Softwaretechnik / Software-Engineering

Lecture 14: Behavioural Software Modelling

2018-06-28

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

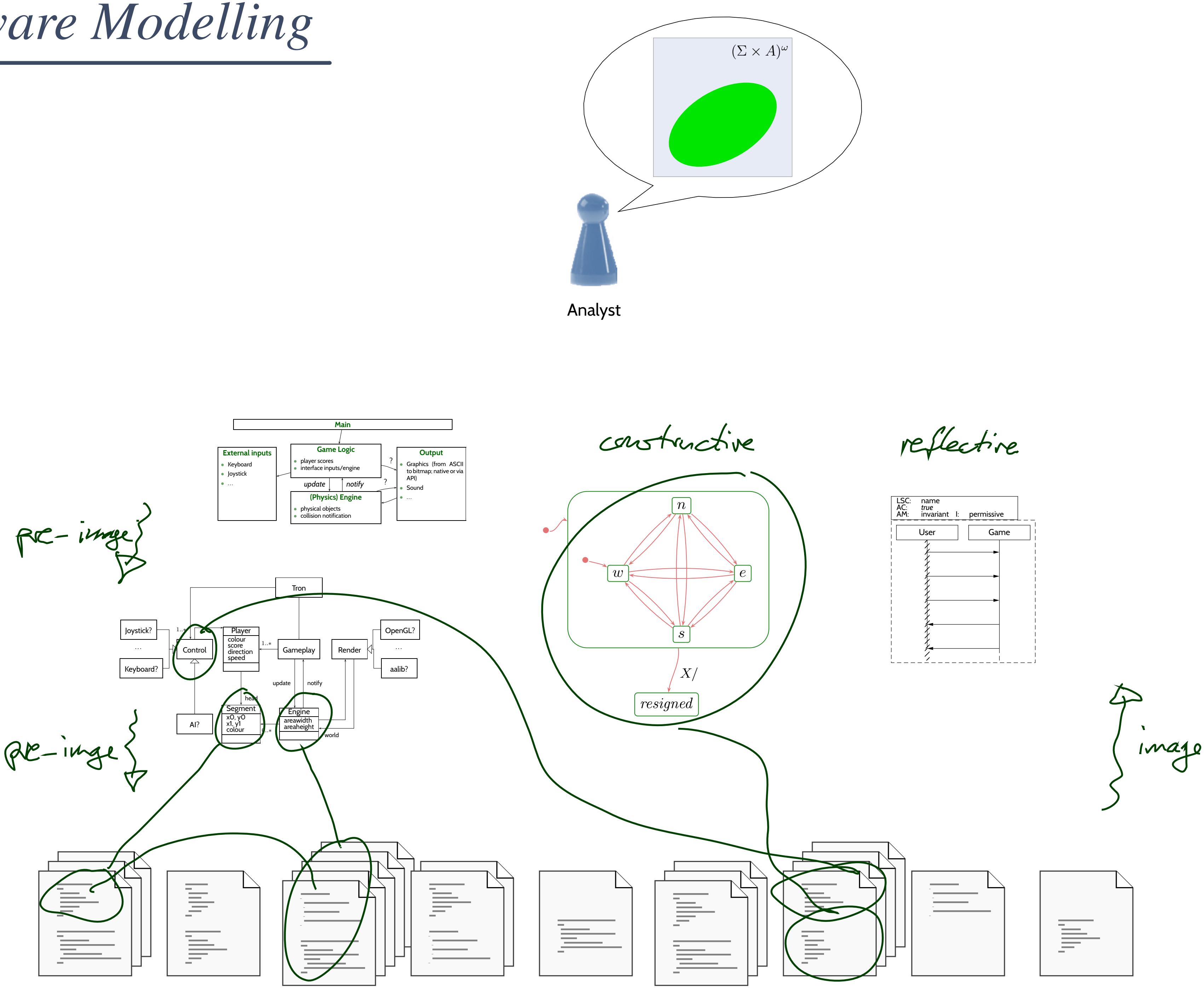
Topic Area Architecture & Design: Content

VL 11	• **Introduction and Vocabulary**
	• **Software Modelling**
⋮	• model; views / viewpoints; 4+1 view
	• **Modelling structure**
	• (simplified) class & object diagrams
VL 12 ⋮	• (simplified) object constraint logic (OCL)
VL 13	• **Principles of Design**
	• modularity, separation of concerns
⋮	• information hiding and data encapsulation
	• abstract data types, object orientation
	• **Design Patterns**
VL 14	• **Modelling behaviour**
	• communicating finite automata (CFA)
⋮	• Uppaal query language
VL 15	• CFA vs. Software
	• Unified Modelling Language (UML)
⋮	• basic state-machines
	• an outlook on hierarchical state-machines
	• **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**

