Modelierungs Sprache
Fortgeschrittenenpraktikum im Sommersemester 2006

Jens Schönborn
Harald Fecher
Marc Walker

Lehrstuhl Softwaretechnologie

1 General

This programming exercise deals with the implementation of a simple graphical modeling language. It includes visual representation and semantic derivation as well as verification support like model checking or refinement checks. The tool has to be developed in groups. 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 requirement is 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.	7.4	General organisation	X
2.	17.4	Arrange groups and assignment of tasks within the groups	Mail
3.	21.4	Presentation of specification draft	X
4.	28.4	Final specification and working plan	X
5.	2.6	Progress report including documentation	X
6.	7.7	Code deadline, final report including documentation	X
7.	14.7	Presentation	X

Participation in these meetings is mandatory except for (2.). For (2.) sending a mail with the required information is sufficient.

2.1 Organization

- Every monday form 14:00 to 16:00 Marc is available for questions, especially for technical questions at room 1210.
- For other questions every group may request a meeting (though not the whole group has to attend it), generally not more often than once a week. Please contact Jens in order to make an appointment.

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 organisation)
  - 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
- 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 following deadlines given in section 2 are binding: (2.), (3.), (4.), (5.), (6.) and (7.). 5 points are assigned for every one that is met. If a deadline is not met up to two extensions will be given, each one yields -5 points. For work associated to three of the deadlines points will be assigned. These deadlines and the maximal reachable points are:

- 20 points for final specification and working plan (4.),
- 20 points for progress report including documentation (5.) and
- 60 points for code, final report including documentation and presentation (6. and 7.).

65 of the maximal 130 reachable points have to be achieved in order to fulfill the Scheinkriterium.

5 Setting

5.1 Modeling language

The model we call 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 actions $\mathcal{A}ct$ (can be understood as method names)
– two graphical components (rectangle and circle), which can be used finite times. Intuitionally rectangles can be seen as system states, and circles represent nondeterminism that will be resolved in later design steps.
– a finite number of transitions (arrows) between the components, whereas no transitions may exist between two circles. A transition coming out of a rectangle has an action from $\mathcal{A}ct$, a guard g and an assignment $x := \mathrm{t}$ where $x \in \mathrm{Var}$, whereas a transition from a circle only has a guard and an assignment. Here

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

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

where $y, z \in \mathrm{Var} \cup \mathbb{N}$. These are interpreted as usual.

5.2 Semantic Model

Our semantical model are *disjunctive modal transition systems*, which are an extension of modal transition systems. These models are explained in the following:

Modal transition systems (MTS) [4, 3], which were introduced as a specification formalism, are LTSs (labeled transition systems) with two kinds of transitions: *must* transitions indicating that the implementation (in terms of LTSs) must have such a transition; and *may* transitions indicating that the implementation may have such transitions, i.e., a transition in the implementation is only allowed if there is a corresponding may transition in the MTS. This intuition is generalized leading to a refinement concept between MTSs as well. Note that a MTS where the must transitions coincide with the may transitions is considered as a (concrete) LTS. For example, in Fig. 1 three MTSs are presented, where must transitions are depicted as solid arrows, may transitions are depicted as dashed arrows and all must transitions imply a corresponding may transition, which are omitted in the drawing. The MTSs depicted on the left and on the right side, which can be considered as concrete LTSs, are refinements of the MTS depicted in the middle. An LTS without an outgoing transition labeled with a is not a refinement of the MTS depicted in the middle of Fig. 1. Note that MTSs are expressive enough to model combinations of inherit and unspecified non-determinism. MTSs have the additional condition that every must transition is also a may transition. If this condition is dropped, the obtained systems are called *mixed transition systems* [2].

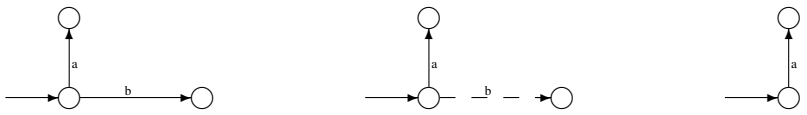

Fig. 1. MTSs illustration

Disjunctive modal transition systems (DMTS) [5], which were originally developed as a model for the characterization of equation-solving within process algebras, generalize MTSs by introducing so-called hypertransitions, which point to sets of states rather

than to single states. The meaning of a hypertransition is that at least one state in its target set must be a target of a transition in the implementation. For example, the LTSs presented in Fig. 2 are refinements of the DMTS presented in Fig. 3, where hypertransitions are depicted by a divided arrow. An LTS without outgoing transitions labeled with a is not an refinement of the DMTS presented in Fig. 1.

