Softwaretechnik / Software-Engineering

Lecture 14: Behavioural Software Modelling

2019-07-01

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Architecture & Design: Content

- Introduction and Vocabulary
- Software Modelling
  - model, views, viewpoints: 4+1 view
  - Modelling structure
  - (simplified) Class & Object diagrams
  - (simplified) Object Constraint Logic (OCL)
- Modelling behaviour
  - Communicating Finite Automata (CFA)
  - Uppaal query language
  - CFA vs. Software
  - Unified Modelling Language (UML)
    - basic state-machines
    - an outlook on hierarchical state-machines
- Principles of Design
  - modularity, separation of concerns
  - information hiding and data encapsulation
  - abstract data types, object orientation
- Design Patterns
- Model-driven/-based Software Engineering

Vocabulary

Techniques

informal

semi-formal

formal
Communicating Finite Automata (CFA)
- concrete and abstract syntax,
- networks of CFA,
- operational semantics.

Transition Sequences

Deadlock, Reachability

Uppaal
- tool demo (simulator),
- query language,
- CFA model-checking.

CFA at Work
- drive to configuration, scenarios, invariants
- tool demo (verifier).

Uppaal Architecture
Example
To define communicating finite automata, we need the following sets of symbols:

- A set \( (a, b \in \text{Chan}) \) of channel names or channels.
- For each channel \( a \in \text{Chan} \), two visible actions: \( a? \) and \( a! \) denote input and output on the channel \( (a?, a! \notin \text{Chan}) \).
- \( \tau \notin \text{Chan} \) represents an internal action, not visible from outside.
- \( (\alpha, \beta \in \text{Act}) \) is the set of actions.

**An alphabet** \( B \) is a set of channels, i.e. \( B \subseteq \text{Chan} \).

For each alphabet \( B \), we define the corresponding action set
\[
B_{!!} := \{ a? \mid a \in B \} \cup \{ a! \mid a \in B \} \cup \{ \tau \}.
\]

**Note:** \( \text{Chan}_{!!} = \text{Act} \).

### Integer Variables and Expressions, Resets

- Let \((v, w \in \text{V})\) be a set of (finite domain integer) variables.

  By \((\varphi \in \Psi(\text{V}))\) we denote the set of integer expressions over \(V\) using function symbols \(+,-,\ldots\) and relation symbols \(<,\leq,\ldots\)

- A modification on \(v \in \text{V}\) is of the form
\[
v := \varphi, \quad v \in \text{V}; \quad \varphi \in \Psi(\text{V}).
\]

  By \(R(\text{V})\) we denote the set of all modifications.

- By \(\vec{r}\) we denote a finite list \(\langle r_1, \ldots, r_n \rangle\), \(n \in \mathbb{N}_0\), of modifications \(r_i \in R(\text{V})\). \(\vec{r}\) is called reset vector (or update vector).

  \(\langle \rangle\) is the empty list \((n = 0)\).

- By \(R(\text{V})^*\) we denote the set of all such finite lists of modifications.
Definition. A communicating finite automaton is a structure

\[ A = (L, B, V, E, \ell_{\text{ini}}) \]

where
- \((\ell \in L)\) is a finite set of locations (or control states),
- \(B \subseteq \text{Chan}\),
- \(V\): a set of data variables,
- \(E \subseteq L \times B \times \Phi(V) \times R(V)^* \times L\): a finite set of directed edges such that
  \[ (\ell, \alpha, \varphi, \vec{r}, \ell') \in E \land \text{chan}(\alpha) \in U \implies \varphi = \text{true}. \]

Edges \((\ell, \alpha, \varphi, \vec{r}, \ell')\) from location \(\ell\) to \(\ell'\) are labelled with an action \(\alpha\), a guard \(\varphi\), and a list \(\vec{r}\) of modifications.
- \(\ell_{\text{ini}} \in L\) is the initial location.

