Softwaretechnik / Software-Engineering

Lecture 13: Behavioural Software Modelling

2016-06-27

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal
Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Architecture & Design: Content

- Introduction and Vocabulary
- Principles of Design
  - (i) modularity
  - (ii) separation of concerns
  - (iii) information hiding and data encapsulation
  - (iv) abstract data types, object orientation
- Software Modelling
  - (i) views and viewpoints, the 4+1 view
  - (ii) model-driven/-based software engineering
  - (iii) Unified Modelling Language (UML)
  - (iv) modelling structure
    - a) (simplified) class diagrams
    - b) (simplified) object diagrams
    - c) (simplified) object constraint logic (OCL)
  - (v) modelling behaviour
    - a) communicating finite automata
    - b) Uppaal query language
    - c) implementing CFA
    - d) an outlook on UML State Machines
- Design Patterns
- Testing: Introduction
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).
- **CFA vs. Software**

**Communicating Finite Automata**

presentation follows (Olderog and Dierks, 2008)
Channel Names and Actions

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.

- \((\alpha, \beta \in \text{Act}) = \{a? \mid a \in \text{Chan}\} \cup \{a! \mid a \in \text{Chan}\} \cup \{\tau\}\) 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 \(\text{V}\) using function symbols \(+, -, \ldots, >, \leq, \ldots\)

- A modification on \(v\) is \((v \leftarrow \varphi, v \in \text{V}, \varphi \in \Psi(\text{V}))\).

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

- By \(r\) we denote a finite list \((r_1, \ldots, r_n), n \in \mathbb{N}_0, \text{of modifications } r_i \in R(V)\).
  \((\emptyset)\) is the empty list \((n = 0)\).

- By \(R(V)^\ast\) 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

\[ (\ell, \alpha, \varphi, \vec{r}, \ell') \in E \wedge \text{chan}(\alpha) \in U \implies \varphi = \text{true}. \]

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}} is the initial location.

Example

\[ (\text{idle}, \text{WATER}?, \text{water_enabled}, \text{false}, \text{water_selected}) \in E \]
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 FCA $C(A_1, \ldots, A_n)$ is the labelled transition system

$$T(C(A_1, \ldots, A_n)) = (Conf, Chan \cup \{\tau\}, \{\lambda \in Chan \cup \{\tau\}, C_{ini})$$

where

- $V = \bigcup_{i=1}^n V_i$,
- $Conf = \{\langle \ell, \nu \rangle \mid \ell \in L_i, \nu : V \rightarrow \mathcal{D}(V)\}$,
- $C_{ini} = \langle \ell_{ini}, \nu_{ini} \rangle$ 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

- $\nu : V \rightarrow \mathcal{D}(V)$ is a valuation of the variables.
- A valuation $\nu$ of the variables canonically assigns an integer value $\nu(\varphi)$ to each integer expression $\varphi \in \Phi(V)$.
- $\models \subseteq (V \rightarrow \mathcal{D}(V)) \times \Phi(V)$ is the canonical satisfaction relation between valuations and integer expressions from $\Phi(V)$.

- Effect of modification $r \in R(V)$ on $\nu$, denoted by $\nu[r]$:

$$\nu(V := \varphi) := \begin{cases} \nu(\varphi), & \text{if } a = v, \\ \nu(a), & \text{otherwise} \end{cases}$$

- We set $\nu([r_1, \ldots, r_n]) := \nu[r_1] \cdots [r_n] = (((\nu[r_1])[r_2]) \cdots )[r_n]$.

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
  - \( \nu|\varphi = \varphi \)
  - \( \vec{\ell}' = \vec{\ell}[\ell_i := \ell_i'] \)
  - \( \nu' = \nu|\vec{r} \)

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
  - \( \nu|\varphi_i \land \varphi_j \)
  - \( \vec{\ell}' = \vec{\ell}[\ell_i := \ell_i'][\ell_j := \ell_j'] \)
  - \( \nu' = \nu|\vec{r}_i|\vec{r}_j \)

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

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

with

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

**Example**

```
water_enabled := false, soft_enabled := false, tea_enabled := false

DOK?
OK!

water_enabled := false, soft_enabled := false, tea_enabled := false

DTEA! DWATER! DSOFT!

water_enabled := true, soft_enabled := false, tea_enabled := false

TEA?
SOFT?
WATER?
```

**ChoicePanel:** (simplified)

- **User:**
  - C50!
  - E1!
  - WATER!
  - SOFT!
  - TEA!

