

This example shows usage of rules with general (also called nested) application conditions and of rule schemes for amalgamated graph transformation.

1 Semantics for statecharts

The grammar is an AGG implementation of the solution for a rule-based semantics of the statecharts which is described in the paper of U. Golas, E.Biermann, H.Ehrig and C.Ermel “A visual Interpreter Semantics for Statecharts Based on Amalgamated Graph Transformation” (see link <http://gcm2010.imag.fr> and Proceedings with all papers of the workshop).

This is a simplified variant of UML statecharts with the most interesting parts of the UML statechart diagrams, where amalgamation is useful for a suitable modeling of the semantical rules. It allows orthogonal regions as well as state nesting. But it does not handle entry and exit actions on states, does not allow extended state variables, and allows guards only to be conditions over active states.

In Fig.1 of the paper an example statechart **ProdCons** is depicted modeling a producer-consumer system. When initialized, the system is in the state **prod**, which has three regions. There, in parallel the producer, a buffer, and the consumer may act. The producer alternates between the states **produced** and **prepare**, where the transition produce between the states **prepare** and **produced** models the actual production activity. It is guarded by a condition that the parallel state **empty** is also current, meaning that the buffer is empty and may receive a produce, which is then modeled by the action **incbuff**. Similarly to the producer, the buffer alternates between the states **empty** and **full**, and the consumer between the states **wait** and **consumed**. The transition consume is again guarded by the state **full** and followed by a **decbuff**-action emptying the buffer.

There are two possible events that may happen causing a state transition leaving the state prod. First, the consumer may decide to leave and finish the complete run. Second, there may be a failure detected after the production leading to the **error**-state. After the machine repaired, the **error**-state can be exited via the corresponding **exit**-transition and the standard behavior in the **prod**-state is executed again, where all three regions are set back to there initial behavior.

Note, that for the states used as conditions in guards we assume to have unique names, but this is merely a problem of the concrete syntax. In the abstract syntax graph, this problem is solved by introducing a direct edge from the guard to this state, and not only a reference by name as done in the concrete statechart diagram.

Please keep open the grammar in AGG to see graphs and rules which will be described bellow.

2 Graph Grammar

2.1 Type Graph

The syntax for statecharts is based on typed attributed graphs and constraints.

The type graph is also given in Fig.2 of the paper.

Multiplicities are used to denote some constraints directly in the type graph.

Note that the edge type **sub** is only necessary to compute all substates of a state, which we need for the definition of the semantics. This relation is computed in the beginning using the states- and regions-edges.

Additional graph constraints are defined and explained in the following. They have to be valid for well-defined statechart diagrams.

2.2 Constraints (boolean formulas) and Atomic Graph Constraints

- **c1** (based on atomics iA1 and iB1)
Each diagram consists of exactly one statemachine SM .
- **c2** (based on atomics iA2, iD2, iE2, iF2)
The state mashin can contain one or more regions R . A region contains states S , where state names are unique within one region. A state may again contain one or more regions. The constraint expresses in addition that each region is contained in either exactly one state or the statemachine.
- **c3** (based on atomics iA3, iC3, iD3, iE3, iF3)
States may be initial (attribute value $isInitial = true$) or final (attribute value $isFinal = true$), each region has to contain exactly one initial and at most one final state, and final states cannot contain regions.
- **c4** (based on atomic iA4)
A transition T begins and ends at a state, is triggered by an event E , and may be restricted by a guard G and followed by an action A . A guard has one or more states as conditions. There is a special event with attribute value $name = "exit"$ which is reserved for exiting state after the completion of all its orthogonal regions, which cannot have a guard condition.
- **c5** (based on atomics iA5, B5)
Final states cannot be the beginning of a transition and their name attribute has to be set to $name = "final"$.
- **c6** (based on atomic iA6)
In addition, transitions cannot link states in different orthogonal regions, which means that both regions are directly contained in the same state.

