Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 148 (2006) 75–88
www.elsevier.com/locate/entcs

Semantics of Visual Models in a Rule-based Setting[1]

Hans-Jörg Kreowski [a,2] Karsten Hölscher[a,3] Peter Knirsch[a,4]

[a] *Universität Bremen, Fachbereich 3*
Postfach 33 04 40
D-28334 Bremen, Germany

Abstract

In this paper, some fundamental aspects of the semantics of rule-based systems are sketched and related to the semantics of visual models. A rule-based system comprises a set of rules and some control conditions including descriptions of initial and terminal configurations. Semantically, the rules specify a binary relation on configurations of some kind by means of rule applications which are restricted according to the control conditions. As visual models are usually represented by diagrams, graphs or similar configurations, the rule-based setting can be employed to provide visual models with semantics.

Keywords: visual languages, operational semantics, graph transformation

1 Introduction

Traditional programs and specifications are represented as textual expressions and strings with some grammatical structure. Their semantics is often operationally defined by term rewriting or state transition or denotationally given as a mapping into a semantic domain that reflects the syntactic structure.

[1] Research partially supported by the EC Research Training Network SegraVis (Syntactic and Semantic Integration of Visual Modeling Techniques) and by the German Research Foundation (DFG) as part of the Collaborative Research Centre 637 *Autonomous Cooperating Logistic Processes — A Paradigm Shift and its Limitations.*

[2] Email: kreo@informatik.uni-bremen.de
[3] Email: hoelscher@informatik.uni-bremen.de
[4] Email: knirsch@informatik.uni-bremen.de

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.12.013

In recent years, visual models have become very popular in systems development in addition to textual descriptions as the wide-spread use of UML and Petri net proves. Visual models are syntactically represented by diagrams, graphs, or similar configurations of some kind. Although they are intentionally more intuitive and suggestive than textual descriptions, their meaning must be fixed to avoid misunderstandings and mistakes. Like in the case of text-based modelling languages, there are the two possibilities of operational and denotational semantics even in the case of visual models in principle. In contrast to the textual case, the semantics of visual models is not yet worked out systematically. But one encounters a number of tentative proposals in the literature. Many of them point in the direction that the transformation of graphs, diagrams, or other kinds of configurations may play a similar central role as term rewriting in the traditional case of textual models. In this paper, we sketch the semantic potentials of a rule-based setting as it is provided by the area of graph transformation for example.

The paper is organized in the following way. The next section presents the general idea of rules and their application to configurations. Section three introduces an approach to regulate the inherently nondeterministic rule application process using control conditions. Additionally, in the next section, valid initial and terminal configurations are specified allowing for further regulations. The framework given so far is then used to introduce different kinds of semantics (i.e. interpreter semantics, compiler semantics, and denotational semantics), as well as model transformation, and language generation and recognition. The paper ends with some short concluding remarks. All main ideas and definitions are demonstrated using a running example of well-structured flow diagrams.

2 Rules and their application

The objective of rules is their application to some kind of configurations like strings, terms, graphs, diagrams, pictures, etc. The application of a rule to a configuration derives a configuration. By means of all of its applications, a rule as a syntactic item yields a binary relation on configurations as a basic semantic entity.

The elementary ingredients of a rule-based setting are a set $\mathcal{K}$ of *configurations*, a set $\mathcal{R}$ of *rules*, and a *rule application operator* $\Longrightarrow$ that assigns a binary relation $\underset{r}{\Longrightarrow} \subseteq \mathcal{K} \times \mathcal{K}$ to each rule $r \in \mathcal{R}$.

A rule application $con \underset{r}{\Longrightarrow} con'$ is called direct derivation, derivation step,

computation step, transition step, evaluation step, or something like this depending on the framework the rules are used in.

As the semantic relations of rules are binary relations on configurations, one gets the union, the sequential composition, and other set-theoretic operations of semantic relations for free. In particular, one gets the union $\underset{P}{\Longrightarrow} = \bigcup_{r \in P} \underset{r}{\Longrightarrow}$ for a set $P \in \mathcal{R}$ of rules and the reflexive and transitive closure $\overset{*}{\underset{P}{\Longrightarrow}}$ of $\underset{P}{\Longrightarrow}$.

This allows one to consider a set $P \subseteq \mathcal{R}$ of rules as the most elementary version of a rule-based system with two variants of semantics.

(i) **Rule application graph:** $Graph(P) = (\mathcal{K}, \underset{P}{\Longrightarrow})$ with the configurations as nodes and the rule applications as edges.

