UML state machine version Fortgeschrittenenpraktikum im Wintersemester 2006/2007

Harald Fecher, Heiko Schmidt, and Meiko Jensen

Lehrstuhl Softwaretechnologie
{hf,hsc,mje}@informatik.uni-kiel.de

1 General

This programming exercise deals with the implementation of a graphical modeling language. It includes visual representation and semantic derivation as well as verification support like model checking and refinement checks. The tool shall be developed in groups (number of groups depends on number of participants). The setting describing the model, semantics and tool support, which has to be implemented, is presented in Section 5. Section 6 deals with the separate programming modules. The evaluation method for this programming exercise can be found in Section 4, and the documentation requirements can be found in Section 3. In Section 2, the schedule for the exercise is given and some advice on some topics regarding this course is given in Section 7.

2 Schedule

Number	Date	Name	Meeting required
1.	20.10.	General organization	X
2.	31.10.	Arrangement of groups and assignment of tasks within the groups	Mail
3.	10.11.	Presentation of specification draft	X
4.	15.11.	Final specification and working plan	Mail
5.	17.11.	Presentation of final specification and working plan	X
6.	13.12.	Progress report including documentation	Mail
7.	15.12.	Presentation of progress report	X
8.	12.01.	Terminated feature report	Mail
9.	31.01.	Code deadline, final report including documentation	Mail
10.	02.02.	Final presentation	X

Participation in required meetings is mandatory. If common sense is agreed, different dates can be arranged, but only during the first 10 days. Thereafter the deadlines are strict.

2.1 Organization

We encourage the participants to contact the organizers in case of questions of any kind, or ideas concerning the project (e.g., about additional features).

- Every Thursday 10-11 Meiko is available for questions, especially for technical questions (Java, programming environment, etc.) at room 1107.
- For other questions every group may request a meeting (though not the whole group has to attend it) by mail with Harald or Heiko. For questions concerning the understanding of the different tasks Harald should be preferred, whereas for other questions Heiko should be favored.

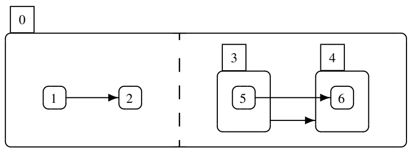
3 Documentation requirement

Writing full length project reports is not part of this exercise. However, three of the deadlines include a short written report (2-3 pages each) about the group's progress.

- Final Specification and working plan should include:
  - detailed responsibility assignment (preferably including comments on group organization)
  - schedule (e.g. using Gantt Chart) including prognosis of work that should be finished until progress report
  - detailed interface description
- Progress report should include:
  - revised schedule, showing previous as well as remaining work
  - progress description of individual modules incl. outlook on remaining work
  - revised interface description (if changed since working plan)
  - unexpected problems that might have occurred and how they have been solved
- Terminated feature report should include:
  - a list of of the contained features for every group (the feature list may not be modified afterwards)
  - a short description of every feature
- Final report should include:
  - program status
  - final interface description (if changed since progress report)
  - description of unfinished tasks and known unsolved bugs
  - unexpected problems that might have occurred and how they have been solved

4 Evaluation

The evaluation is made at the end of the course depending on the delivered code, documentation, and presentations. The final code is most important. We also expect interaction between the participants (including organizers) in order to maximize the success of the resulting software project. **Deadlines are strict** (once they have been agreed upon). If deadlines are missed, it is possible that not all SWS-points can be given. Missing SWS-points for a Schein can however be obtained after the semester by providing extra work.



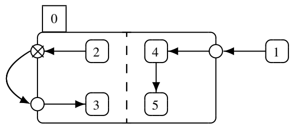


Fig. 1. Two core state machines.

5 Setting

In this section the theoretical foundations are presented.

5.1 Modeling language

First an informal Before the system is formally presented, we refer to a illustration example in Figure 1.

