Topic Area Architecture & Design: Content

- **VL 11**
  - Introduction and Vocabulary
  - Software Modelling
    - Model, views / viewpoints: 4+1 view
  - Modelling structure
    - (simplified) class & object diagrams
    - (simplified) object constraint logic (OCL)

- **VL 12**

- **VL 13**
  - Principles of Design
    - Modularity, separation of concerns
    - Information hiding and data encapsulation
    - Abstract data types, object orientation

- **VL 14**
  - Design Patterns
  - 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

- **VL 15**

- **Model-driven/-based Software Engineering**
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

Software Modelling
Example
Channel Names and Actions

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

- A set \((a, b \in)\) Chan of channel names or channels.

- For each channel \(a \in\) Chan, two visible actions: \(a?\) and \(a!\) denote input and output on the channel \((a? , a! \notin\) Chan).

- \(\tau \notin\) Chan represents an internal action, not visible from outside.

- \((\alpha, \beta \in)\) Act := \({a? | a \in Chan}\) \(\cup\) \({a! | a \in Chan}\) \(\cup\) \{\(\tau\)\} is the set of actions.

- An alphabet \(B\) is a set of channels, i.e. \(B \subseteq\) Chan.

- For each alphabet \(B\), we define the corresponding action set

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

Note: Chan_? = Act.

Integer Variables and Expressions, Resets

- Let \((v, w \in)\) V be a set of (finite domain) integer variables. Including 0.

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

- A modification on \(v \in V\) is of the form

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

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

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

\[\langle\rangle\] 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 finite set of directed edges such that
  \[ (\ell, \alpha, \varphi, \vec{r}, \ell') \in E \land \text{chan}(\alpha) \in U \Rightarrow \varphi = \text{true} \].

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

\[ L = \{ \text{idle, water\_selected, } \ldots \} \]
\[ B = \{ \text{WATER, OK, } \ldots \} \]
\[ V = \{ \text{water\_enabled, } \ldots \} \]

ChoicePanel: (simplified)

\[ E = \{ \text{(idle, WATER, water\_enabled, false, water\_selected), } \ldots \} \]
\[ \ell_{\text{ini}} = \text{idle} \]
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 \textit{operational semantics} of the \textit{network} of CFA \( C(A_1, \ldots, A_n) \) is the labelled transition system
\[
T(C(A_1, \ldots, A_n)) = (\text{Conf}, \text{Chan} \cup \{\tau\}, \{\lambda \mapsto \lambda \in \text{Chan} \cup \{\tau\}\}, C_{\text{ini}})
\]
where
- \( V = \bigcup_{i=1}^{n} V_i \times L_i \times L_2 \times \cdots \times L_n \)
- \( \text{Conf} = \{ (\vec{\ell}, \nu) \mid \vec{\ell}_i \in L_i, \nu : V \rightarrow \mathcal{P}(V) \} \)
- \( C_{\text{ini}} = (\vec{\ell}_{\text{ini}}, \nu_{\text{ini}}) \) with \( \nu_{\text{ini}}(v) = 0 \) for all \( v \in V \).

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

\( \tau \rightarrow \omega \)

\[ \nu \mapsto \nu' \]

**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 \( r \in R(V) \) on \( \nu \), denoted by \( \nu[r] \):
  \[ \nu[r] := \nu(\phi), \text{if } a = v, \nu(a), \text{otherwise} \]
- We set \( \nu[\{r_1, \ldots, r_n\}] := \nu[r_1] \ldots \nu[r_n] = (((\nu[r_1])|r_2|) \ldots)[r_n] \).

That is, modifications are executed sequentially from left to right.
An internal transition \( \langle \vec{e}, \nu \rangle \xrightarrow{\tau} \langle \vec{e}', \nu' \rangle \) 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
  - \( \nu \models \varphi \), "source valuation satisfies guard"
  - \( \vec{e}' = \ell_i[\ell_i := \ell_i'] \), "automaton \( i \) changes location"
  - \( \nu' = \nu[\vec{r}] \), "\( \nu' \) is the result of applying \( \vec{r} \) on \( \nu \)"

A synchronisation transition \( \langle \vec{e}, \nu \rangle \xrightarrow{b} \langle \vec{e}', \nu' \rangle \) 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
  - \( \nu \models \varphi_i \land \varphi_j \), "source valuation satisfies guards (!)"
  - \( \vec{e}' = \ell_i[\ell_i := \ell_i'] \ell_j[\ell_j := \ell_j'] \), "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).

Example

**ChoicePanel:** (simplified)
**Transition Sequences**

- A transition sequence of $C(A_1, \ldots, A_n)$ is any (in)finitesimal sequence of the form
  \[
  (\ell_0, \nu_0) \xrightarrow{\lambda_1} (\ell_1, \nu_1) \xrightarrow{\lambda_2} (\ell_2, \nu_2) \xrightarrow{\lambda_3} \ldots
  \]
  with
  - $(\ell_0, \nu_0) = C_{\text{ini}}$ (without "start from")
  - for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ in $T(C(A_1, \ldots, A_n))$ with $(\ell_i, \nu_i) \xrightarrow{\lambda_{i+1}} (\ell_{i+1}, \nu_{i+1})$.

