside}(p,b_1) \wedge \mathsf{attached}(p_1,b_2) \wedge \mathsf{attached}(p_2,b_1)$
$\wedge\ \mathsf{attached}(p_1',b_3) \wedge \mathsf{attached}(p_2',b_1)$

$\mathsf{box}(b_1),\ \mathsf{fbox}(b_2),\ \mathsf{arc}(p_1,p_2) \longrightarrow \mathsf{fbox}(b_1),\ \mathsf{fbox}(b_2)$
if $\mathsf{attached}(p_1,b_2) \wedge \mathsf{attached}(p_2,b_1)$

$\mathsf{fbox}(b) \longrightarrow$

Predicate Definitions:

$\mathsf{inside}(\mathsf{pt}(x,y),\mathsf{bb}(x_1,y_1,x_2,y_2)) \Leftarrow$
$x_1 \leq x \wedge x \leq x_2 \wedge y_1 \leq y \wedge y \leq y_2$

$\mathsf{attached}(p,b) \Leftarrow \mathsf{inside}(p,b)$

An initial picture is represented by a set of terms using the constructor symbols box, num, op, and arc. The rewrite rules 1, 2, and 3 handle the three different kinds of boxes. They remove the num, op, and arc terms from the set, and replace the box by an fbox (a "finished box"). Rule 4 then removes all fboxes. Under this CSRS, a picture is legal if it can be rewritten to the empty set. For instance, the picture shown in Fig. 3 is legal, so it can be rewritten as shown in Table 1.

Now, let's assume we want to *translate* pictures from this visual language into a textual one. Defining a translation function from a visual to a textual language is extremely useful, as it allows us to use the existing methods for describing the semantics of a language. We can formally define a visual language by describing it through a CSRS, thereby defining its syntax and providing a translation to a textual language, and then by giving type rules and an operational or denotational semantics for the textual language.

In this particular example, the textual target language has an assignment statement set(v,e) (which we abbreviate to $v := e$), a block statement block(s_1,s_2) (which we abbreviate to $s_1; s_2$), and a null statement noop. An expression e can be a (target language) variable v, an integer constant, or an addition add(e_1,e_2) (or $e_1 + e_2$ for short). id(t) (where t is a CSRS term) denotes a variable in the target language.

We assume that the set describing the initial picture contains one extra term, target(noop), which is then used to build up the target expression.

The following CSRS performs the desired translation:

Rewrite Rules:

$\mathsf{box}(b) \longrightarrow \mathsf{nbox}(\mathsf{id}(\mathsf{box}(b)),b)$

$\mathsf{nbox}(v,b),\ \mathsf{num}(n,p),\ \mathsf{target}(e) \longrightarrow \mathsf{fbox}(v,b),\ \mathsf{target}(e\ ;\ v{:=}n)$
if $\mathsf{inside}(p,b)$

$\mathsf{nbox}(v_1,b_1),\ \mathsf{plusop}(p),\ \mathsf{fbox}(v_2,b_2),\ \mathsf{fbox}(v_3,b_3),\ \mathsf{arc}(p_1,p_2),$
$\mathsf{arc}(p_1',p_2'),\ \mathsf{target}(e) \longrightarrow$
$\mathsf{fbox}(v_1,b_1),\ \mathsf{fbox}(v_2,b_2),\ \mathsf{fbox}(v_3,b_3),\ \mathsf{target}(e\ ;\ v_1{:=}v_2{+}v_3)$
if $\mathsf{inside}(p,b_1) \wedge \mathsf{attached}(p_1,b_2) \wedge \mathsf{attached}(p_2,b_1)$
$\wedge\ \mathsf{attached}(p_1',b_3) \wedge \mathsf{attached}(p_2',b_1)$

$\mathsf{nbox}(v_1,b_1),\ \mathsf{fbox}(v_2,b_2),\ \mathsf{arc}(p_1,p_2),\ \mathsf{target}(e) \longrightarrow$
$\mathsf{fbox}(v_1,b_1),\ \mathsf{fbox}(v_2,b_2),\ \mathsf{target}(e; v_1 := v_2)$
if $\mathsf{attached}(p_1,b_2) \wedge \mathsf{attached}(p_2,b_1)$

$\mathsf{fbox}(v,b) \longrightarrow$

and attached and inside defined as before

Under this CSRS, a picture is legal if it can be rewritten to the empty set.