- **c7** (based on atomics iA7, iB7, iC7)
A pointer *P* describes the active states of the statemachine. Note, that newly inserted current states are marked by the **new**-edge, while for established current states the **current**-edge is used (which is assumed to be the standard type and thus not marked in our diagrams). This differentiation is necessary for the semantics, where we need to differ between states that were current before and states that just became current in the last state transition. Trigger elements *TE* describe the events which have to be handled by the statemachine. Note, that this is not necessarily a queue because of orthogonal states, but for simplicity we still call it event queue. There are at least the empty trigger element with attribute value *name* = *null* and no outgoing **next**-edge, and exactly one pointer in each diagram. The pointer and trigger elements are used later for the description of the operational semantics, but they do not belong to the general syntactical description.

2.3 Graph

The example statechart **ProdCons** is depicted in abstract syntax and is the current graph of our grammar. This graph is also shown in Fig.3 of the paper.

Note, that for final states, which do not have a name in the concrete syntax, the attribute is set to *name* = "*final*". Moreover, the nodes *P* and *TE* are added, which have to exist for a valid statechart model, but are not visible in the concrete syntax (see Fig.1 of the paper). For simulating statechart runs, the event queue of the statechart, which consists only of the default element named *null*, can be filled with events to be processed as explained later.

2.4 Rules

- **setSub, trunsSub**
For the initialization step, compute all substates of all states by applying these two rules as long as possible.
- **init**
Then with init interaction scheme the pointer is associated to the statemachine and all initial states of the statemachines regions.
- **enterRegions**
This interaction scheme handles the nesting and sets the current pointer also to the initial states contained in an active state. When applied as long as possible, this means that all substates are handled. Note, that not all initial substates become active, but only these which are contained in a hierarchy of nested initial states. The identical kernel rule of this interaction scheme ensures that this kernel rule is also applied in the lowest hierarchy level changing the **new**- to a **current**-edge. For later use, also double edges are deleted and if the direct superstate is not marked by the pointer a new edge is added to it.

- **transitionStep**

A state transition representing a semantical step, i. e. switching from one state to another, is done by the application of the interaction scheme **transitionStep** followed by the interaction schemes **enterRegions**, **leaveState1**, **leaveState2**, and **leaveRegions**, each of them applied as long as possible. For such a semantical step, the first trigger element (or one of the first if more than one action of different orthogonal substates may occur next) is chosen and deleted, while the corresponding state transitions are executed. exit-trigger elements are handled with priority which is ensured by the application condition of the kernel rule. Note, that a transition triggered by this trigger element is active if the state it begins at is active, its guard condition state is active, and it has no active substate where a transition triggered by the same event is active. These restrictions are handled by the application conditions of multi rules. Moreover, if an action is provoked, this has to be added as one of the first next trigger elements. The two multi rules of **transitionStep** handle the state transition with and without action, respectively.

- **leaveState1, leaveState2, leaveRegions**

These interaction schemes handle the correct selection of the active states. When for a yet active state with regions, by state transitions all states in one of its regions are no longer active, also this superstate is no longer active, which is described by **leaveState1**. The interaction scheme **leaveState2** handles the case that, when a state become inactive by a state transition, also all its substates become inactive. If for a state with orthogonal regions the final state in each region is reached then these final states become inactive, and if the superstate has an exit-transition it is added as the next trigger element. This is handled by **leaveRegions**.

2.5 Transformation by Rule Sequence

Combining the rules which were explained above leads to the semantics of statecharts.

The operational semantics of statecharts is defined as follows:

- **Initialization step**

For a statechart model we obtain an initial model by applying the sequence
 $\text{setSub!}, \text{transSub!}, \text{init}, \text{enterRegions!}$.

It corresponds to the first subsequence of the rule sequence in AGG:
 $(\text{setSub}^* \text{transSub}^* \text{init} \text{enterRegions}^*)$

- **Semantical step**

A semantical step is computed by applying the sequence
 $\text{transitionStep}, \text{enterRegions!}, \text{leaveState1!}, \text{leaveState2!}, \text{leaveRegions!}$.

It corresponds to the second subsequence of the rule sequence in AGG:
 $(\text{transitionStep} \text{enterRegions}^* \text{leaveState1}^* \text{leaveState2}^* \text{leaveRegions}^*)$