Contents of the Block “Design”

(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)
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.

- \((\alpha, \beta \in \text{Act}) := \{a? | a \in \text{Chan}\} \cup \{a! | a \in \text{Chan}\} \cup \{\tau\}\) 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

  \[B?!= := \{a? | a \in B\} \cup \{a! | a \in B\} \cup \{\tau\}.\]

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

Integer Variables 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 \(+, -, \ldots, >, \times, \ldots\).

- A modification on \(v\) is

  \[v := \varphi, \quad v \in V, \quad \varphi \in \Psi(V).\]

  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
\[ A = (L, B, V, E, \ell_{\text{ini}}) \]
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 set of directed edges such that
  \[ (\ell, \alpha, \varphi, \vec{r}, \ell') \in E \land \text{chan}(\alpha) \in U \implies \varphi = \text{true} \].
- \( \ell_{\text{ini}} \) is the initial location.

Example

ChoicePanel:

\[ L = \{ \text{idle, water\_selected, ...} \} \]
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

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

where

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

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

 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 $\nu(\varphi)$ to each integer expression $\varphi \in \Phi(V)$.
- $\models \subseteq (V \rightarrow \mathcal{P}(V)) \times \Phi(V)$ is the canonical satisfaction relation between valuations and integer expressions from $\Phi(V)$.

- $\varphi = x + y$, $\nu = \{x \mapsto 3, y \mapsto 10\}$
  $$\nu(\varphi) = 13$$
- $\varphi = x \neq 0$
• $\nu : V \rightarrow \mathcal{P}(V)$ is a valuation of the variables,

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

• $\models \subseteq (V \rightarrow \mathcal{P}(V)) \times \Phi(V)$ 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 $\nu[r_1, \ldots, r_n] := \nu[r_1] \ldots [r_n] = (((\nu[r_1])[r_2]) \ldots)[r_n].$

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

\[\nu[x := 3, y := 10] = \{ x \mapsto 3, y \mapsto 10 \}\]

**Operational Semantics of Networks of FCA**

• 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
    • $\nu |\varphi$, "some valuation satisfies guard"
    • $\vec{\ell}' = \ell_i[\ell_i := \ell_i']$, "automaton $i$ changes location"
    • $\nu' = \nu[r_i]$, "$\nu'$ is $\nu$ modified by $\vec{r}$"

• 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 $(\ell_i, b_i, \varphi_i, \vec{r}_i, \ell_i') \in E_i$ and $(\ell_j, b_j, \varphi_j, \vec{r}_j, \ell_j') \in E_j$ such that
    • $\nu |\varphi_i \wedge \varphi_j$, "output first, the input"
    • $\vec{\ell}' = \ell_i[\ell_i := \ell_i']$, $\vec{\ell}_j[\ell_j := \ell_j']$,
    • $\nu' = \nu[r_i][r_j]$, "synchronisation" communication (and possibly many others).
A transition sequence of $C(A_1, \ldots, A_n)$ is any (in)finite sequence of the form

$$\langle \ell_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \ell_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \ell_2, \nu_2 \rangle \xrightarrow{\lambda_3} \ldots$$

with

- $\langle \ell_0, \nu_0 \rangle = C_{ini}$,
- for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ 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

$$\exists \lambda \in T(C(A_1, \ldots, A_n)), \langle \ell', \nu' \rangle \in Conf \bullet \langle \ell, \nu \rangle \xrightarrow{\lambda} \langle \ell', \nu' \rangle.$$
Model Architecture — Who Talks What to Whom

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 $[\cdot]: S \rightarrow [S]$ is called interpretation of $S$.

- Note: Uppaal does not support the definition of scopes for channels — that is, ‘Service’ could send ‘WATER’ if the modeler wanted to…

A CFA Model Is Software

Let $\mathcal{C}(A_1, \ldots, A_n)$ be a network of CFA.

- $\Sigma = \text{Conf}$
- $A = \text{Chan} \cup \{\tau\}$
- $[\mathcal{C}] = \{ \pi = (\vec{e}_0, \nu_0) \xrightarrow{\lambda_1} (\vec{e}_1, \nu_1) \xrightarrow{\lambda_2} (\vec{e}_2, \nu_2) \xrightarrow{\lambda_3} \cdots \mid \pi \text{ is a computation path of } \mathcal{C} \}$.

- Note: the structural model just consists of the set of variables and the locations of $\mathcal{C}$.
CFA Model-Checking

Definition. The model-checking problem for a network $\mathcal{C}$ of communicating finite automata and a query $F$ is to decide whether

$$(\mathcal{C}, F) \in \models.$$

$\mathcal{C} \models F$

$\forall \mathcal{K} \in \mathcal{C} \neq \emptyset$ 

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

ChoicePanel:

Uppaal Architecture
Recall: Universal LSC Example

Implementing Communicating Finite Automata
Implementing CFA

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.
**Deterministic CFA**

- 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.

- 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:
  $$(A, \ell \wedge \varphi) \rightarrow (A, \ell')$$
  for each edge $(\ell, \alpha, \varphi, r, \ell')$ of $A$. 

\[ E? \quad F? \quad \ell \quad \ell' \quad (\varphi \wedge \varphi') \rightarrow \neq \varphi' \]
**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:**

$$G^?$$

$$F!$$

$$n := n + 1$$

$$E?$$

$$v := v_{env}$$

$$E!$$

$$v_{env} > -10$$

$$v_{env} := v_{env} - 1$$

$$v_{env} < 10$$

$$v_{env} := v_{env} + 1$$

$$G!$$

$$F!$$

$$G!$$

**$A_1$:**

- $A_1$ is implementable in $C(A_1, A_2, E)$ (environment: only $E$)
  - deterministic: ✔
  - only local variables, environment variables with input: ✔
  - locally deadlock-free: ✔

- $A_1$ is **not** implementable in $C(A_1, A_2, E)$.

**Model vs. Implementation**

- Now an implementable model $C(A_1, \ldots, A_n)$ has **two semantics**:
  - $[C]_{std}$ — standard semantics.
  - $[C]_{grd}$ — greedy semantics.

- Are they **related** in any way?
References