• 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

Communicating Finite Automata

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 \((a? / a! \in \text{Chan})\).

• \(\tau / \in \text{Chan}\) 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 \((v, w \in \text{V})\) be a set of (finite domain) integer variables.

• By \((\phi \in \text{Ψ(\text{V})})\) we denote the set of integer expressions over \(\text{V}\) using function symbols \(+, -, \ldots\).

• A modification on \(v\) is \(v := \phi, v \in \text{V}, \phi \in \text{Ψ(\text{V})}\).

• By \(\text{R(\text{V})}\) 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 \(r_i \in \text{R(\text{V})}\).

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

• By \(\text{R(\text{V})}^*\) we denote the set of all such finite lists of modifications.
Example of Communicating Finite Automata (CFA) and Their Interpretation.
always globally

exists finally

exists globally

Consider Tool Demo

reachable, i.e. there exist \( \vec{\ell}, \nu \in L \) with

\[ A \xrightarrow{\lambda} \ldots \xrightarrow{\lambda} 0 \]

which is a deadlock.

\[ \text{water_enabled := false, soft_enabled := false, tea_enabled := false} \]

\[ \text{simplified} \]

\[ \text{(simplified)} \]

\[ \text{Example} \]

Transition Sequences

The Event Graph Language
Example: Computation Paths vs. Computation Trees

Example: Computation Paths vs. Computation Trees
Satisfaction of Uppaal Queries by Configurations

• Always globally:
  \[ \langle \vec{\ell}_0, \nu_0 \rangle \models \forall \Box \text{term} \iff \langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists \Diamond \neg \text{term} \]
  "not (some configuration satisfying \( \neg \text{term} \) is reachable)"
or: "all reachable configurations satisfy \( \text{term} \)"

• Always finally:
  \[ \langle \vec{\ell}_0, \nu_0 \rangle \models \forall \Diamond \text{term} \iff \langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists \Box \neg \text{term} \]
  "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:

\[ \langle \vec{\ell}_0, \nu_0 \rangle \models \text{term}_1 \rightarrow \text{term}_2 \iff \forall \text{path } \xi \text{ of } N \text{ starting in } \langle \vec{\ell}_0, \nu_0 \rangle \forall i \in N_0 \cdot \xi_i \models \text{term}_1 \Rightarrow \xi_i \models \forall \Diamond \text{term}_2 \]
"on all paths, from each configuration satisfying \( \text{term}_1 \), a configuration satifying \( \text{term}_2 \) is reachable" (response pattern)

Example:
\[ \langle \vec{\ell}_0, \nu_0 \rangle \models \phi_1 \rightarrow \phi_2 \]

\begin{align*}
\phi_1 & = \neg \phi_2 \\
\phi_2 & = \phi_1 \wedge \neg \phi_2
\end{align*}

CFA and Queries at Work

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

CoinValidator
User
ChoicePanel
WaterDispenser
SoftDispenser
TeaDispenser
Service
C50, E1
WATER, SOFT, TEA
OK
DWATER
DSOFT
DTEA
DOK
FILLUP

ENVIRONMENTSYSTEM

half_idle
request_sent
tea_selected
soft_selected
water_selected
idle
DOK?
OK!
water_enabled := false,soft_enabled := false,tea_enabled := false
DTEA!
DWATER!
DSOFT!
tea_enabled
soft_enabled
water_enabled
WATER?

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.

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
\[ \exists \, \delta \mid N_{VM} \models w = 0. \]

References


