Parsing And Semantics of a Statechart Variant by Contextual Layered Graph Transformation

Paolo Bottoni,
University of Rome, Italy
bottoni@dsi.uniroma1.it

Gabriele Taentzer*
University of Paderborn, Germany
gabi@upb.de

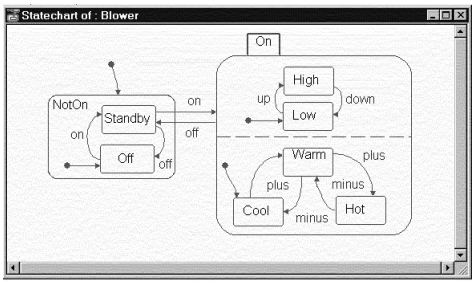

Figure 1. A sample statechart

1 Introduction

We refer here to a variant of Statecharts, as supported by the Rhapsody tool. Figure 1 shows one example of a Rhapsody chart modelling the behavior of a blower with `High` and `Low` ventilation modes and three temperature modes.

In Rhapsody, transitions of the following kinds, breaking the principle of state abstraction, are not permitted: 1) transitions between states of different subdiagrams not nested inside each other (e.g. from `Low` to `Warm` or from `Standby` to `Cool`), 2) transitions between a state and one of its superstates (e.g from `Low` to its superstate `On` or from `Standby` to `NotOn`), and 3) transitions which enter subdiagrams of and-states (e.g. from `NotOn` to `High`).

For this variant we give graph rules for (high-level) parsing and define the operational semantics, basing both on contextual layered graph transformation [2].

2 The Approach

Following a Graph Transformation approach, we describe a sentence in the Statecharts language by a graph where nodes represent entities

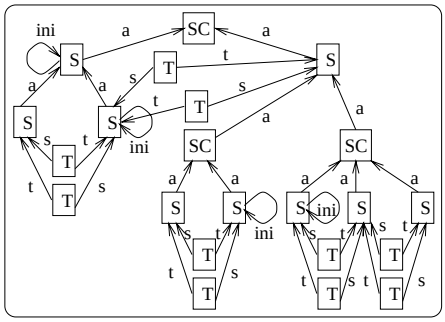

Figure 2. Sample high level graph

(at different levels of abstraction) and edges relations between them. A low-level graph describes the physical level of the sentence, and a high-level one the logical level. Figure 2 is the high level description of the Statechart in Figure 1. Parsing proceeds bottom-up, through repeated application of graph rewrite rules, to a start graph, or to signalling an error.

Graph rules consist of a left and a right-hand side ($LHS \rightarrow RHS$) over typed (labelled) graphs. Parts of both rule sides are related to each others (by numbers). The related parts are preserved during a graph transformation. All non-related graph objects of LHS are deleted, all non-related objects of RHS are created.

A rule may contain a set of *negative application conditions (NACs)* specifying matchings that *must not* exist for it to be applicable. The rule antecedent (LHS+NACs) may contain *complex predicates* on edge types. Thus, an edge with such a predicate as label may be matched to a path in the host graph such that the labels in the path satisfy the predicate in the rule. *Set nodes* may appear in the rule, which are mapped (in a maximal way) to any number of nodes in the host graph, including zero. Consider Figure 3 and 4 for the high-level parsing rules of State-

On the leave of the Technical University of Berlin, Germany, gabi@cs.tu-berlin.de

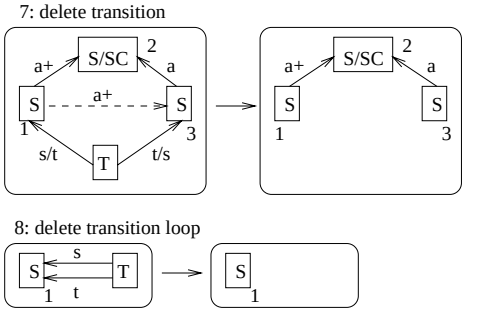

Figure 3. Parsing rules (layer 1)

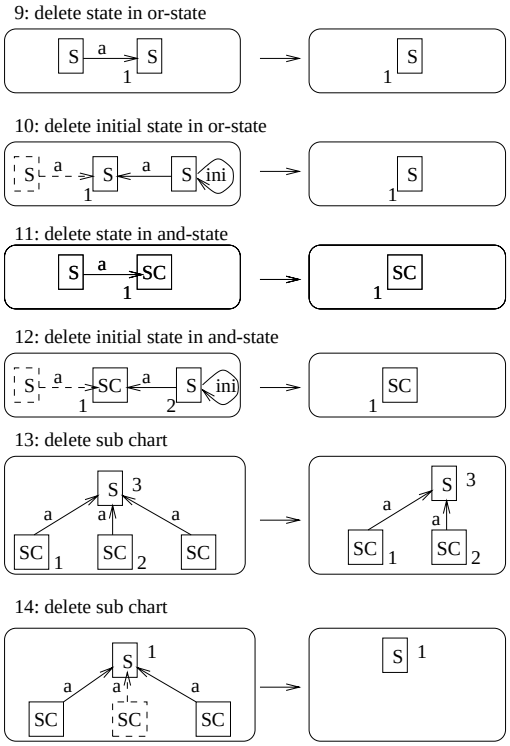

Figure 4. Parsing rules (layer 2)

charts with graph parts in NAC's drawn dashed and complex edge predicates ($a+$) for the deletion of transitions. Rule 1 actually stands for four rules, namely node 2 may be a state (S) or a Statechart (SC) and transition T can go from state 1 to state 3 or the other way round.