Example

Abstract syntax: \( A = (L, B, V, E, \ell_{\text{ini}}) \)

\[ A_1 : \]

\[ L = \{ 0, 1, 2 \} \]
\[ B = \{ A \} \]
\[ V = \{ x \} \]
\[ \ell_{\text{ini}} = 0 \]
\[ E = \{ (0, A), x := 0, x := 2, A \} \]

\[ A_2 : \]

\[ m_0 \quad A_7 \quad m_1 \]
Definition. Let $A_i = (L_i, B_i, V_i, E_i, \ell_{ini,i}), 1 \leq i \leq n$, be communicating finite automata. The operational semantics of the network of CFA $C(A_1, \ldots, A_n)$ is the labelled transition system $T(C(A_1, \ldots, A_n)) = (\text{Conf}, \text{Chan} \cup \{\tau\}, \{\lambda \rightarrow \lambda \mid \lambda \in \text{Chan} \cup \{\tau\}\}, C_{ini})$ where

- $V = \bigcup_{i=1}^{n} V_i$,
- $\text{Conf} = \{(\vec{\ell}, \nu) \mid \ell_i \in L_i, \nu : V \rightarrow \mathcal{P}(V)\}$,
- $C_{ini} = (\vec{\ell}_{ini}, \nu_{ini})$ with $\nu_{ini}(v) = 0$ for all $v \in V$.

The transition relation consists of transitions of the following two types.

 Helpers: Extended Valuations and Effect of Resets

- $\nu : V \rightarrow \mathcal{P}(V)$ is a valuation of the variables,
- A valuation $\nu$ of the variables canonically assigns an integer value $\nu(\phi)$ to each integer expression $\phi \in \Phi(V)$.
- $\models \subseteq (V \rightarrow \mathcal{P}(V)) \times \Phi(V)$ is the canonical satisfaction relation between valuations and integer expressions from $\Phi(V)$.
- Effect of modification $r \in \mathcal{R}(V)$ on $\nu$, denoted by $\nu[r]$:
  $$\nu[r] := \begin{cases} \nu(\phi), & \text{if } a = v, \\ \nu(a), & \text{otherwise} \end{cases}$$
- We set $\nu[r_1, \ldots, r_n] := \nu[r_1]\ldots[r_n] = ((\nu[r_1])[r_2])\ldots[r_n]$.

That is, modifications are executed sequentially from left to right.
An internal transition \( \langle \vec{\ell}, \nu \rangle \xrightarrow{\tau} \langle \vec{\ell}', \nu' \rangle \) occurs if there is \( i \in \{1, \ldots, n\} \) and

- there is a \( \tau \)-edge \( (\ell_i, \tau, \varphi, \vec{r}, \ell'_i) \in E_i \) such that
  - \( \nu|_{\vec{r}} = \varphi \), "source valuation satisfies guard"
  - \( \vec{r} = \ell_i \quad \Rightarrow \quad \ell'_i \), "automaton \( i \) changes location"
  - \( \nu' = \nu|_{\vec{r}} \), "\( \nu' \) is the result of applying \( \vec{r} \) on \( \nu \)

A synchronisation transition \( \langle \vec{\ell}, \nu \rangle \xrightarrow{b} \langle \vec{\ell}', \nu' \rangle \) occurs if there are \( i, j \in \{1, \ldots, n\} \) with \( i \neq j \) and

- there are edges \( (\ell_i, b!, \varphi_i, \vec{r}_i, \ell'_i) \in E_i \) and \( (\ell_j, b?, \varphi_j, \vec{r}_j, \ell'_j) \in E_j \) such that
  - \( \nu|_{\vec{r}_i} = \varphi_i \land \nu|_{\vec{r}_j} = \varphi_j \), "source valuation satisfies guards (0)"
  - \( \vec{r}_i = \ell_i \quad \Rightarrow \quad \ell'_i \), "automaton \( i \) and \( j \) change location"
  - \( \nu' = \nu|_{\vec{r}_i}|\vec{r}_j \), "\( \nu' \) is the result of applying first \( \vec{r}_i \) and then \( \vec{r}_j \) on \( \nu \)

This style of communication is known under the names "rendezvous", "synchronous", "blocking" communication (and possibly many others).
Transition Sequences

- A transition sequence of $C(A_1, \ldots, A_n)$ is any (in)finite sequence of the form

$$\langle \vec{l}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{l}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{l}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \ldots$$

with

- $\langle \vec{l}_0, \nu_0 \rangle = C_{\text{ini}}$.
- for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ in $T(C(A_1, \ldots, A_n))$ with $\langle \vec{l}_i, \nu_i \rangle \xrightarrow{\lambda_{i+1}} \langle \vec{l}_{i+1}, \nu_{i+1} \rangle$. 

