Softwaretechnik / Software-Engineering

Lecture 13: Behavioural Software Modelling

2015-06-29

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
(i) **Introduction and Vocabulary**

(ii) **Principles of Design**
   a) modularity
   b) separation of concerns
   c) information hiding and data encapsulation
   d) abstract data types, object orientation

(iii) **Software Modelling**
   a) views and viewpoints, the 4+1 view
   b) model-driven/based software engineering
   c) Unified Modelling Language (UML)
   d) **modelling structure**
      1. (simplified) class diagrams
      2. (simplified) object diagrams
      3. (simplified) object constraint logic (OCL)
   e) **modelling behaviour**
      1. communicating finite automata
      2. Uppaal query language
      3. basic state-machines
      4. an outlook on hierarchical state-machines

(iv) **Design Patterns**
Contents & Goals

Last Lecture:
- Class diagrams, object diagrams, (Proto-)OCL

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - What is a communicating finite automaton?
  - Which two kinds of transitions are considered in the CFA semantics?
  - Given a network of CFA, what are its computation paths?
  - Is this configuration / location reachable in the given CFA?

- Content:
  - Networks of Communicating Finite Automata
  - Uppaal Demo
  - Implementable CFA
Communicating Finite Automata

presentation follows (Olderog and Dierks, 2008)
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:** \(\text{Chan}?! = \text{Act}\).
Integers and Expressions, Resets

- Let \((v, w \in V)\) be a set of (finite domain) integer variables. By \((\varphi \in \Psi(V))\) we denote the set of integer expressions over \(V\) using function symbols +, -, \ldots, >, \neq, ...

- A modification on \(v\) is

\[
\begin{array}{c}
v := \varphi, \quad v \in V, \quad \varphi \in \Psi(V). \\
\end{array}
\]

By \(R(V)\) 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 \(r_i \in R(V)\); \(\langle \rangle\) is the empty list \((n = 0)\).

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

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

where

- \( (\ell \in) L \) is a finite set of locations (or control states),
- \( B \subseteq \text{Chan} \),
- \( V \): a set of data variables,
- \( E \subseteq L \times B \star \times \Phi(V) \times R(V)^\star \times L \): a 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.
- \( \ell_{ini} \) is the initial location.
Example

ChoicePanel:

1. idle
2. soft_selected
3. tea_selected
4. water_selected

Initial location: idle

Transition labels:
- WATER?: water_enabled := false, soft_enabled := false, tea_enabled := false
- SOFT?: soft_enabled := false
- TEA?: tea_enabled := false
- DOK?: (req_sent, DOK?, true, <>, half_idle)
- request_sent

States:
- idle
- soft_selected
- tea_selected
- water_selected
- half_idle
**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

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

where

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

The transition relation consists of transitions of the following two types.
• $\nu : V \to \mathcal{D}(V)$ is a valuation of the variables,

• A valuation $\nu$ of the variables canonically assigns an integer value $\nu(\varphi)$ to each integer expression $\varphi \in \Phi(V)$.

• $|= \subseteq (V \to \mathcal{D}(V)) \times \Phi(V)$ is the canonical satisfaction relation between valuations and integer expressions from $\Phi(V)$.

• $\varphi = x + y$, $\nu = \{ x \mapsto 3, y \mapsto 10 \}$

  $\nu(\varphi) = 13$

• $\nu \models x \neq 0$
• \( \nu : V \rightarrow \mathcal{D}(V) \) 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 \( \nu[r] \):

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

• We set \( \nu[\langle r_1, \ldots, r_n \rangle] := \nu[r_1] \ldots [r_n] = (((\nu[r_1])[r_2]) \ldots)[r_n] \).

That is, modifications are executed sequentially from left to right.

\[
\nu[\langle x := 3, y := x, x := 7 \rangle] = \begin{cases} 
x \mapsto 7, \\
y \mapsto 31
\end{cases}
\]
Operational Semantics of Networks of FCA

- An **internal transition** \( \langle \vec{l}, \nu \rangle \xrightarrow{\tau} \langle \vec{l}', \nu' \rangle \) occurs if there is \( i \in \{1, \ldots, n\} \) and

  - there is a \( \tau \)-edge \( (l_i, \tau, \varphi, \vec{r}, l'_i) \in E_i \) such that
    - \( \nu \models \varphi \), "source valuation satisfies guard"
    - \( \vec{l}' = \vec{l}[l_i := l'_i] \), "automaton \( i \) changes location"
    - \( \nu' = \nu[\vec{r}] \), "\( \nu' \) is \( \nu \) modified by \( \vec{r} \)"