The model we call *core state machine* consists of

- a finite set Var of variables, which have a finite domain in the natural numbers, which in turn may vary between the variables
- a finite set of events $\mathcal{E}$
- different graphical components: states (final or non-final), which can be nested, regions, pseudostates (entry, exit (three different kinds), choice). States are rectangular boxes. Regions are depicted as dashed lines dividing a state. Choice pseudostates are depicted as diamonds. Entry pseudostates are circles on a state border. Exit pseudostates are circles on the state border, where a p, a n, or c is written inside. Different states and choice pseudostates may not overlap.
- a finite number of transitions (arrows) between (pseudo)states. For further constraints how the source and target of the transitions are restricted we refer to [1]. A transition has an element from $\mathcal{E}$ (or none), a guard g and an activity α. Here

$$\alpha ::= \mathsf{skip} \mid x := \mathrm{t} \mid x := \mathsf{random}(N) \mid \mathsf{send}(e, y)$$

$$\mathrm{t} ::= y \mid \mathrm{t} + \mathrm{t} \mid -\mathrm{t} \mid \mathrm{t} * \mathrm{t}$$

$$\mathrm{g} ::= y \geq z \mid y = z \mid \mathrm{nab} \mid \mathrm{wla} \mid \mathrm{g} \wedge \mathrm{g} \mid \neg \mathrm{g} \mid \mathrm{g} \vee \mathrm{g} \mid$$

where $x \in \mathrm{Var}$, $y, z \in \mathrm{Var} \cup \mathbb{N}$ is variable, N is a finite subset of natural numbers, and e an event. The meaning of these terms are as follows: skip is an activity without any effect; $x := \mathrm{t}$ is a variable assignement; $x := \mathsf{random}(N)$ is a variable assignement where a random variable from N is assigned to x; $\mathsf{send}(e, y)$ is a sending of event e combined with the number y, resp. the number assigned in y, to the environment. Terms and guards are interpreted as usual, exept that guard wla indicates that the target state of the transition having wla as guard was last active and nab indicates that the region of the target state of the transition having nab as guard was not active before (can only be understood after working through the whole semantics; nevertheless for the most paricipants its meaning is not important).

Furthermore, states may also have internal transitions, where the source and target are the same.

See also [1] for explanations. There also deferal of events and do-actions are considered. These are not relevant for this project. Every student should have a look at the first 4 section of [1], but do not have to understand everything.

5.2 Semantic Model

For a binary relation $R \subseteq M_1 \times M_2$ abd $\ddot{M}_1 \subseteq M_1$, we define $\ddot{M}_1.R = \{m_2 \in M_2 \mid (m_1, m_2) \in R\}$. For a trinary relation $\rightsquigarrow \subseteq M_1 \times \mathcal{A}ct \times M_2$ we write $m_1 \overset{a}{\rightsquigarrow} m_2$ for $(m_1, a, m_2) \in \rightsquigarrow$, and write $\overset{a}{\rightsquigarrow}$ for the binary relation $\{(m_1, m_2) \mid m_1 \overset{a}{\rightsquigarrow} m_2\}$, thus $\{m_1\}. \overset{a}{\rightsquigarrow} = \{m_2 \in M_2 \mid m_1 \overset{a}{\rightsquigarrow} m_2\}$. Furthermore, relation $\rightsquigarrow$ is deterministic, if $\forall m \in M_1, a \in \mathcal{A}ct : |\{m\}. \overset{a}{\rightsquigarrow} | \leq 1$.

Our semantical model are *ν-automata*, which are explained in the following:

Definition 1 (ν-automaton). *A* ν-automaton *with respect to $\mathcal{A}ct$ is a tuple $(S, S^i, \longrightarrow)$, where S is its set of states, $S^i \subseteq S$ is its set of root states, and $\longrightarrow \subseteq S \times \mathcal{A}ct \times \mathcal{P}(S)$ is its hypertransition relation. (we restrict to those where for every state and action an outgoing transition exists).*

A ν-automaton is concrete *if it has exactly one root state, i.e., $|S^i| = 1$, and exactly one transition per label and per source state.*

An example of a ν-automaton is depicted in Figure 2.