Example
Reachability

- A configuration \(\langle \vec{\ell}, \nu \rangle\) is called **reachable** (in \(C(A_1, \ldots, A_n)\)) from \(\langle \vec{\ell}_0, \nu_0 \rangle\) if and only if there is a transition sequence of the form
  \[
  \langle \vec{\ell}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{\ell}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{\ell}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \cdots \xrightarrow{\lambda_n} \langle \vec{\ell}_n, \nu_n \rangle = \langle \vec{\ell}, \nu \rangle.
  \]

- A configuration \(\langle \vec{\ell}, \nu \rangle\) is called **reachable** (without “from”!) if and only if it is reachable from \(C_{ini}\).

- A location \(\ell \in L_i\) is called **reachable** if and only if any configuration \(\langle \vec{\ell}, \nu \rangle\) with \(\ell_i = \ell\) is reachable, i.e. there exist \(\vec{\ell}\) and \(\nu\) such that \(\ell_i = \ell\) and \(\langle \vec{\ell}, \nu \rangle\) is reachable.

Deadlock

- A configuration \(\langle \vec{\ell}, \nu \rangle\) of \(C(A_1, \ldots, A_n)\) is called **deadlock** if and only if there are no transitions from \(\langle \vec{\ell}, \nu \rangle\), i.e. if
  \[
  \neg (\exists \lambda \in \Lambda \exists \langle \ell', \nu' \rangle \in Conf \bullet \langle \ell, \nu \rangle \xrightarrow{\lambda} \langle \ell', \nu' \rangle).
  \]
  The network \(C(A_1, \ldots, A_n)\) is said to **have a deadlock** if and only if there is a reachable configuration \(\langle \ell, \nu \rangle\) which is a deadlock.
Uppaal

(Larsen et al., 1997; Behrmann et al., 2004)

Tool Demo
Consider $N = C(A_1, \ldots, A_n)$ over data variables $V$.

- **basic formula:**
  
  $\text{atom} ::= A_i.\ell \mid \varphi \mid \text{deadlock}$

  where $\ell \in L_i$ is a location and $\varphi$ an expression over $V$.

- **configuration formulae:**

  $\text{term} ::= \text{atom} \mid \text{not term} \mid \text{term}_1 \land \text{term}_2$

- **existential path formulae:**

  $e\text{-formula} ::= \exists^0 \text{term}$ (exists finally)
  
  $\mid \exists^\infty \text{term}$ (exists globally)

- **universal path formulae:**

  $a\text{-formula} ::= \forall^0 \text{term}$ (always finally)
  
  $\mid \forall^\infty \text{term}$ (always globally)
  
  $\mid \text{term}_1 \Rightarrow \text{term}_2$ (leads to)

- **formulae (or queries):**

  $F ::= e\text{-formula} \mid a\text{-formula}$

---

**Satisfaction of Uppaal Queries by Configurations**

- The satisfaction relation $\langle \vec{\ell}, \nu \rangle \models F$ between configurations $\langle \vec{\ell}, \nu \rangle = \langle (\ell_1, \ldots, \ell_n), \nu \rangle$ of a network $C(A_1, \ldots, A_n)$ and formulae $F$ of the Uppaal logic is defined inductively as follows:

  - $\langle \vec{\ell}, \nu \rangle \models \text{deadlock}$ if $\langle \vec{\ell}, \nu \rangle$ is a deadlock config.
  