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

  - there are edges \( (l_i, b!, \varphi_i, \vec{r}_i, l'_i) \in E_i \) and \( (l_j, b?, \varphi_j, \vec{r}_j, l'_j) \in E_j \) such that
    - \( \nu \models \varphi_i \land \varphi_j \),
    - \( \vec{l}' = \vec{l}[l_i := l'_i][l_j := l'_j] \),
    - \( \nu' = \nu[\vec{r}_i][\vec{r}_j] \), "output first, then input"

This style of communication is known under the names "**rendezvous**", "**synchronous**", "**blocking**" communication (and possibly many others).
A transition sequence of $C(A_1, \ldots, A_n)$ is any (in)finite sequence of the form

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

with

- $\langle \ell_0, \nu_0 \rangle = C_{\text{ini}}$,
- for all $i \in \mathbb{N}$, there is $\frac{\lambda_{i+1}}{}$ in $T(C(A_1, \ldots, A_n))$ with $\langle \ell_i, \nu_i \rangle \xrightarrow{\lambda_{i+1}} \langle \ell_{i+1}, \nu_{i+1} \rangle$.

A configuration $\langle \ell, \nu \rangle$ is called reachable (in $C(A_1, \ldots, A_n)$) if and only if there is a transition sequence of the form

$$\langle \ell_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \ell_2, \nu_2 \rangle \xrightarrow{\lambda_3} \ldots \xrightarrow{\lambda_n} \langle \ell_n, \nu_n \rangle = \langle \ell, \nu \rangle$$

A location $\ell_i$ is called reachable if and only if any configuration $\langle \ell, \nu \rangle$ is reachable, i.e. there exists a valuation $\nu$ such that $\langle \ell, \nu \rangle$ is reachable.

The network $C(A_1, \ldots, A_n)$ is said to have a deadlock if and only if there is a configuration $\langle \ell, \nu \rangle$ such that

$$\not\exists \frac{\lambda}{\rightarrow} \in T(C(A_1, \ldots, A_n)), \langle \ell', \nu' \rangle \in \text{Conf} \bullet \langle \ell, \nu \rangle \xrightarrow{\lambda} \langle \ell', \nu' \rangle.$$
Example

ChoicePanel:

User:

\[
\langle (idle, e), w_{en} = 1 \rangle \xrightarrow{\text{water}} \langle (w_{sel}, e), v \rangle \xrightarrow{\tau} \langle (\text{reg}_s, e), v \rangle
\]

\[
\langle (idle, e), s_{en} = 1 \rangle \xrightarrow{\text{soft}} \langle (s_{sel}, e), v \rangle \xrightarrow{\tau} \langle (\text{reg}_s, e), v \rangle
\]

deadlock
**Note:** Uppaal does not support the definition of scopes for channels — that is, ‘Service’ could send ‘WATER’ if the modeler wanted to...
Definition. **Software** is a finite description $S$ of a (possibly infinite) set $[[S]]$ of (finite or infinite) computation paths of the form

$$
\sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \cdots
$$

where

- $\sigma_i \in \Sigma$, $i \in \mathbb{N}_0$, is called **state** (or **configuration**), and
- $\alpha_i \in A$, $i \in \mathbb{N}_0$, is called **action** (or **event**).

The (possibly partial) function $[[\cdot]] : S \mapsto [[S]]$ is called **interpretation** of $S$.

- Let $C(A_1, \ldots, A_n)$ be a network of CFA.
- $\Sigma = \text{Conf}$
- $A = \text{Chan} \cup \{\tau\}$
- $[[C]] = \{\pi = \langle \vec{l}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{l}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{l}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \cdots | \pi \text{ is a computation path of } C\}$.
- **Note**: the structural model just consists of the set of variables and the locations of $C$. 