(ii) **Iterated rule application relation:** $\overset{*}{\underset{P}{\Longrightarrow}} \subseteq \mathcal{K} \times \mathcal{K}$.

$Graph(P)$ is a proper operational semantics while $\overset{*}{\underset{P}{\Longrightarrow}}$ only indicates what is reachable from what configuration and abstracts from the intermediate configurations. In grammatical frameworks, $Graph(P)$ is known as the derivation graph and $\overset{*}{\underset{P}{\Longrightarrow}}$ as the derivation relation (with respect to graph transformation, see, e.g., Rozenberg [27]). In the area of Petri nets, $Graph(P)$ is the reachability graph and $\overset{*}{\underset{P}{\Longrightarrow}}$ the usual firing relation.

Example

As a running example, we discuss well-structured flow diagrams (see, e.g., Farrow, Kennedy, and Zucconi [12]) which are well-known visual models and quite typical predecessors of more modern diagrams like the UML diagrams.

A well-structured flow diagram has a unique entry (○) and a unique exit (□) and is composed of basic statements and decisions indicated by boxes and diamonds resp. (cf. Fig. 1). A diamond is inscribed with a Boolean expression. A basic statement may have an empty box or a box inscribed with an assignment statement. The empty box represents a nonterminal placeholder for another well-structured flow diagram. The first three rules in Fig. 1 show that such a placeholder may be replaced by an assignment, a compound statement, or by a *while*-loop. The control flow in the well-structured flow diagram is given by the direction of the edges. While the exit is never the source of further control flow, each other node is followed by a unique basic statement or decision. The two edges leaving a decision diamond are labelled with T (for true) and F (for false), resp., indicating that the direction of the control flow

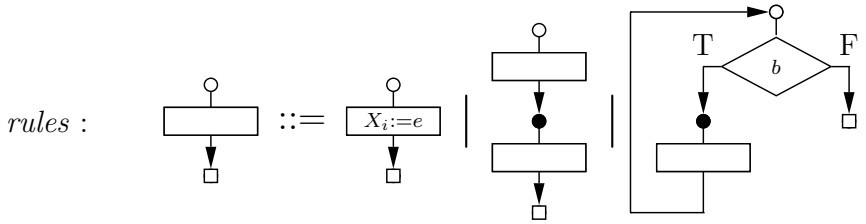


Fig. 1. Rules to refine well-structured flow diagrams

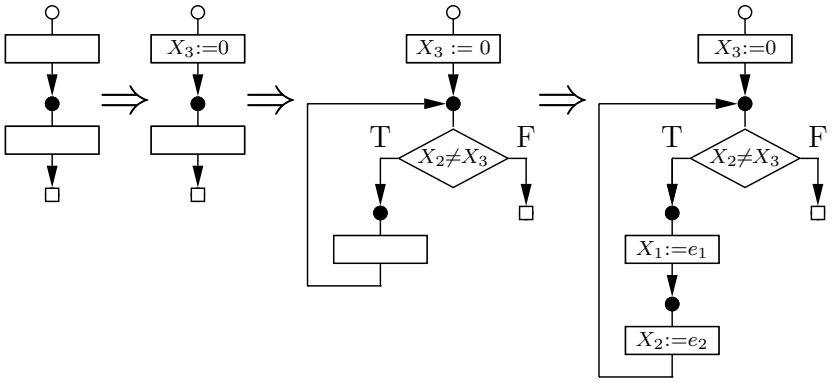


Fig. 2. An example of a derivation sequence

depends on the evaluation of the Boolean expression. Moreover, we assume the natural numbers $\mathbb{N}$ as the only data type and k variables $X_1, \ldots, X_k$ for some $k \in \mathbb{N}$. Then the textual rules in Fig. 1 say that the available arithmetic expressions (to be assigned to a variable) are 0, successor or predecessor of a variable and that the inequality predicate of two variables is the only Boolean expression.

Altogether, the rules describe all possible refinements of well-structured flow diagrams. Fig. 2 shows a sample derivation which refines a compound statement with two nonterminal boxes into an assignment followed by a *while*-loop; the derivation of the textual inscription is omitted.

The example comprises all the ingredients of the computational framework of *while*-programming (see, e.g., Kfoury, Moll, and Arbib [16]).