Table 2 shows the rewriting of the set representing the picture shown in Fig. 3. This process translates the picture into the textual program s := 2; u := 3; t := s+u; v := t.

CSRS are expressive enough to allow for the specification of three-dimensional visual languages.

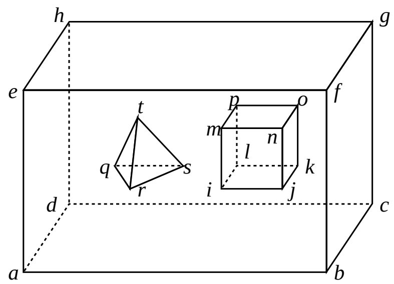

Figure 4: 3D filesystem representation

Example 4: Consider the 3D visual language consisting of all the pictures which contain only tetrahedrons. This language can be described by the following CSRS (we use a Prolog-like notation for lists):

Rewrite Rules:

$\mathsf{pgon}(a),\ \mathsf{pgon}(b),\ \mathsf{pgon}(c),\ \mathsf{pgon}(d) \longrightarrow \mathsf{tetr}(w,x,y,z)$
  if $\mathsf{perm}(a,[w,x,y]) \wedge \mathsf{perm}(b,[w,x,z])$
  $\wedge\ \mathsf{perm}(c,[w,y,z]) \wedge \mathsf{perm}(d,[x,y,z])$

Predicate Definitions:

$\mathsf{perm}([\,],[\,])$

$\mathsf{perm}(l_1,a.l_3) \Leftarrow \mathsf{select}(a,l_1,l_2) \wedge \mathsf{perm}(l_2,l_3)$

$\mathsf{select}(a,a.l,l)$

$\mathsf{select}(x,a.l_1,a.l_2) \Leftarrow \mathsf{select}(x,l_1,l_2)$

A tetrahedron can be described by its 4 corners. Clearly, there are 4 three-element subsets of those corners. Each of the four triangles which make up the tetrahedron can be identified with one of those subsets. In order to decide if two sets are equal, we have to see if the list-representation of the first set unifies with any permutation of the list-representation of the second set. $\mathsf{perm}(l,l')$ is a predicate which holds if the list l' is a permutation of l.

Example 5: Consider the 3D visual language in which all legal pictures contain only axis-aligned parallelopipeds (boxes). This language is specified by the following CSRS:

Rewrite Rules:

$\mathsf{pgon}(a),\ \mathsf{pgon}(b),\ \mathsf{pgon}(c),\ \mathsf{pgon}(d),\ \mathsf{pgon}(e),\ \mathsf{pgon}(f) \longrightarrow$
  $\mathsf{box}(lll,uuu)$
  if $aaa = \mathsf{pt}(x_1,y_1,z_1) \wedge aab = \mathsf{pt}(x_1,y_1,z_2)$
  $\wedge\ aba = \mathsf{pt}(x_1,y_2,z_1) \wedge abb = \mathsf{pt}(x_1,y_2,z_2)$
  $\wedge\ baa = \mathsf{pt}(x_2,y_1,z_1) \wedge bab = \mathsf{pt}(x_2,y_1,z_2)$
  $\wedge\ bba = \mathsf{pt}(x_2,y_2,z_1) \wedge bbb = \mathsf{pt}(x_2,y_2,z_2)$
  $\wedge\ \mathsf{fliprot}(a,[aaa,aab,abb,aba]) \wedge \mathsf{fliprot}(b,[baa,bab,bbb,bba])$
  $\wedge\ \mathsf{fliprot}(c,[aaa,aab,bab,baa]) \wedge \mathsf{fliprot}(d,[aba,abb,bbb,bba])$
  $\wedge\ \mathsf{fliprot}(e,[aaa,aba,bba,baa]) \wedge \mathsf{fliprot}(f,[aab,abb,bbb,bab])$

Predicate Definitions:

$\mathsf{fliprot}(a,b) \Leftarrow \mathsf{fliprot}'(a,[\,],b)$

$\mathsf{fliprot}'(a,b,c) \Leftarrow \mathsf{reverse}(b,b') \wedge \mathsf{append}(a,b',c)$

$\mathsf{fliprot}'(a,b,c) \Leftarrow \mathsf{reverse}(a,a') \wedge \mathsf{append}(a',b,c)$

$\mathsf{fliprot}'(x.a,b,c) \Leftarrow \mathsf{fliprot}'(a,x.b,c)$

$\mathsf{reverse}([\,],[\,])$

$\mathsf{reverse}(a.l,n) \Leftarrow \mathsf{reverse}(l,m) \wedge \mathsf{append}(m,[a],n)$

$\mathsf{append}([\,],l,l)$

$\mathsf{append}(a.l,m,a.n) \Leftarrow \mathsf{append}(l,m,n)$

Tetrahedrons are composed of triangles, so it was sufficient to simply permute the corner vertices of each triangle in order to fit them together. The situation is slightly more complicated for rectangles, where not every permutation describes a valid "rotation" or "flip" of the rectangle (both $\mathsf{pgon}([b,c,d,a])$ and $\mathsf{pgon}([d,c,b,a])$ are valid transformations of $\mathsf{pgon}([a,b,c,d])$, but $\mathsf{pgon}([b,a,c,d])$ isn't!). The predicate $\mathsf{fliprot}(l,l')$ holds if the list of corner vertices l' is a valid transformation of l.

Example 6: Finally, consider a (fictional) 3D visual language for describing Unix file systems. A file is shown as a tetrahedron, a directory is shown as a box, which may contain tetrahedrons and other boxes. A legal picture contains one box (the root directory), which may contain other objects. The following CSRS describes this language:

Rewrite Rules:

$\mathsf{box}(a,b) \longrightarrow \mathsf{dir}(a,b,[\,])$

$\mathsf{tetr}(p_1,p_2,p_3,p_4),\ \mathsf{dir}(a,b,f) \longrightarrow \mathsf{dir}(a,b,\mathsf{tetr}(p_1,p_2,p_3,p_4).f)$
  if $\mathsf{inside}(p_1,a,b) \wedge \mathsf{inside}(p_2,a,b)$
  $\wedge\ \mathsf{inside}(p_3,a,b) \wedge \mathsf{inside}(p_4,a,b)$

$\mathsf{dir}(a,b,f),\ \mathsf{dir}(c,d,f') \longrightarrow \mathsf{dir}(a,b,\mathsf{dir}(c,d,f').f)$
  if $\mathsf{inside}(c,a,b) \wedge \mathsf{inside}(d,a,b)$

Predicate Definitions:

$\mathsf{inside}(\mathsf{pt}(x_1,y_1,z_1),\mathsf{pt}(x_2,y_2,z_2),\mathsf{pt}(x_3,y_3,z_3)) \Leftarrow$
  $\mathsf{between}(x_1,x_2,x_3) \wedge \mathsf{between}(y_1,y_2,y_3) \wedge \mathsf{between}(z_1,z_2,z_3)$

$\mathsf{between}(a,b,c) \Leftarrow (b \leq a \wedge a \leq c) \vee (c \leq a \wedge a \leq b)$

box and tetr are defined in example 4 and 5, respectively

A picture is legal if it can be rewritten to a set of the form $\mathsf{dir}(a,b,f)$. For example, the picture shown in Fig. 4 is legal as it can be rewritten as shown in Table 3.

5 Conclusion

We have introduced the notion of Conditional Set Rewrite Systems, and argued that they provide a very expressive medium for specifying the syntax of two- and threedimensional visual languages, and for translating pictures belonging to a visual language into strings belonging to a textual one.

We have shown CSRS to be more expressive than Picture Layout Grammars and "Logic Grammars", as far as language *specification* is concerned. They are unique in their ability for language *translation*.

Finally, we have given a number of example CSRS to specify various 2D and 3D visual languages, and to perform a translation from a visual to a textual language.

References

[1] Shi-Kuo Chang, Picture Processing Grammar and its Applications, *Information Sciences* **3** (1971), pp. 121 – 148.