- **Transition Diagram:**
  - State transitions:
    - idlec => idle => tea_selected => tea_enabled => TEA? => TEA!
    - water_enabled => WATER => WATER!
    - SOFT!
    - DWATER!
    - DSOFT!

- **Guard Conditions:**
  - $\text{guard tea_enabled not satisfied, and no comm. partners for C50}$
Deadlock, Reachability

- A configuration \( (\ell, \nu) \) of \( C(A_1, \ldots, A_n) \) is called **deadlock** if and only if there are no transitions from \( (\ell, \nu) \), i.e. if

  \[
  \neg (\exists \lambda \in \Lambda \exists (\ell', \nu') \in \text{Conf} \cdot (\ell, \nu) \xrightarrow{\lambda} (\ell', \nu')).
  \]

  The network \( C(A_1, \ldots, A_n) \) is said to have a deadlock if and only if there is a configuration \( (\ell, \nu) \) which is a deadlock.

- A configuration \( (\vec{\ell}, \nu) \) is called **reachable** (in \( C(A_1, \ldots, A_n) \)) if and only if there is a transition sequence of the form

  \[
  (\vec{\ell}_0, \nu_0) \xrightarrow{\lambda_1} (\vec{\ell}_1, \nu_1) \xrightarrow{\lambda_2} (\vec{\ell}_2, \nu_2) \xrightarrow{\lambda_3} \cdots \xrightarrow{\lambda_n} (\vec{\ell}_n, \nu_n) = (\vec{\ell}, \nu).
  \]

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

**Uppaal**

*(Larsen et al., 1997; Behrmann et al., 2004)*
Consider \( N = C(A_1, \ldots, A_n) \) over data variables \( V \).

- **basic formula:**
  
  \[
  \text{atom} ::= A_i \cdot \ell \mid \varphi \mid \text{deadlock}
  \]
  
  where \( \ell \in L_i \) is a location and \( \varphi \) an expression over \( V \).

- **configuration formulae:**
  
  \[
  \text{term} ::= \text{atom} \mid \text{not term} \mid \text{term}_1 \text{ and } \text{term}_2
  \]

- **existential path formulae:**
  
  \[
  e\text{-formula} ::= \exists \Diamond \text{term} \quad (\text{exists finally})
  \]
  
  \[
  \quad \mid \exists \Box \text{term} \quad (\text{exists globally})
  \]

- **universal path formulae:**
  
  \[
  a\text{-formula} ::= \forall \Diamond \text{term} \quad (\text{always finally})
  \]
  
  \[
  \quad \mid \forall \Box \text{term} \quad (\text{always globally})
  \]
  
  \[
  \quad \mid \text{term}_1 \rightarrow \text{term}_2 \quad (\text{leads to})
  \]

- **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:

- \( \langle \vec{\ell}, \nu \rangle \models \text{deadlock} \) iff \( \vec{\ell}_0, i \) is a deadlock configuration
- \( \langle \vec{\ell}, \nu \rangle \models A_i \cdot \ell \) iff \( \ell_0, i = \ell \)
- \( \langle \vec{\ell}, \nu \rangle \models \varphi \) iff \( \nu \models \varphi \)
- \( \langle \vec{\ell}, \nu \rangle \models \text{not term} \) iff \( \langle \vec{\ell}, \nu \rangle \not\models \text{term} \)
- \( \langle \vec{\ell}, \nu \rangle \models \text{term}_1 \text{ and term}_2 \) iff \( \langle \vec{\ell}, \nu \rangle \models \text{term}_1 \) and \( \langle \vec{\ell}, \nu \rangle \models \text{term}_2 \)