Assigning rules as well as node and edge types to layers such that a certain layering condition is satisfied, layer-wise application of rules to a given terminal graph always terminates. Roughly speaking, the layering condition is fulfilled if each rule deletes at least one node or edge from a lower or the same level and creates only graph objects of a higher level. Rules 1 and 2 in Fig. 3 as well as types T, s and t belong to layer

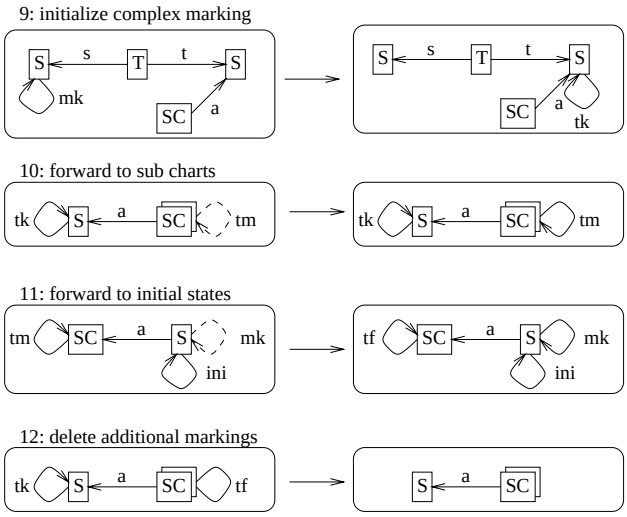

Figure 5. Marking of initial states in subcharts

1 while rules 3 - 8 in Fig. 4 and types S, SC, a, and ini belong to layer 2.

3 Semantics

Graph rewrite rules of the same kind can also be employed to specify the operational semantics of a Statechart, by defining a concurrent state of the system depicted by the Statechart as a collection of marks. Such marks can be modelled as labelled self-edges, i.e. loops, on a state. A transition of the Statechart is modelled as a transformation from one marking to another. If transitions are simple, this reduces to moving the loop from the source to the target state. For complex transitions, such as one to a state composed of several subcharts, all initial states in the subcharts have to become marked. Such a rule can be constructed during the parsing process creating a specialised system of rules for each Statechart. Alternatively, a uniform mechanism can be devised which iterates on all subcharts to mark the initial states. Figure 5 describes such a mechanism, where `mk` stands for an actual mark, `tk` for (temporary) mark, `tm` for moving and `tf` for finished. Due to lack of space we show rules overlooking the possibility of further nesting of the initial states. The complete set of rules requires an additional type of edge to propagate through substates.

4 Conclusions and Tool Support

The proposed approach allows parsing of context-sensitive graph grammars satisfying a layering condition, which reduces sources of inefficiency in existing parsing algorithms. The use of set nodes and complex predicates allows a more compact notation, reducing the size of the grammar, and reducing potential sources of conflicts in the parsing process.

Graph parsing facilities have recently been integrated in the environment AGG [3, 1] for algebraic graph transformation. Most of the graph transformation approach described here (including layers and management of semantics) is supported, apart from set nodes and complex predicates. Set nodes in the presented rules may be replaced by the usage of NACs to check that no node of a given type has been left unaffected by a transformation. Complex predicates can be substituted by preprocessing with another layer containing one rule that computes the transitive closure of the abstraction relation.

AGG embeds management of attributes (storing, updating and checking values). In the application to Statecharts, this allows the association of actions with transitions, as well as the management of internal transition, so that the same tool can be used to construct a Statechart, parse and interpret it, and executing transitions on it.

References

[1] R. Bardohl, T. Schultzke, and G. Taentzer. Visual Language Parsing in GenGEd. In *Proc. of Graph Transformation and Visual Modeling Techniques'01*, volume 55. Electronic Notes in Theoretical Computer Science, 2001. to appear.

[2] P. Bottoni, A. Schürr, and G. Taentzer. Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. Si-2000-06, University of Rome, 2000.

[3] C. Ermel, M. Rudolf, and G. Taentzer. The AGG-Approach: Language and Tool Environment. In H. Ehrig, G. Engels, J.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools*, pages 551–603. World Scientific, 1999.