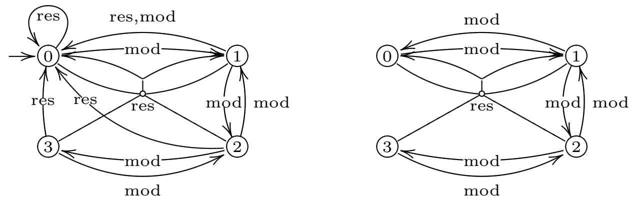


Fig. 2. Two ν-automata over the possible transition labels from $\{\text{mod}, \text{res}\}$. A hypertransition is illustrated by an arrow with a single starting point, a single label and several targets. A set of hypertransitions sharing the same targets and label are combined using a small circle.

Definition 2 (ν-refinement). *The ν-automaton $(S_1, S_1^i, \longrightarrow_1)$ ν-refines the ν-automaton $(S_2, S_2^i, \longrightarrow_2)$ if there is a ν-refinement R, i.e., a relation $R \subseteq S_1 \times S_2$ such that*

- *for every more concrete root state there is an related abstract root state, i.e., $\forall s_1^i \in S_1^i : \exists s_2^i \in S_2^i : (s_1^i, s_2^i) \in R$, and*

– *for any pair of related states* $(s_1, s_2) \in R$ *and any event* $e \in \mathcal{A}ct$ *any more concrete transition can be matched by an abstract one such that the states of the two target sets can be pairwise related, i.e.,* $\forall \Theta_1 \in \{s_1\}. \xrightarrow{a}_1 : \exists \Theta_2 \in \{s_2\}. \xrightarrow{a}_2 : (\forall s_1' \in \Theta_1 : \exists s_2' \in \Theta_2 : s_1' R s_2') \wedge (\forall s_2' \in \Theta_2 : \exists s_1' \in \Theta_1 : s_1' R s_2')$.

For example, the ν-automaton on the right hand side of Figure 2 is a ν-refinement of the ν-automaton on the left hand side of the same Figure.

Semantics The semantics of a state machine is declared via a transformation into a ν-automaton. This transformation is given in [1], where some minor changes have to be made in order to handle the random-operator (will be explained by discussion with Harald). Note that we do not expect that this transformation will be understood by reading the paper, i.e., intensive discussion with Harald should take place. An example of the semantic is illustrated in Figure 3. The ν-automaton on the left of Figure 2 is the semantics for the state machine on the left hand side of Figure 3, with label "res [true] $x := 1$" changed to "res [true] x:=$rand(0, 1)$". The ν-automaton on the right of Figure 2 is the semantics of the state machine, where furthermore the transition labelled "res [true] x:=0" is deleted.

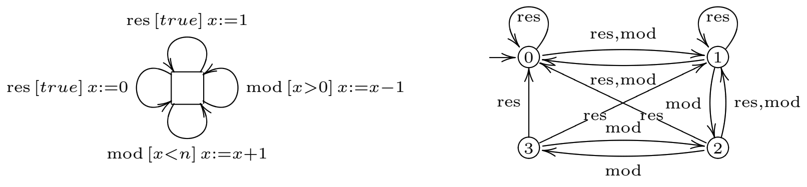


Fig. 3. A simple state machine having variable x. The state machine, which is depicted at the left hand side, reacts on event *res* in which case the variable is reset to a not yet fixed value 0 or 1. Furthermore, the state machine reacts also on event *mod* in which case x is incremented or decremented by 1, which is only possible if the value of x remains in $\{0, ..., n\}$. The initial value of x is 0. Its semantics for $n = 3$ in terms of transition system is depicted on the right hand side the state machine. The value of n is written inside each transition system state.

5.3 Property setting

The property language is the modal mu-calculus:

$$\phi ::= X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a \rangle \phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi$$

The dual formula is obtained by replacing $\wedge$ with $\vee$, $\langle a \rangle$ with $[a]$, μ with ν, and vice versa.