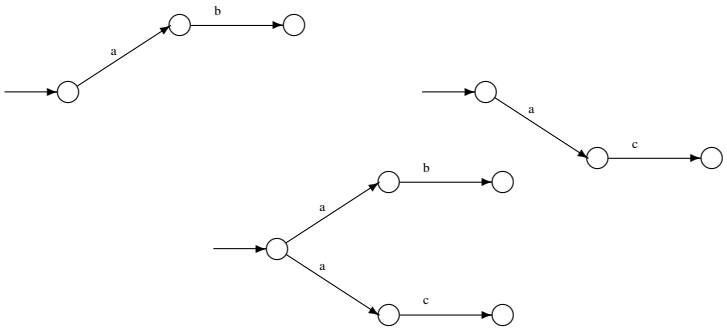

Fig. 2. LTS illustration for DMTS

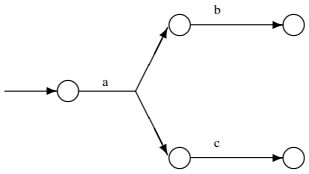

Fig. 3. DMTS illustration

The model is introduced formally in the following, where we first introduce some notations. $\mathcal{A}ct$ denotes the set of all possible actions (i.e., transition labels). $\mathcal{P}(M)$ denotes the power set of M. For a binary relation $R \subseteq M_1 \times M_2$ and $\ddot{M}_1 \subseteq M_1$, we define $\ddot{M}_1.R = \{m_2 \in M_2 \mid (m_1, m_2) \in R\}$. For a relation $\rightsquigarrow \subseteq M_1 \times \mathcal{A}ct \times M_2$ we write $m_1 \overset{a}{\rightsquigarrow} m_2$ for $(m_1, a, m_2) \in \rightsquigarrow$, define $\mathcal{O}_{\rightsquigarrow}(m_1) = \{a \in \mathcal{A}ct \mid \exists m_2 \in M_2 : m_1 \overset{a}{\rightsquigarrow} m_2\}$, and write $\overset{a}{\rightsquigarrow}$ for the binary relation $\{(m_1, m_2) \mid m_1 \overset{a}{\rightsquigarrow} m_2\}$, thus $\{m_1\}.\overset{a}{\rightsquigarrow} = \{m_2 \in M_2 \mid m_1 \overset{a}{\rightsquigarrow} m_2\}$.

Definition 1 (Disjunctive modal transition systems). *A* disjunctive modal transition system *(DMTS) $\mathcal{D}$ with respect to $\mathcal{A}ct$ is a tuple $(S, S^{\mathrm{i}}, \longrightarrow, \longmapsto)$ such that*

- *$(s \in) S$ is a set of states,*
- *$S^{\mathrm{i}} \subseteq S$ is a nonempty set of initial states.*
- *$\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is its may transition relation, and*

– $\longmapsto \subseteq S \times \mathcal{A}ct \times \mathcal{P}(S)$ *its must transition relation, such that* $\forall s \in S, a \in \mathcal{A}ct, \ddot{S} \subseteq S : s \xmapsto{a} \ddot{S} \Rightarrow \ddot{S} \subseteq \mathcal{O}_{\longrightarrow}(s)$

Definition 2. $R \subseteq S_1 \times S_2$ *is a* refinement *between two DMTSs* $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *if*

– $\forall s_1^{\mathrm{i}} \in S_1^{\mathrm{i}} : \exists s_2^{\mathrm{i}} \in S_2^{\mathrm{i}} : (s_1^{\mathrm{i}}, s_2^{\mathrm{i}}) \in R$.
– $\forall a \in \mathcal{A}ct, s_1, s_1', s_2 : ((s_1, s_2) \in R \wedge s_1 \xrightarrow{a}_1 s_1') \Rightarrow \exists s_2' : s_2 \xrightarrow{a}_2 s_2' \wedge (s_1', s_2') \in R$.
– $\forall a \in \mathcal{A}ct, \ddot{S}_2 \in \{s_2\}. \xmapsto{a}: \exists \ddot{S}_1 \in \{s_1\}. \xmapsto{a}: \forall s_1' \in \ddot{S}_1 : \exists s_2' \in \ddot{S}_2 : (s_1', s_2') \in R$ *whenever* $(s_1, s_2) \in R$.