  - $\langle \vec{\ell}, \nu \rangle \models A_i.\ell$ if $\ell_i = \ell$
  
  - $\langle \vec{\ell}, \nu \rangle \models \varphi$ if $\nu \models \varphi$
  
  - $\langle \vec{\ell}, \nu \rangle \models \text{not term}$ if $\nu \not\models \varphi$
  
  - $\langle \vec{\ell}, \nu \rangle \models \text{term}_1 \land \text{term}_2$ if $\nu \models \text{term}_1$ and $\nu \not\models \text{term}_2$
Example: Computation Paths vs. Computation Tree

Example: Computation Paths vs. Computation Graph
(or: Transition Graph)
Satisfaction of Uppaal Queries by Configurations

**Exists finally:**

\[ \langle \vec{e}_0, \nu_0 \rangle \models \exists \diamond \text{term} \quad \text{iff} \quad \exists \text{path } \xi \text{ of } N \text{ starting in } \langle \vec{e}_0, \nu_0 \rangle \quad \exists i \in \mathbb{N}_0 \quad \xi^i \models \text{term} \]

"some configuration satisfying term is reachable"

**Example:** \[ \langle \vec{e}_0, \nu_0 \rangle \models \exists \diamond \varphi \]

\[ \langle \vec{e}_0, \nu_0 \rangle \not\models \varphi \]

\[ \lambda_1 \]

\[ \lambda_2 \]

\[ \lambda_{1,1} \]

\[ \lambda_{2,1} \]

\[ \lambda_{2,2} \]

\[ \lambda_{2,2,1} \]

\[ \lambda_{2,2,2} \]

\[ \langle \vec{e}, \nu \rangle \]

**Exists globally:**

\[ \langle \vec{e}_0, \nu_0 \rangle \models \exists \square \text{term} \quad \text{iff} \quad \exists \text{path } \xi \text{ of } N \text{ starting in } \langle \vec{e}_0, \nu_0 \rangle \quad \forall i \in \mathbb{N}_0 \quad \xi^i \models \text{term} \]

"on some computation path, all configurations satisfy term"

**Example:** \[ \langle \vec{e}_0, \nu_0 \rangle \models \exists \square \varphi \]
• Always globally:

\[ \langle \vec{\ell}_0, \nu_0 \rangle \models \forall \Diamond \text{term} \iff \langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists 0 \neg \text{term} \]

"not (some configuration satisfying \( \neg \text{term} \) is reachable)"

or: "all reachable configurations satisfy \( \text{term} \)"

• Always finally:

\[ \langle \vec{\ell}_0, \nu_0 \rangle \models \forall \Box \text{term} \iff \langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists \neg 0 \text{term} \]

"not (on some computation path, all configurations satisfy \( \neg \text{term} \))"

or: "on all computation paths, there is a configuration satisfying \( \text{term} \)"

**Leads to:**

\[ \langle \vec{\ell}_0, \nu_0 \rangle \models \text{term}_1 \longrightarrow \text{term}_2 \iff \forall \text{path } \xi \text{ of } \mathcal{N} \text{ starting in } \langle \vec{\ell}_0, \nu_0 \rangle \forall i \in \mathbb{N}_0 \bullet \]

\[ \xi^i \models \text{term}_1 \implies \xi^i \models \forall \Box \text{term}_2 \]

"on all paths, from each configuration satisfying \( \text{term}_1 \),

a configuration satisfying \( \text{term}_2 \) is reachable" (response pattern)

**Example:** \( \langle \vec{\ell}_0, \nu_0 \rangle \models \varphi_1 \longrightarrow \varphi_2 \)

![Diagram showing a model of Uppaal queries with configurations and paths]

(End of page 26)
Definition. Let $\mathcal{N} = C(A_1, \ldots, A_n)$ be a network and $F$ a query.

(i) We say $\mathcal{N}$ satisfies $F$, denoted by $\mathcal{N} \models F$, if and only if $C_{ini} \models F$.

(ii) The model-checking problem for $\mathcal{N}$ and $F$ is to decide whether $(\mathcal{N}, F) \in \models$.

Proposition.
The model-checking problem for communicating finite automata is decidable.
Model Architecture — Who Talks What to Whom

- **Shared variables:**
  - bool water_enabled, soft_enabled, tea_enabled;
  - int w = 3, s = 3, t = 3;

- **Note:** Our model does not use scopes ("information hiding") for channels. That is, 'Service' could send 'WATER' if the modeler wanted to.
**Design Sanity Check: Drive to Configuration**

- **Question**: Is it (at all) possible to have no water in the vending machine model? (Otherwise, the design is definitely broken.)