In order to define the satisfaction relation mu-calculus formulas are translated into tree automaton [2]:

Definition 3 (Tree automata). *An* alternating tree automaton *is a tuple (Q, δ, Θ), where*

- *$(q \in)Q$ is a finite, nonempty set of states*
- *δ is a transition relation, which maps an automaton state to one of the following forms, where q, q_1, q_2 are automaton states: $q \mid q_1 \tilde{\wedge} q_2 \mid q_1 \tilde{\vee} q_2 \mid \langle \alpha \rangle q \mid [\alpha] q$ and*
- *$\Theta \colon Q \to \mathbb{N}$ an acceptance condition with finite image.*

μ-calculus formulas are transformed into tree automata as follows, where we assume that every variable X is under the scope of $\mu.X$ or $\nu.X$, and no variable is bound twice: formulas different from $\mu X.\phi$, $\nu X.\phi$, X are straightforwardly translated. $\mu X.\phi$ (respectively $\nu X.\phi$, X) becomes automaton state X with $\delta(X)$ is the automaton state corresponding to ϕ. Function Θ is everywhere equal to 0 except on X states, here the value is determined by the alternating depth (counting in a path starting in $\mu X.\phi$ how often the binders switch from μ to ν, respectively from ν to μ, where one is added if the last binder is a μ operator. For examples: If $\nu Z.\mu Y.((\mu X.X) \vee (\nu V.V)$, then $\Theta(X) = 1$, $\Theta(V) = 0$, $\Theta(Y) = 1$, and $\Theta(Z) = 2$.

The satisfaction relation is defined as follows:

Definition 4 (Satisfaction).

- *Finite satisfaction plays for ν-automaton $\mathcal{D} = (S, S^i, \longrightarrow)$ and alternating tree automaton A have the rules and winning conditions as stated in Table 1. An infinite play Φ is a* win *for Player I iff* $\sup(\mathrm{map}(\Theta, \Phi[2]))]$ *is even; otherwise it is won by Player II.*
- *The model $\mathcal{D}$* satisfies *the automaton A in state $(s, q) \in S \times Q$, written as $(\mathcal{D}, s) \models (A, q)$ iff Player I has a strategy for the corresponding satisfaction game between $\mathcal{D}$ and A such that Player I wins all satisfaction plays started at (s, q) with her strategy.*

q':	the next configuration is (s, q')
$q_1 \tilde{\wedge} q_2$:	Player II picks a q' from $\{q_1, q_2\}$; the next configuration is (s, q')
$q_1 \tilde{\vee} q_2$:	Player I picks a q' from $\{q_1, q_2\}$; the next configuration is (s, q')
$\langle a \rangle q'$:	Player II picks $\ddot{S}' \in \{s\}. \xrightarrow{a}$; Player I picks $s' \in \ddot{S}'$; the next configuration is (s', q')
$[a] q'$:	Player II picks $\ddot{S}' \in \{s\}. \xrightarrow{a}$ and $s' \in \ddot{S}'$; the next configuration is (s', q')

Table 1. Moves of satisfaction game at configuration (s, q). Satisfaction plays are sequences of configurations generated thus

6 Modules

Here the different modules together with the SWS-points we have in mind are presented. The groups can assign tasks between the members such that participants can obtain enough SWS-points to get a Schein over either 4 or 8 SWS at the end. Note that the SWS-points are only for orientation and depend in particular on the task promised in the working plan. Then we decide how much SWS the students get for the planned tasks.

6.1 Coordinator

A person responsible for delivering the common parts like the working plan. He/she has to write the common documentations and give the common presentation. Here common means the non single module dependent issues.

Value: 2-4 SWS

6.2 Data structure implementation

The common data structure has to be implemented with efficient algorithms, hence good underlying structures have to be used and in particular presented in the documentation. Further, a file representation of this structure (including save-to-file and load-from-file operations) must be specified and implemented.

Value: 4-6 SWS

6.3 Input of core state machines

Implementation of graphical input facilities (GUI) for core state machines. Modifications should be possible at any time. Furthermore, concurrent execution and thread safe programming should take place.

Value: 8-16 SWS

Optional:

Implementation of graphical input facilities for ν-automata.

Value: +2 SWS

6.4 Semantics

A transformation from state machines into ν-automata has to be implemented reflecting the semantics as described in Subsection 5.2.

Value: 8 SWS

Optional:

The following interactive simulation should be implemented: A user may give an action, where the system executes a corresponding allowed transition. If there is more than one possibility a random one should be chosen. If no such allowed transition exists, the user is informed of the absence. A user may save the current simulation point. A user may switch back to a saved simulation point.

A user may terminate its examination, in which case his/her simulation may be displayed via a labeled tree.