Example: Computation Paths vs. Computation Tree

ChoicePanel:

User:

\[
\begin{align*}
\langle \text{water\_selected}, l \rangle, \quad & \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \\
\langle \text{request\_sent}, l \rangle, \quad & \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \\
\langle \text{half\_idle}, l \rangle, \quad & \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right.
\end{align*}
\]

\[ \text{WATER} \]

\[ \langle \text{water\_selected}, l \rangle, \quad \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \]

\[ \langle \text{request\_sent}, l \rangle, \quad \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \]

\[ \langle \text{half\_idle}, l \rangle, \quad \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \]

\[ \text{SOFT} \]

\[ \langle \text{soft\_selected}, l \rangle, \quad \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \]

\[ \langle \text{request\_sent}, l \rangle, \quad \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \]

\[ \langle \text{half\_idle}, l \rangle, \quad \left\{ \begin{array}{l}
\text{water\_selected} = \text{false} \\
\text{tea\_selected} = \text{false} \\
\text{soft\_selected} = \text{false}
\end{array} \right. \]
Example: Computation Paths vs. Computation Tree

ChoicePanel:

User:

Example: Computation Paths vs. Computation Graph

ChoicePanel:

User:
Satisfaction of UPpaal Queries by Configurations

Exists finally:

- $\langle \vec{t}_0, \nu_0 \rangle \models \exists \diamond term$ iff $\exists \text{path } \xi \text{ of } C \text{ starting in } \langle \vec{t}_0, \nu_0 \rangle$

"some configuration satisfying term is reachable"

Example: $\langle \vec{t}_0, \nu_0 \rangle \models \exists \diamond \varphi$

Satisfaction of UPpaal Queries by Configurations

Exists globally:

- $\langle \vec{t}_0, \nu_0 \rangle \models \exists \square term$ iff $\exists \text{path } \xi \text{ of } C \text{ starting in } \langle \vec{t}_0, \nu_0 \rangle$

"on some computation path, all configurations satisfy term"

Example: $\langle \vec{t}_0, \nu_0 \rangle \models \exists \square \varphi$
Satisfaction of Uppaal Queries by Configurations

• Always globally:
  \[ \langle \vec{l}_0, \nu_0 \rangle \vdash \forall \square \text{term} \iff \langle \vec{l}_0, \nu_0 \rangle \nvdash \exists \lozenge \neg \text{term} \]

  “not (some configuration satisfying \(\neg\text{term}\) is reachable)”
  or: “all reachable configurations satisfy \text{term}”

• Always finally:
  \[ \langle \vec{l}_0, \nu_0 \rangle \vdash \forall \lozenge \text{term} \iff \langle \vec{l}_0, \nu_0 \rangle \nvdash \exists \square \neg \text{term} \]

  “not (on some computation path, all configurations satisfy \(\neg\text{term}\))”
  or: “on all computation paths, there is a configuration satisfying \text{term}”

Satisfaction of Uppaal Queries by Configurations

Leads to:
  \[ \langle \vec{l}_0, \nu_0 \rangle \vdash \text{term}_1 \rightarrow \text{term}_2 \iff \forall \text{path } \xi \text{ of } \mathcal{N} \text{ starting in } \langle \vec{l}_0, \nu_0 \rangle \forall i \in \mathbb{N}_0 \bullet \]
  \[ \xi^i \vdash \text{term}_1 \Rightarrow \xi^i \vdash \forall \lozenge \text{term}_2 \]

  “on all paths, from each configuration satisfying \text{term}_1,
  a configuration satisfying \text{term}_2 is reachable” (response pattern)

Example: \(\langle \vec{l}_0, \nu_0 \rangle \vdash \phi_1 \rightarrow \phi_2\)
**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**.

---

**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).

- **CFA vs. Software**
**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.
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

  \[ \mathcal{N}_{VM} \models \exists \Diamond w = 0. \]

  for the vending machine model \( \mathcal{N}_{VM} \).
References