3 Regulated rule application

But a set of rules is rarely enough to describe what one wants. In grammatical frameworks, derivations start in some initial configuration and end in terminal configurations. In the area of Petri nets, the firing of transitions begins with an initial marking. In the area of term rewriting, only ground terms or reduced

terms are accepted as results of evaluations. Moreover, sometimes one likes to regulate the process of rule application. Examples are the parallel mode of rewriting in L systems and evaluation strategies for terms like *leftmost-innermost* or *parallel-outermost*.

Formally, we may assume a set $\mathcal{C}$ of *control conditions* where a control condition $C \in \mathcal{C}$ provides semantically a binary relation $SEM(C) \subseteq \mathcal{K} \times \mathcal{K}$.

This allows one to consider a pair (P, C) consisting of a set $P \subseteq \mathcal{R}$ of rules and a control condition $C \in \mathcal{C}$ as a *rule-based system* with the intersection of $\overset{*}{\underset{P}{\Longrightarrow}}$ and $SEM(C)$ as relational semantics:

$$SEM(P, C) = \overset{*}{\underset{P}{\Longrightarrow}} \cap\ SEM(C).$$

4 Input-output relations

A typical example of a control condition is a pair (I, T) where I specifies a set $\mathcal{K}(I) \subseteq \mathcal{K}$ of *initial configurations* and T a set $\mathcal{K}(T) \subseteq \mathcal{K}$ of *terminal configurations*. Then the system $(P, (I, T))$ models an input-output relation by

$$SEM(P, (I, T)) = \overset{*}{\underset{P}{\Longrightarrow}} \cap\ (\mathcal{K}(I) \times (\mathcal{K}(T)).$$

This notion covers many computational models like Turing machines, term rewrite systems, graph transformation systems, etc.

In this case, the rule application graph can also be extended as an alternative semantics by $Graph(P, (I, T)) = (\mathcal{K}, \overset{*}{\underset{P}{\Longrightarrow}}, \mathcal{K}(I), \mathcal{K}(T))$ where the nodes corresponding to initial and terminal configurations are distinguished accordingly.

Example

In Fig. 3, we supplement the rules in Fig. 1 by an initial flow diagram consisting of a single empty box and accept everything as terminal except the empty box and the nonterminal letters e and b. This rule-based system specifies the language of *while*-programs in form of well-structured flow diagrams as all terminal configurations that are derivable from the only initial diagram by the given rules.

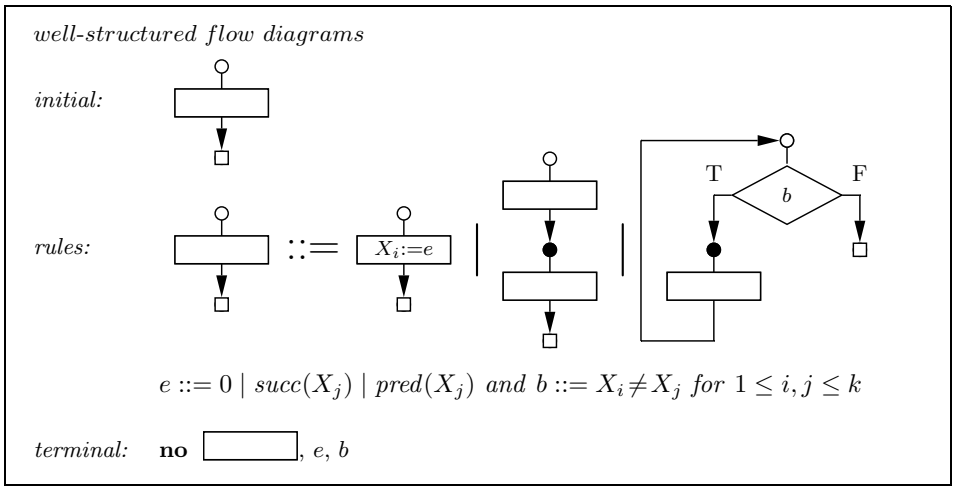


Fig. 3. A rule-based system specifying all well-structured flow diagrams

5 Interpreter semantics

Rules and control conditions together with the rule application graphs or the iterated rule application relations provide the framework for interpreter semantics. Typically, an abstract interpreter is specified in some rule-based language. If the abstract syntax of a language is represented by textual expressions or terms, it is usual to use the same representation for the states of the interpreter and term rewrite rules to model the operations of the interpreter. Analogously, if the abstract syntax of a model is represented by diagrams or graphs, it is meaningful to represent the states of the interpreter by graphs and to model its operations by graph transformation rules (see, e.g., [2,22,29]).