The steps can also be highlighted those activities inside the corresponding core state machine

Value: 2-6 SWS

6.5 Testing

Data test instances have to be provided, enabling testing for the different groups. The implemented code has to be additionally tested by the Tester. Furthermore, a benchmark has to be provided and runtime examination has to be made.

Value: 2-4 SWS

6.6 Model checking

This task is alrady assigned (for one group).

Optional:

Input facilities for the modal μ-calculus, a transformation from the mu-calculus into tree automata, and a satisfaction check between a μ-calculus and a tree automata has to be implemented. The satisfaction check should yield true (if the ν-automaton satisfies the formula), false (if the ν-automaton satisfies the dual formula), or undefined (otherwise) as output.

Value: 8 SWS

6.7 Refinement check

Implementation of an algorithm for refinement check via greatest fixpoint calculation (ask Harald for that technique).

Value: 2 SWS

7 Advice

7.1 About groupwork

To learn something about successful groupwork regarding software projects, the following essay is recommended, not only for this project but for all future group assignments:

`http://www.csc.calpoly.edu/∼sludi/SEmanual/TableOfContents.html`

7.2 Tool recommendations

The implementation has to be made in *JAVA SE 5.0* We recommend to use a programming environment like
eclipse (`http://www.eclipse.org/`).
We expect reasonable comments in your code explaining non-trivial parts of code. We recommend to make use of
javadoc (`http://java.sun.com/j2se/javadoc/`).
We recommend to use *lint4j* for code optimization. Also *DOT* can be used for graphs illustrations.

7.3 Presentation

Here are a view tips about presenting you probably already know, but experience shows you can never say it often enough:

- be prepared – do not just hope everything will work, make sure it will (as good as you can)
- know your work – learning the speech by heart is not enough, you have to know what you are talking about
- stay focussed – don't get carried away by unnecessary details
- know your audience – do not waste time for telling things everyone already knows, but do not leave important things out the audience probably does not know
- rehearse your presentation before you give it – not only will it save you a lot of embarrassment, without rehearsing it you will not know what time it takes to give it
- rehearse your presentation before you give it – we mean it!

7.4 Subversion

- Access to the Subversion repository
  In order to get an account for the projects repository an account for the RBG-Rechnernetz is required. The corresponding user name should be included when enlisting in the list of participants on October 20th. If you were prevented to do so for any reason, please send a mail with the missing information to `hsc@informatik.uni-kiel.de` as soon as possible. The passwords will be handed out on Monday, October 23th. With your password and username you will be able to visit `https://snert.informatik.uni-kiel.de`. Attention: please accept the ssl certificate although it is issued by an unknown authority.
- There is a good book "Version Control with Subversion" at
  `http://svnbook.red-bean.com/`
- Initial Checkout
  Checking out a repository creates a copy of projects repository on your local machine in directory smile. Attention: please accept the ssl certificate although it is issued by an unknown authority. This working copy contains the latest revision of the repository that you specify on the command line:

  ```
  svn checkout https://snert.informatik.uni-kiel.de/svn/smile
  ```

  After that, entering subversion command line client commands within the working directory does not require the url anymore.
- The typical work cycle using Subversion
  - Update your working copy with `svn update`
  - Change, add, delete, copy and move files with `svn add <file>`, `svn delete <file>`, `svn copy <file>` and `svn move <file>`
  - Examine your changes with `svn status`, solve conflicts and merge others' changes into your working copy `svn update`
  - Commit your changes with short comment `svn commit -m "<your comment>"`

7.5 Code Reuse

In the previous semester there was a similar project. Its task can be downloaded from `http://www.informatik.uni-kiel.de/inf/deRoever/SS06/Praktikum/`. It is allowed to reuse code from that project but you should be careful and tell us. It can be found at `https://snert.informatik.uni-kiel.de/simdim.zip`. However, note that there is a modified task now and you are responsible for possibly existing bugs or missing comments in the reused code.

References

[1] H. Fecher and J. Schönborn. Uml 2.0 state machines: Complete formal semantics via core state machines. In *FMICS*, Lecture Notes in Computer Science. Springer, 2006. Accepted for publication.

[2] Th. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2):359–391, May 2001.