[2] Gennaro Costagliola, Masaru Tomita, and Shi-Kuo Chang, A Generalized Parser for 2-D Languages, *1991 IEEE Workshop on Visual Languages*, pp. 98 – 104.

[3] C. Crimi, A. Guercio, G. Nota, G. Pacini, G. Tortora, and M. Tucci, Relation Grammars and their Application to Multidimensional Languages, *Journal of Visual Languages and Computing* **2** (1991), pp. 333 – 346.

[4] Eric J. Golin. A Method for the Specification and Parsing of Visual Languages, Ph. D. thesis, Brown University, 1990.

[5] Eric J. Golin, Parsing Visual Languages with Picture Layout Grammars, *Journal of Visual Languages and Computing* **2** (1991), pp. 371 – 393.

[6] Eric J. Golin and Steven P. Reiss, The specification of visual language syntax, *Journal of Visual Languages and Computing* **1** (1990), pp. 141 – 157.

[7] Nevin Heintze, Joxan Jaffar, Spiro Michaylov, Peter Stuckey, and Roland Yap, The CLP($\mathcal{R}$) Programmer's Manual, Monash University, Australia, 1987.

[8] Richard Helm and Kim Marriott, Declarative Specification of Visual Languages, *1990 IEEE Workshop on Visual Languages*, pp. 98 – 103.

[9] Richard Helm and Kim Marriott, A Declarative Specification and Semantics of Visual Languages, *Journal of Visual Languages and Computing* **2** (1991), pp. 311 – 331.

[10] Stéphane Kaplan, Conditional Rewrite Rules, *Theoretical Computer Science* **33** (1984), pp. 175 – 193.

[11] Kent Wittenburg, Earley-style Parsing for Relational Grammars, *1992 IEEE Workshop on Visual Languages*, pp. 192 – 199.

{ box(bb(0,6,2,8)), num(2,pt(1,7)), box(bb(4,6,6,8)), plusop(pt(5,7)), box(bb(8,6,10,8)), num(3,pt(9,7)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)) } →
{ fbox(bb(0,6,2,8)), box(bb(4,6,6,8)), plusop(pt(5,7)), box(bb(8,6,10,8)), num(3,pt(9,7)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)) } →
{ fbox(bb(0,6,2,8)), box(bb(4,6,6,8)), plusop(pt(5,7)), fbox(bb(8,6,10,8)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)) } →
{ fbox(bb(0,6,2,8)), fbox(bb(4,6,6,8)), fbox(bb(8,6,10,8)), box(bb(4,2,6,4)), arc(pt(5,6),pt(5,4)) } →
{ fbox(bb(0,6,2,8)), fbox(bb(4,6,6,8)), fbox(bb(8,6,10,8)), fbox(bb(4,2,6,4)) } →
{ fbox(bb(4,6,6,8)), fbox(bb(8,6,10,8)), fbox(bb(4,2,6,4)) } →
{ fbox(bb(8,6,10,8)), fbox(bb(4,2,6,4)) } →
{ fbox(bb(4,2,6,4)) } → { }