In the latter case, one can distinguish two main approaches. Graph transformation rules may be used as the graphical counterparts of classical approaches to semantics like rewriting logic [25] or the chemical abstract machine [4] while graphical deduction rules follow the structural operational semantics paradigm. The former are often simpler to write because each rule represents a complete interpreter step (see, e.g., [13,15]), while the latter allow a more modular view of the behavior (see, e.g., [7,10,14]).

Example

The semantics of a *while*-program with the variables $X_1, \ldots, X_k$ can be modelled by state transformation. As the variables are global and all of the type of natural numbers, a state is a vector $(n_1, \ldots, n_k) \in \mathbb{N}^k$. The initial state can be chosen arbitrarily. The evaluation starts at the entry and follows the unique

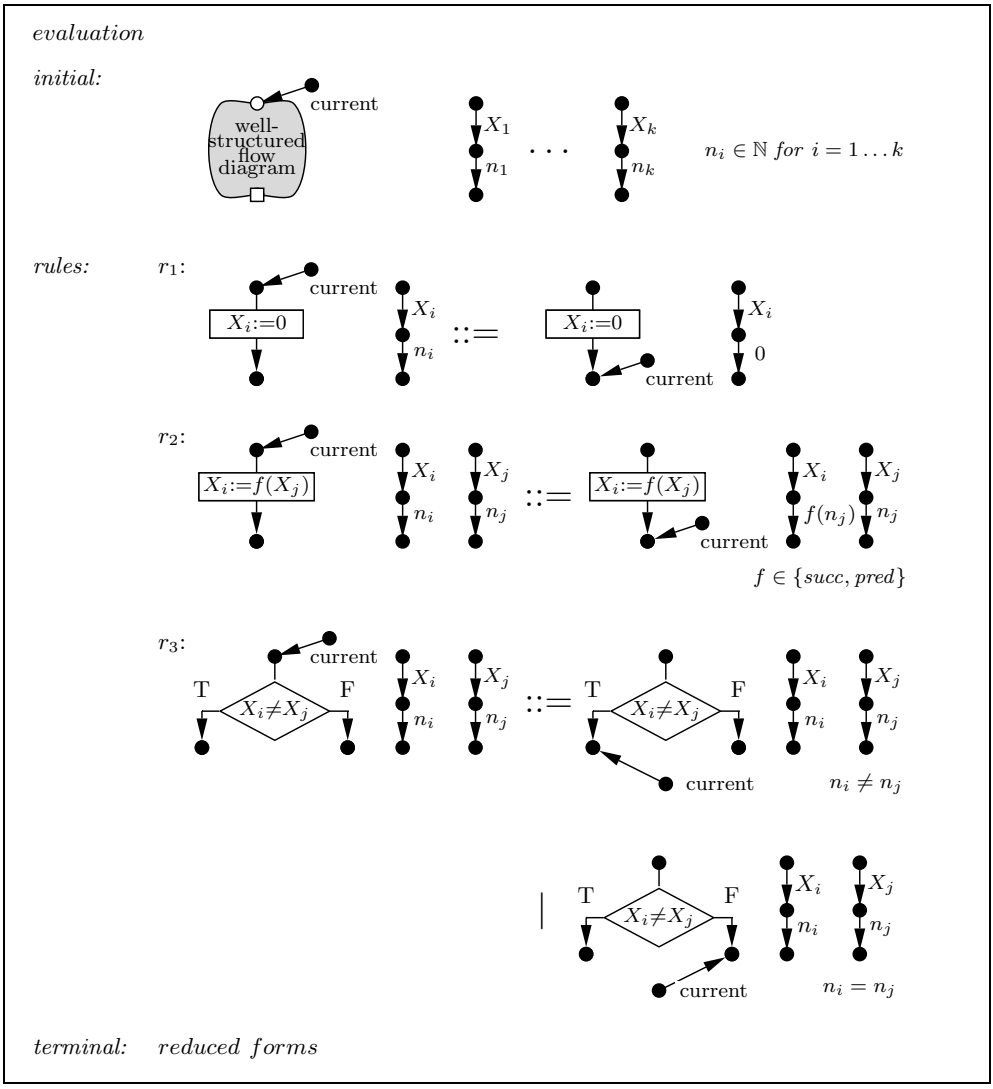


Fig. 4. Rule-based evaluation of well-structured flow diagrams

control flow where the current node is indicated by a respective pointer in each case. If the current node is followed by the assignment $X_i{:=}e$, the expression is evaluated according to the actual state and the resulting value is assigned to X_i yielding the new current state. If the current node is followed by a decision with the Boolean expression $X_i{\neq}X_j$, the actual state is not changed. But the new current node depends on the evaluation of $X_i{\neq}X_j$ according to the actual state. It is the target of the T-edge if $n_i{\neq}n_j$ and the target of the F-edge otherwise. The rules in Fig. 4 describe these evaluation steps. The initial configurations are *well-structured flow diagrams* with a current pointer