---

**Deadlock**

- 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
  \[
  \neg (\exists \lambda \in \Lambda \exists (\ell', \nu') \in \text{Conf} \bullet (\ell, \nu) \xrightarrow{\lambda} (\ell', \nu')).
  \]
  The network $C(A_1, \ldots, A_n)$ is said to have a deadlock
  if and only if there is a reachable configuration $(\ell, \nu)$ which is a deadlock.
A configuration $\langle \vec{\ell}, \nu \rangle$ is called \textbf{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 \textbf{reachable} (without “from”!) if and only if it is reachable from $C_{\text{ini}}$.

A location $\ell \in L_i$ is called \textbf{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.

\textbf{Uppaal}

\textit{(Larsen et al., 1997; Behrmann et al., 2004)}
The Uppaal Query Language

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

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

- **configuration formulae:**
  \[ term ::= atom \mid \text{not} \ term \mid term_1 \ and \ term_2 \]

- **existential path formulae:**
  \[ e\text{-formula} ::= \exists \diamond term \]
  \[ \exists \Box term \]

- **universal path formulae:**
  \[ a\text{-formula} ::= \forall \Diamond term \]
  \[ \forall \Box term \]
  \[ term_1 \rightarrow term_2 \]

- **formulae (or queries):**
  \[ F ::= e\text{-formula} \mid a\text{-formula} \]
The satisfaction relation \( \langle \ell, \nu \rangle \models F \) between configurations \( \langle \ell, \nu \rangle = \langle (\ell_1, \ldots, \ell_n), \nu \rangle \) of a network \( C(A_1, \ldots, A_n) \) and formulae \( F \) of the Uppaal logic is defined inductively as follows:

- \( \langle \ell, \nu \rangle \models \text{deadlock} \) iff \( \ell_0, i \) is a deadlock configuration
- \( \langle \ell, \nu \rangle \models A_i \cdot \ell \) iff \( \ell_0, i = \ell \)
- \( \langle \ell, \nu \rangle \models \varphi \) iff \( \nu \models \varphi \)
- \( \langle \ell, \nu \rangle \models \text{not term} \) iff \( \langle \ell, \nu \rangle \not\models \text{term} \)
- \( \langle \ell, \nu \rangle \models \text{term}_1 \text{ and term}_2 \) iff \( \langle \ell, \nu \rangle \models \text{term}_i \), \( i = 1, 2 \)

Example: Computation Paths vs. Computation Tree

ChoicePanel:

User:
Satisfaction of Uppaal Queries by Configurations

Exists finally:

\* \( \vec{\ell}_0, \nu_0 \models \exists \Diamond \text{term} \)

iff \( \exists \text{path } \xi \text{ of } N \text{ starting in } \langle \vec{\ell}_0, \nu_0 \rangle \)

\[ \exists i \in \mathbb{N}_0 \cdot \xi^i \models \text{term} \]

"some configuration satisfying term is reachable"

Example: \( \langle \vec{\ell}_0, \nu_0 \rangle \models \exists \Diamond \varphi \)
Satisfaction of Uppaal Queries by Configurations

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

"on some computation path, all configurations satisfy term"

Example: \( \langle \vec{\ell}_0, \nu_0 \rangle \models \exists \square \varphi \)

Satisfaction of Uppaal Queries by Configurations

• Always globally:

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

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

• Always finally:

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

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

- \( \langle \vec{ℓ_0}, ν_0 \rangle \models \text{term}_1 \rightarrow \text{term}_2 \) iff \( \forall \) path \( ξ \) of \( N \) starting in \( \langle \vec{ℓ_0}, ν_0 \rangle \) \( ∀ i \in N_0 \ ⋅ \), \( \xi^i \models \text{term}_1 \implies \text{ξ}^i \models \forall \, \Diamond \text{term}_2 \)

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

Example: \( \langle \vec{ℓ_0}, ν_0 \rangle \models \varphi_1 \rightarrow \varphi_2 \)

CFA Model-Checking

Definition. Let \( N = C(A_1, \ldots, A_n) \) be a network and \( F \) a query.

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

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

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).

- Uppaal Architecture

CFA and Queries at Work
• **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

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

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  
  
  User  
  
  Coin Validator  
  
  Choice Panel  
  
  C  
  
  C  
  
  C  
  
  TEA  
  
  ¬E1  

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

  \[ \text{end_of_scenario} \rightarrow \text{TEA}! \rightarrow \text{C50}! \rightarrow \text{C50}! \rightarrow \text{C50}! \]

  instead of User and check whether location end_of_scenario is reachable, i.e. check

  \[ \mathcal{N}_{VM} \models \exists \text{Scenario}.\text{end_of_scenario}. \]

  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 \( \mathcal{E}\.50 \) 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

  \[ \mathcal{N}_{VM} \models \forall \Box \text{tea_enabled} \implies \text{CoinValidator\_have\_c150} \]

  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.

---

**Design Verification: Another Invariant**

- **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
  \[ \mathcal{N}_V \models \forall (\text{CoinValidator\_have\_c50 or CoinValidator\_have\_c100 or CoinValidator\_have\_c150}) \]
  imply \( \text{water\_enabled} \).
Recall: Universal LSC Example

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

User
CoinValidator
ChoicePanel
Dispenser

¬ (C50 ∨ E1 ∨ pSOFT! ∨ pTEA! ∨ pFILLUP!)
water_in_stock

water

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
Uppaal Architecture
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