$\mathcal{D}_1$ DMTS-refines $\mathcal{D}_2$ *if there exists a DMTS-refinement* R *between* $\mathcal{D}_1$ *and* $\mathcal{D}_2$.

5.3 Semantics

The semantics of a state machine is declared as a DMTS as follows:

– The states of the DTMS conform to possible variable assignments (f) in combination with the possible rectangles (ℓ), that is, with information about which current state of the state machine is active.
– A must transition from (f, ℓ) to D labeled a exists if there is a state machine transition t from ℓ such that t has action a, the guard of t evaluates to true under f, and D is determined as follows: let f' be obtained by applying the assignment of t on f yielding; if the target of t is a rectangle ℓ' then $D = \{(f', \ell')\}$; if the target is a circle ℓ' then $(f'', \ell'') \in D$ iff there is is a state machine transition t' from ℓ' such that the guard of t evaluates to true under f', f'' is obtained by applying the assignment of t' on f'; and ℓ'' is the target of t. Note that in case D is the empty set, no transition is generated.
– A may transition from (f, ℓ) to (f', ℓ') labeled a exists if there is a must transition from (f, ℓ) to D labeled a such that $(f', \ell') \in D$.

5.4 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 [6]:

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*
– Θ*:* $Q \to \mathbb{N}$ *an acceptance condition with finite image.*

Mu calculus formulas are transformed into tree automaton as follows, where we assume that every variable X is under the scope of μX or $\nu.X$, and no variable is bound twice: formulas different from $\mu X.\phi$, $\mu X.\phi$, X are straightforwardly translated. $\mu X.\phi$ (respectively $\mu X.\phi$, 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 model $\mathcal{D}$ 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 I picks $\ddot{S}' \in \{s\}$. $\overset{a}{\longmapsto}$; Player II picks $s' \in \ddot{S}'$; the next configuration is (s', q')
$[a]q'$: Player II picks $s'' \in \{s\}$. $\overset{a}{\longrightarrow}$; 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

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 documentation. Furthermore, he/she has to implement the main procedure. The main procedure has to allow concurrence, e.g. for proving two poperties on the same DMTS at the same time, and has to ensure mutual exclusion, e.g. a DMTS cannot be modified if it is currently used for property verification.

Value: 4 SWS

6.2 Input and data handling of state machines and DMTS

Implementation of graphical input facilities for state machines. Modifications should be possible at any time. Furthermore, the data structure of state machines and also of DMTS shall be implemented and the specified interface shall be provided.

Value: 16 SWS

Optional:
Implementation of graphical input facilities for DMTS.
Value: +2 SWS

6.3 Semantics

A transformation from state machines into DMTS has to be implemented reflecting the semantics as described in Subsection 5.3.

Value: 4 SWS

Optional:

The following interactive simulation should be implemented: A user may give an action, where the system executes a corresponding may transition. If there is more than one possibility a random one should be chosen. If no such may transition exists, the user is informed of the absense. 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 should be displayed via a labelled tree.

Value: +4 SWS

6.4 Testing

The implemented code has to be tested. In particular, a benchmark has to be provided and tools supporting testing have to be used (have to be found by the tester him/herself). Furthermore, a presentation including a sketched describtion of the tools used has to be given.

Value: 4 SWS

6.5 Model checking

Optional:

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

Value: 8 SWS

6.6 Refinement check

Implementation of an algorithm for refinement check.

Value: 2 SWS

Optional:

Implementation of more efficient algorithms for refinement check. In particular a variant of the one presented in [1]

Value: +2 SWS

7 Advice

7.1 About groupwork

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

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

7.2 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 unneccessary 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!

7.3 Subversion

- Access to the Subversion repostitory
  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 April 7th. If you were prevented to do so for any reason, please send a mail with the missing information to `jes@informatik.uni-kiel.de` as soon as possible. The passwords will be handed out on Monday, April 10th. 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 simdim. 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/simdim
```

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>"`

References

[1] R. Cleaveland and O. Sokolsky. Equivalence and preorder checking for finite-state systems. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 391–424. North-Holland, 2001.

[2] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

[3] K. G. Larsen. Modal specifications. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 232–246. Springer-Verlag, 1990.

[4] K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210. IEEE Computer Society Press, 1988.

[5] K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117. IEEE Computer Society Press, 1990.

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