at the entry together with a simple graphical representation of a state. The evaluation terminates if no further rule is applicable meaning that the reached configuration is in *reduced form*. This is the case if and only if the current pointer has reached the exit. In this way, Fig. 4 provides an example of an interpreter semantics for visual models.

6 Model transformation

If $\mathcal{K}(I)$ and $\mathcal{K}(T)$ in Section 3 are sets of (syntactic representations of) models of some kind, $SEM(P,(I,T))$ is a rule-based model transformation semantics. An example of this type is the translation of sequence diagrams into collaboration diagrams in [6].

Model transformation resembles the situation of denotational semantics if $\mathcal{K}(I)$ is a set of syntactic items while $\mathcal{K}(T)$ is a set of representations of semantic entities. In this case, the rule-based transformation assigns a meaning to each syntactic item according to its structure. Examples of this kind are the translations of sequence and collaboration diagrams into meta model object diagrams in [6] (cf. [5]).

Various proposals to use graph transformation as a framework for model transformation point in this direction (see, e.g., [1,23,30] and also [26,28] for earlier approaches).

Example

Besides the visual representation as well-structured flow diagrams, *while*-programs can be represented textually, too, in the usual style of imperative programs. Fig. 5 presents a rule-based transformation of the visual models into the textual ones. The initial configurations are well-structured flow diagrams with a translate pointer at the entry. The rules transform the diagram step by step into a string graph that consists of a simple path from the entry to the exit the edges of which are labelled with pieces of program texts. The textual *while*-program is obtained by reading the labels along the path. The translation follows the directions of the edges. An assignment statement is easily translated because only the assignment must be kept as edge label and the translate pointer placed to the target. The other two rules deal with the case that the translate pointer faces a decision diamond. In this case, the edges of the *while*-statement as well as the following statement must be translated and put together in the proper order. The translation terminates in reduced forms (i.e. no rule is applicable anymore) which is only the case if the translate pointer has reached the exit.