Uppaal

(Larsen et al., 1997; Behrmann et al., 2004)
**Definition.** The model-checking problem for a network $C$ of communicating finite automata and a query $F$ is to decide whether

$$(C, F) \in \models.$$

$\forall M \in C \quad w_0 \models F$$

**Proposition.** The model-checking problem for communicating finite automata is decidable.
Example: Invariants in the Model

ChoicePanel:

- \( w > 0 \)
- \( \text{water_enabled := false, soft_enabled := false, tea_enabled := false} \)
- \( \text{DOK?} \)
- \( \text{OK!} \)
- \( \text{half_idle} \)
- \( \text{DTEA!} \)
- \( \text{DSOFT!} \)
- \( \text{WATER?} \)
- \( \text{SOFT?} \)
- \( \text{TEA?} \)
- \( \text{tea_enabled} \)
- \( \text{water_enabled} \)
- \( \text{soft_enabled} \)
- \( \text{tea_selected} \)
- \( \text{water_selected} \)
- \( \text{request_sent} \)
Recall: Universal LSC Example

LSC: buy water
AC: true
AM: invariant I: strict

User → CoinValidator → ChoicePanel → Dispenser

- \neg(C50 \lor E1 \lor pSOFT \lor pTEA \lor pFILLUP)
- \neg(dSoft \lor dTEA)
- water\_in\_stock

\(C50\) → \(pWATER\) → \(dWATER\) → \(OK\) → Dispenser

\(pWATER\) → \(water\_in\_stock\)
Implementing Communicating Finite Automata
Implementing CFA

```haskell
st : { idle, wsel, ssel, tsel, reqs, half };

take_event( E : { TAU, WATER, SOFT, TEA, ... } ) {
  bool stable = 1;
  switch (st) {
    case idle :
      switch (E) {
        case WATER :
          if (water_enabled) { st := wsel; stable := 0; }
        ;;
        case SOFT :
          ...
        }
    case wsel:
      switch (E) {
        case TAU :
          send_DWATER(); st := reqs;
        ;;
      }
  }
}
```
Would be Too Easy...

- How are we supposed to implement that?
  - There is non-determinism in the upper automaton,
  - Internal transitions can interleave, one interleaving leads to a deadlock.

- We are not!

- We define
  - deterministic CFA,
  - a greedy semantics for internal transitions.

and only implement deterministic CFA using the greedy semantics.
The communicating finite automaton \( A = (L, B, V, E, \ell_{ini}) \) is called **deterministic** if and only if

- for each location \( \ell \),
  - either all edges with \( \ell \) as source location have pairwise different input actions,
  - or there is no edge with an input action starting at \( \ell \), and all edges starting at \( \ell \) have pairwise (logically) disjoint guards.
The communicating finite automaton $A = (L, B, V, E, \ell_{init})$ is called **deterministic** if and only if

- for each location $\ell$,
  - either all edges with $\ell$ as source location have pairwise different **input actions**, or there is no edge with an input action starting at $\ell$, and all edges starting at $\ell$ have pairwise (logically) disjoint guards.

Let each automaton in the network $C(A_1, \ldots, A_n)$ be marked as either **environment** or **controller**. We call $C$ **implementable** if and only if, for each **controller** $A$ in $C$,

- (i) $A$ is deterministic,
- (ii) $A$ reads/writes only its local variables, may also read variables written by environment automata, but only in modification vectors of edges with input synchronisation,
- (iii) $A$ is **locally deadlock-free**, i.e. enabled edges with output-actions are not blocked forever.

**Note:** implementable (i) and (ii) can be checked syntactically. Property (iii) is a property of the whole network.

Can be checked with Uppaal:

$$ (A.\ell \land \varphi) \longrightarrow (A.\ell') $$

for each edge $(\ell, \alpha, \varphi, \vec{r}, \ell')$ of $A$. 
Greedy CFA Semantics

- **Greedy** semantics:
  - each input synchronisation transition (plus: system start) of automaton $A$ is followed by a maximal sequence of internal transitions or output transitions of $A$.
  - **Maximal**: cannot be extended by an internal transition.

There may still be interleaving of the internal transitions, but (by forbidding shared variables for controllers) cannot be observed outside of an automaton.

**Example:**

$A_1$:

$A_{2,1}$:

$A_{2,2}$:

$E$: 

$\mathcal{E}$:

$A_1$ is implementable in $C(A_1, A_{2,1}, \mathcal{E})$ (environment: only $\mathcal{E}$)

- deterministic: ✔,
- only local variables, environment variables with input: ✔,
- locally deadlock-free: ✔.

$A_1$ is **not** implementable in $C(A_1, A_{2,2}, \mathcal{E})$. 
Now an implementable model $C(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ has two semantics:

- $\llbracket C \rrbracket_{std}$ — standard semantics.
- $\llbracket C \rrbracket_{grd}$ — greedy semantics.

Are they related in any way?
References
References