Table 1: *Parsing* of Fig. 3

{ box(bb(0,6,2,8)), num(2,pt(1,7)), box(bb(4,6,6,8)), plusop(pt(5,7)), box(bb(8,6,10,8)), num(3,pt(9,7)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop) } →
{ nbox(s,bb(0,6,2,8)), num(2,pt(1,7)), box(bb(4,6,6,8)), plusop(pt(5,7)), box(bb(8,6,10,8)), num(3,pt(9,7)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop) } →
{ nbox(s,bb(0,6,2,8)), num(2,pt(1,7)), nbox(t,bb(4,6,6,8)), plusop(pt(5,7)), box(bb(8,6,10,8)), num(3,pt(9,7)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop) } →
{ nbox(s,bb(0,6,2,8)), num(2,pt(1,7)), nbox(t,bb(4,6,6,8)), plusop(pt(5,7)), nbox(u,bb(8,6,10,8)), num(3,pt(9,7)), box(bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop) } →
{ nbox(s,bb(0,6,2,8)), num(2,pt(1,7)), nbox(t,bb(4,6,6,8)), plusop(pt(5,7)), nbox(u,bb(8,6,10,8)), num(3,pt(9,7)), nbox(v,bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop) } →
{ fbox(s,bb(0,6,2,8)), nbox(t,bb(4,6,6,8)), plusop(pt(5,7)), nbox(u,bb(8,6,10,8)), num(3,pt(9,7)), nbox(v,bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop; s := 2) } →
{ fbox(s,bb(0,6,2,8)), nbox(t,bb(4,6,6,8)), plusop(pt(5,7)), fbox(u,bb(8,6,10,8)), nbox(v,bb(4,2,6,4)), arc(pt(2,7),pt(4,7)), arc(pt(8,7),pt(6,7)), arc(pt(5,6),pt(5,4)), target(noop; s := 2; u := 3) } →
{ fbox(s,bb(0,6,2,8)), fbox(t,bb(4,6,6,8)), fbox(u,bb(8,6,10,8)), nbox(v,bb(4,2,6,4)), arc(pt(5,6),pt(5,4)), target(noop; s := 2; u := 3; t := s+u) } →
{ fbox(s,bb(0,6,2,8)), fbox(t,bb(4,6,6,8)), fbox(u,bb(8,6,10,8)), fbox(v,bb(4,2,6,4)), target(noop; s := 2; u := 3; t := s+u; v := t) } →
{ fbox(t,bb(4,6,6,8)), fbox(u,bb(8,6,10,8)), fbox(v,bb(4,2,6,4)), target(noop; s := 2; u := 3; t := s+u; v := t) } →
{ fbox(u,bb(8,6,10,8)), fbox(v,bb(4,2,6,4)), target(noop; s := 2; u := 3; t := s+u; v := t) } →
{ fbox(v,bb(4,2,6,4)), target(noop; s := 2; u := 3; t := s+u; v := t) } →
{ target(noop; s := 2; u := 3; t := s+u; v := t) }

where s = id(bb(0,6,2,8)), t = id(bb(4,6,6,8)), u = id(bb(8,6,10,8)), and v = id(bb(4,2,6,4))

Table 2: *Translation* of Fig. 3

a = pt(0,0,0)	k = pt(8,4,2)	{ pgon([a,b,c,d]), pgon([e,f,g,h]), pgon([a,e,h,d]), pgon([b,f,g,c]), pgon([a,b,f,e]), pgon([d,c,g,h]),
b = pt(10,0,0)	l = pt(6,4,2)	pgon([i,j,k,l]), pgon([m,n,o,p]), pgon([i,m,p,l]), pgon([j,n,o,k]), pgon([i,j,n,m]), pgon([l,k,o,p]),
c = pt(10,6,0)	m = pt(6,2,4)	pgon([q,s,t]), pgon([s,r,t]), pgon([q,r,t]), pgon([q,s,r]) } →
d = pt(0,6,0)	n = pt(8,2,4)	{ box(a,g), pgon([i,j,k,l]), pgon([m,n,o,p]), pgon([i,m,p,l]), pgon([j,n,o,k]), pgon([i,j,n,m]),
e = pt(0,0,6)	o = pt(8,4,4)	pgon([l,k,o,p]), pgon([q,s,t]), pgon([s,r,t]), pgon([q,r,t]), pgon([q,s,r]) } →
f = pt(10,0,6)	p = pt(6,4,4)	{ box(a,g), box(i,o), pgon([q,s,t]), pgon([s,r,t]), pgon([q,r,t]), pgon([q,s,r]) } →
g = pt(10,6,6)	q = pt(2,4,2)	{ box(a,g), box(i,o), tetr(q,s,r,t) } →
h = pt(0,6,6)	r = pt(3,2,2)	{ dir(a,g,[ ]), box(i,o), tetr(q,s,r,t) } →
i = pt(6,2,2)	s = pt(4,4,2)	{ dir(a,g,[ ]), dir(i,o,[ ]), tetr(q,s,r,t) } →
j = pt(8,2,2)	t = pt(3,3,4)	{ dir(a,g,[tetr(q,s,r,t)]), dir(i,o,[ ]) } →
		{ dir(a,g,[dir(i,o,[ ]),tetr(q,s,r,t)]) }

Table 3: Rewrite of Fig. 4