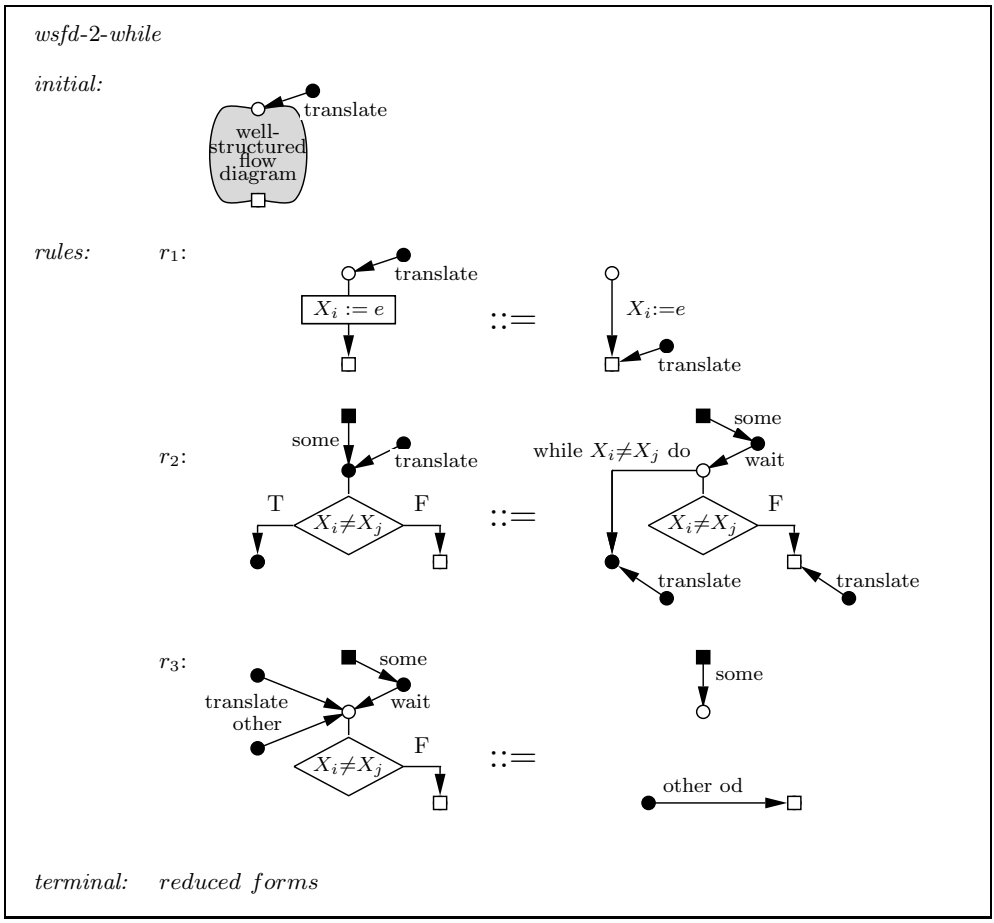


Fig. 5. Transforming visual models into textual models

7 Compiler semantics and denotational semantics

The idea of compiler semantics is to translate each program, specification, or model resp. of the source language into an executable program, specification, or model of a target language. The executability of the target language may be described by means of an interpreter semantics. In the case of a visual modeling language, the executable entity can be given by a set of graph transformation rules working on the graph-based representations of the states of the model (see, e.g., [17,21,24]).

Semantics defined as a mapping from the abstract syntax into some semantic domain is referred to as denotational. According to this definition, the compiler semantics is denotational as well, with a semantic domain that is itself operational. If possible, the mapping of programs, specifications, or

models resp. into the semantic domain should be defined separately for each element of the abstract syntax so that the meaning of the complete entity can be assembled from the meaning of its elements. This compositionality principle is typical for denotational semantics of programming languages, and it is the basis for modular verification, analysis, and evolution of models (see, e.g., [8]).

Assuming that the abstract syntax of visual models is represented by graphs, one faces the problem of describing a mapping from graphs to graphs (if the semantic domain happens to have a diagrammatic syntax, like with Petri nets) or from graphs to text (if the semantic domain is algebraic or logic-based, like a process calculus). For both variants, different forms of graph transformation rules can be found in the literature (see, e.g., [3,9,11]).

Example

The model transformation in Fig. 5 may be seen as a simple example of a compiler that translates visual source programs into textual target programs.

Moreover, *while*-programs have a proper denotational semantics, too. In Fig. 4, the function on the state space computed by a *while*-program is specified by a rule-based interpreter that evaluates a *while*-program for some input state by traversing the diagram in the proper way. In contrast to this, the denotational semantics reflects the syntactic structure of well-structured flow diagrams as given by the rules in Fig. 3. If the mapping that assigns the computed function on states to each well-structured flow diagram is denoted by $[\![\ ]\!]$, then $[\![\ ]\!]$ can be defined by some kind of graphical rules as given in Fig. 6.

8 Language generation and recognition

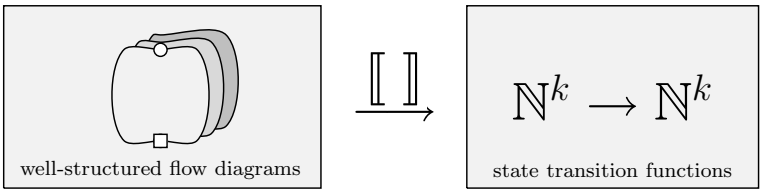
A single configuration (with $\mathcal{K}(S) = \{S\}$) provides an interesting special case of a specification of initial configurations. The relational semantics gets the form

$$SEM(P,(S,T)) = \{S\} \times \{con \in \mathcal{K}(T) \mid S \overset{*}{\underset{P}{\Longrightarrow}} con\}.$$

As the first component of each pair of configurations is S, only the second components are significant. In other words, the system $(P,(S,T))$ can be considered as a grammar or language-generation device with

$$L(P,(S,T)) = \{con \in \mathcal{K}(T) \mid S \overset{*}{\underset{P}{\Longrightarrow}} con\}.$$

Indeed, most notions of grammars that one encounters in the literature



rules :

(i) $[\![X_i{:=}0]\!](n_1, \ldots, n_k) = (n_1, \ldots, n_{i-1}, 0, n_{i+1}, \ldots, n_k)$

(ii) $[\![X_i{:=}f(X_j)]\!](n_1, \ldots, n_k) = (n_1, \ldots, n_{i-1}, f(n_j), n_{i+1}, \ldots, n_k)$