- **Approach**: Check whether a configuration satisfying
  \[ w = 0 \]
  is reachable, i.e. check whether
  \[ \mathcal{N}_{VM} \models \exists w = 0. \]
  for the vending machine model \( \mathcal{N}_{VM} \).

---

**Design Check: Scenarios**

- **Question**: Is the following existential LSC satisfied by the model? (Otherwise, the design is definitely broken.)

  LSC: buy tea
  AC: true
  AM: initial I: permissive

- **Approach**: Use the following newly created CFA 'Scenario'

  ![Diagram of the vending machine model with locations and transitions]

  instead of User and check whether location `end_of_scenario` is reachable, i.e. check whether
  \[ \mathcal{N}'_{VM} \models \exists \text{Scenario}.end\_of\_scenario. \]
  for the modified vending machine model \( \mathcal{N}'_{VM} \).
**Design Verification: Invariants**

- **Question**: Is it the case that the “tea” button is *only* enabled if there is €1.50 in the machine? (Otherwise, the design is broken.)

- **Approach**: Check whether the implication

  \[ \text{tea\_enabled} \implies \text{CoinValidator\_have\_c150} \]

  holds in all reachable configurations, i.e. check whether

  \[ \mathcal{N}_VM \models \forall \square (\text{tea\_enabled} \implies \text{CoinValidator\_have\_c150}) \]

  for the vending machine model \( \mathcal{N}_VM \).

---

**Design Verification: Sanity Check**

- **Question**: Is the “tea” button ever enabled? (Otherwise, the considered invariant

  \[ \text{tea\_enabled} \implies \text{CoinValidator\_have\_c150} \]

  holds vacuously.)

- **Approach**: Check whether a configuration satisfying \( \text{water\_enabled} = 1 \) is reachable. Exactly like we did with \( w = 0 \) earlier (i.e. check whether \( \mathcal{N}_VM \models \exists \diamond \text{water\_enabled} = 1 \)).
• **Question:** Is it the case that, if there is money in the machine and water in stock, that the "water" button is enabled?

• **Approach:** Check

\[ V_{VM} \models \forall \Box (\text{CoinValidator}.\text{have}_c50 \text{ or CoinValidator}.\text{have}_c100 \text{ or CoinValidator}.\text{have}_c150) \implies \text{water}_{\text{enabled}}. \]

---

**Recall: Universal LSC Example**
What Can We Conclude From Verification Results?

- Assume that query \( Q \) corresponds to a requirement on the system under development, and \( \mathcal{N} \) is our design-idea model.
- Assume that the verification tool states \( \mathcal{N} \models Q \). What can we conclude from that?

<table>
<thead>
<tr>
<th>Tool Result</th>
<th>( \mathcal{N} \not\models Q )</th>
<th>( \mathcal{N} \models Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Idea</td>
<td>False Negative</td>
<td>True Positive</td>
</tr>
<tr>
<td>( \text{sat. } Q )</td>
<td>True Negative</td>
<td>False Positive</td>
</tr>
<tr>
<td>( \text{does not sat. } Q )</td>
<td>False Positive</td>
<td>True Positive</td>
</tr>
</tbody>
</table>

Content

- **Communicating Finite Automata (CFA)**
  - concrete and abstract syntax,
  - networks of CFA,
  - operational semantics.

- **Transition Sequences**

- **Deadlock, Reachability**

- **Uppaal**
  - tool demo (simulator),
  - query language,
  - CFA model-checking.

- **CFA at Work**
  - drive to configuration, scenarios, invariants
  - tool demo (verifier).

- **Uppaal Architecture**
Tell Them What You’ve Told Them...

- A network of communicating finite automata
  - describes a labelled transition system,
  - can be used to model software behaviour.

- The Uppaal Query Language can be used to
  - formalize reachability ($\exists CF, \forall CF, \ldots$) and
  - leadsto ($CF_1 \rightarrow CF_2$) properties.

- Since the model-checking problem of CFA is decidable,
  - there are tools which automatically check
    whether a network of CFA satisfies a given query.

- Use model-checking, e.g., to
  - obtain a computation path to a certain configuration
    (drive-to-configuration),
  - check whether a scenario is possible,
  - check whether an invariant is satisfied.
    (If not, analyse the design further using the obtained counter-example).
References