(iii) $[\![\ ]\!](n_1, \ldots, n_k) = [\![\ ]\!]([\![\ ]\!](n_1, \ldots, n_k))$

(iv) $[\![\text{T} \ X_i{\neq}X_j \ \text{F}]\!](n_1, \ldots, n_k) =$

if $n_i \neq n_j$ then $[\![\text{T} \ X_i{\neq}X_j \ \text{F}]\!](n_1, \ldots, n_k)([\![\ ]\!](n_1, \ldots, n_k))$

else $(n_1, \ldots, n_k)$

Fig. 6. Denotational semantics of well-structured flow diagrams

are covered by this case of a rule-based system. Elementary net systems and place/transition systems are further examples of this kind where all cases resp. markings are considered as terminal. A particular example is given in Fig. 3 specifying the language of well-structured flow diagrams. In [8], the set of all UML sequence diagrams is generated as a language by means of graph transformation rules from an initial graph.

The recognition of languages can be modeled similarly. Finite state automata and pushdown automata are examples of this kind.

9 Concluding Remark

In this paper, some basic aspects of the semantics of rule-based systems and of visual modeling have been discussed. It has been sketched how interpreters and compilers as well as operational and denotational semantics of visual models can be seen as special cases of model transformation and hence defined as rule-based systems. This may be considered as one of the first steps into the systematic investigation of the semantics of visual modelling. Further topics like correctness, modularity, compositionality, and others should be included in the future studies. The structuring concepts of transformation unit (see, e.g.,[18,20]) may be helpful in this respect.

References

[1] David H. Akehurst and Stuart J. H. Kent. A relational approach to defining transformations in a metamodel. In Jean-Marc Jézéquel, Heinrich Hussmann, Stephen Cook, editors. *Proc. 5th Conference on UML* 2002 - The Unified Modeling Language. *Lecture Notes in Computer Science*, volume 2460, pages 243-258, Springer, 2002.

[2] Roswitha Bardohl, Mark Minas, Andy Schürr, and Gabriele Taentzer. Application of graph transformation to visual languages. In Ehrig et al., editors. Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools, pages 105–180. World Scientific 1999.

[3] Luciano Baresi. Formal customization of graphical notations. PhD thesis, Dipartimento di Elettronica e Informazione – Politecnico di Milano, 1997. In Italian.

[4] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, Volume 96, No. 1, pages 217–248, 1992.

[5] Grady Booch, James Rumbaugh, and Ivar Jacobsen. The unified modeling language user guide. Addison-Wesley, 1998.

[6] Björn Cordes, Karsten Hölscher, and Hans-Jörg Kreowski. *UML Interaction Diagrams: Correct translation of sequence diagrams into collaboration diagrams.* In John L. Pfaltz, Manfred Nagl, Boris Böhlen, editors, Applications of Graph Transformation with Industrial Relevance *Proc. 2nd Intl. Workshop AGTIVE'03*, Charlottesville, USA, 2003, *Lecture Notes in Computer Science*, volume 3062, pages 275–291. Springer, 2004.

[7] Andrea Corradini, Reiko Heckel, and Ugo Montanari. Graphical operational semantics. In *Proc. ICALP 2000 Workshop on Graph Transformation and Visual Modelling Techniques*. Carleton Scientific, 2000.

[8] Gregor Engels, Luuk Groenewegen, Reiko Heckel, and Jochen Malte Kuester. Consistency-preserving model evolution through transformations. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors. *Proc. UML 2002*, Dresden, Germany, volume 2460 of *Lecture Notes in Computer Science*. Springer, 2002.

[9] Gregor Engels, Jochen Malte Küster, Luuk Groenewegen, and Reiko Heckel. A methodology for specifying and analyzing consistencies of object-oriented behavioral models. In V. Gruhn, editor, *European Software Engineering Conference 2001, Vienna, Austria*, *Lecture Notes in Computer Science*, volume 1301, pages 327–343. Springer, 2001.

[10] Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In Andy Evans, Stuart Kent, and Bran Selic, editors, *Proc. UML 2000 – The Unified Modeling Language. Advancing the Standard*, volume 1939 of *Lecture Notes in Computer Science*, pages 323–337. Springer, 2000.

[11] Gregor Engels, Reiko Heckel, and Jochen Malte Küster. Rule-based specification of behavioral consistency based on the UML meta-model. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 – The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, volume 2185 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2001.

[12] Rodney Farrow Ken Kennedy, and Linda Zucconi. Graph grammars and global program data flow analysis. In *Proc. 17th Annual IEEE Symposium on Foundations of Computer Science*, pages 42–56, Houston, 1976.

[13] Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Dynamic meta modeling with time: Specifying the semantics of multimedia sequence diagram. *Journal of Software and Systems Modelling*, 3(3):181–193, Springer, 2004.

[14] Reiko Heckel and Albert Zündorf. How to specify a graph transformation approach: A meta model for FUJABA. In H. Ehrig and J. Padberg, editors, *Uniform Approaches to Graphical Process Specifiction Techniques, Satellite Workshop of ETAPS 2001, Genova, Italy*, volume 44.4 of *Electronic Notes in TCS*. Elsevier Science, 2001.

[15] Berthold Hoffmann and Mark Minas. A generic model for diagram syntax and semantics, In *Proc. ICALP 2000 Workshop on Graph Transformation and Visual Modelling Techniques*. pages 443–450, Carleton Scientific, 2000.

[16] A. J. Kfoury, Robert N. Moll and Michael A. Arbib. A programming approach to computability. Springer, New York, 1982.

[17] Hans-Jörg Kreowski. Translations into the graph grammar machine. In Ronald Sleep, Rinus Plameijer, and Marco van Eekelen, editors. *Term rewriting: Theory and practice*. John Wiley, New York, pages 171–183, 1993.

[18] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units with interleaving semantics. *Formal Aspects of Computing*, 11(6):690–723, 1999.

[19] Hans-Jörg Kreowski and Sabine Kuske. Approach-independent structuring concepts for rule-based systems. In Martin Wirsing, Dirk Pattison, Rolf Hennicker, editors. *Proc. 16th Int. Workshop on Algebraic Development Techniques (WADT 2002). Lecture Notes in Computer Science*, vol. 2755, pages 299–311, Springer, 2003.

[20] Hans-Jörg Kreowski, Sabine Kuske, and Andy Schürr. Nested graph transformation units. *International Journal on Software Engineering and Knowledge Engineering*, 7(4):479–502, 1997.

[21] Sabine Kuske. A formal semantics of UML state machines based on structured graph transformation. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 – The Unified Modeling Language. Modeling languages, Concepts, and Tools*, volume 2185 of *Lecture Notes in Computer Science*, pages 241–256. Springer, 2001.

[22] Sabine Kuske, Martin Gogolla, Hans-Jörg Kreowski, and Ralf Kollmann. An integrated semantics for UML class, object and state diagrams based on graph transformation. In Michael Butler, Luigia Petre, and Kaisa Sere, editors, *Proc. Third International Conference on Integrated Formal Methods (IFM 2002)*, volume 2335 of *Lecture Notes in Computer Science*, pages 11–28. Springer, 2002.

[23] Juan de Lara and Hans Vangheluwe. A tool for multi-formalism and meta-modelling. In Ralf-Detlef Kutsche, Herbert Weber, editors. *Proc. 5th Int. Conference on Fundamental Approaches to Software Engineering*, vol. 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2002.

[24] Andrea Maggiolo-Schettini and Adriano Peron. Semantics of full statecharts based on graph rewriting. In Hans-Jürgen Schneider, Hartmut Ehrig, editors. *Proc. Graph Transformation in Computer Science*, volume 776 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 1994.

[25] Jóse Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, volume 96, pages 73–155, 1992.

[26] Terrence W. Pratt. Pair grammars, graph languages and string-to-graph translations. *Journal of Computer and System Sciences*, 5:560–595, 1971.

[27] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, Singapore, 1997.

[28] Andy Schürr. Specification of graph translators with triple graph grammars. In G. Tinnhofer, editor, *Proc. WG'94 20th Int. Worhshop on Graph-Theoretic Concepts in Computer Science*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 1994.

[29] Aliki Tsiolakis and Hartmut Ehrig. Consistency analysis of UML class and sequence diagrams using attributed graph grammars. In Hartmut Ehrig, Gabriele Taentzer, editors, *Proc. of Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems* (2000), Technical Report No. 2000/2, pages 77–86, Technical University of Berlin.

[30] Daniel Varró. A formal semantics of UML statecharts by model transition systems. In Andrea Corradini, Hartmut Ehrig, Hans-Jörg Kreowski, Grzegorz Rozenberg, editors. *Proc. First Int. Conference on Graph Transformation,* volume 2505 of *Lecture Notes in Computer Science*, pages 378–79. Springer, 1994.