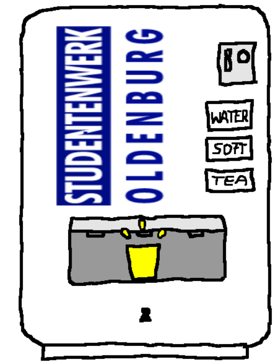
Software Modelling



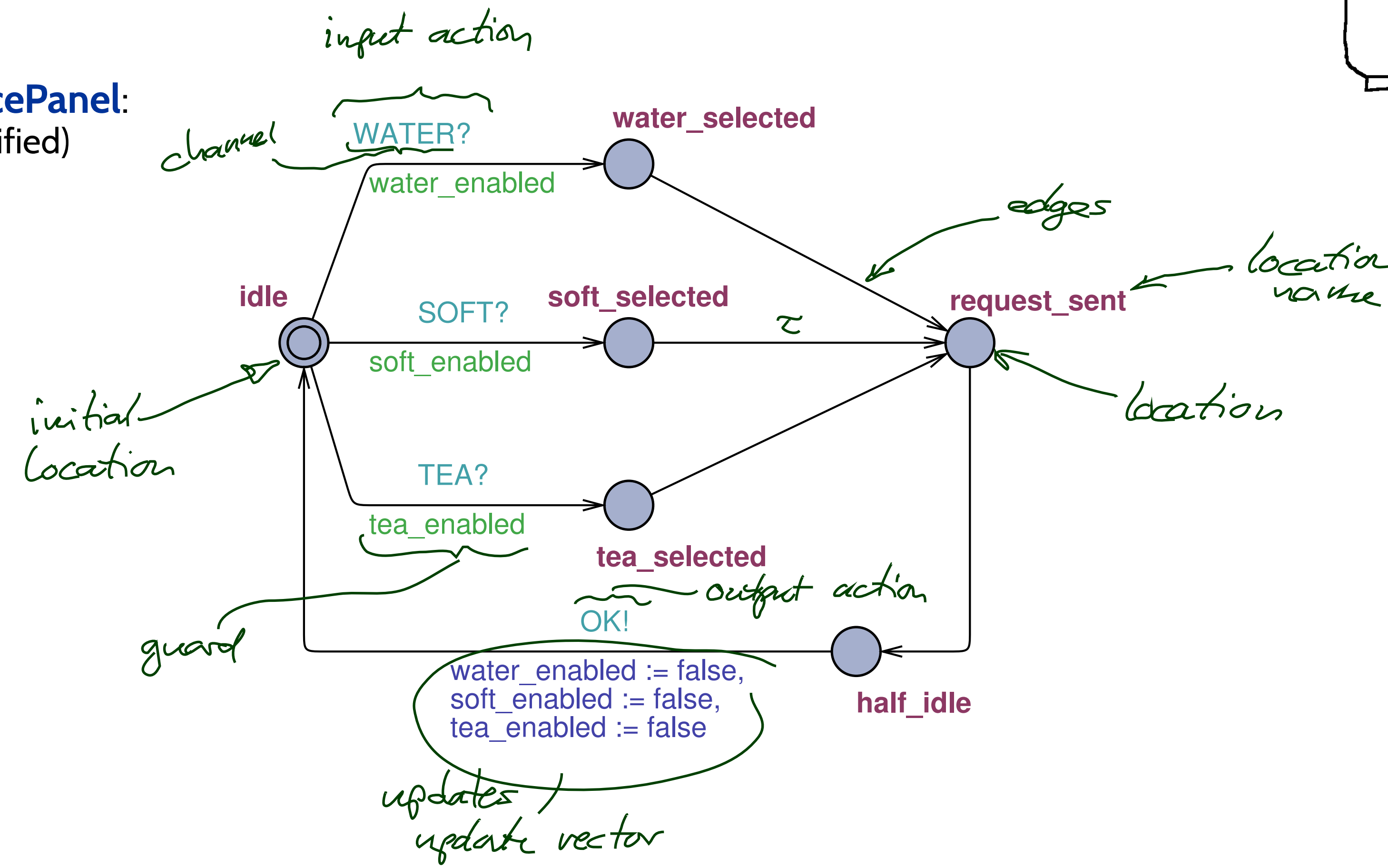
Communicating Finite Automata

presentation follows (Olderog and Dierks, 2008)

Example



ChoicePanel:
(simplified)



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.
- $(\alpha, \beta \in)$ $Act := \{a? \mid a \in \text{Chan}\} \cup \{a! \mid a \in \text{Chan}\} \cup \{\tau\}$ is the set of **actions**.
- An **alphabet** B is a set of **channels**, i.e. $B \subseteq$ 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: $\text{Chan}_{?!} = Act$.

Integer Variables and Expressions, Resets

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

  $v+w$

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

  $v<w$

- A **modification** on $v \in V$ is of the form
  or update

$$v := \varphi, \qquad v \in V, \quad \varphi \in \Psi(V).$$

  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 $r_i \in R(V)$.
  $\vec{r}$ 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.

Communicating Finite Automata

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**),
- $B \subseteq$ 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

  $$(\ell, \alpha, \varphi, \vec{r}, \ell') \in E \wedge \mathrm{chan}(\alpha) \in U \implies \varphi = \mathit{true}.$$

  Edges $(\ell, \alpha, \varphi, \vec{r}, \ell')$ from location ℓ to ℓ' are labelled with an **action** α, a **guard** φ, and a list $\vec{r}$ of **modifications**.
- $\ell_{ini} \in L$ is the **initial location**.

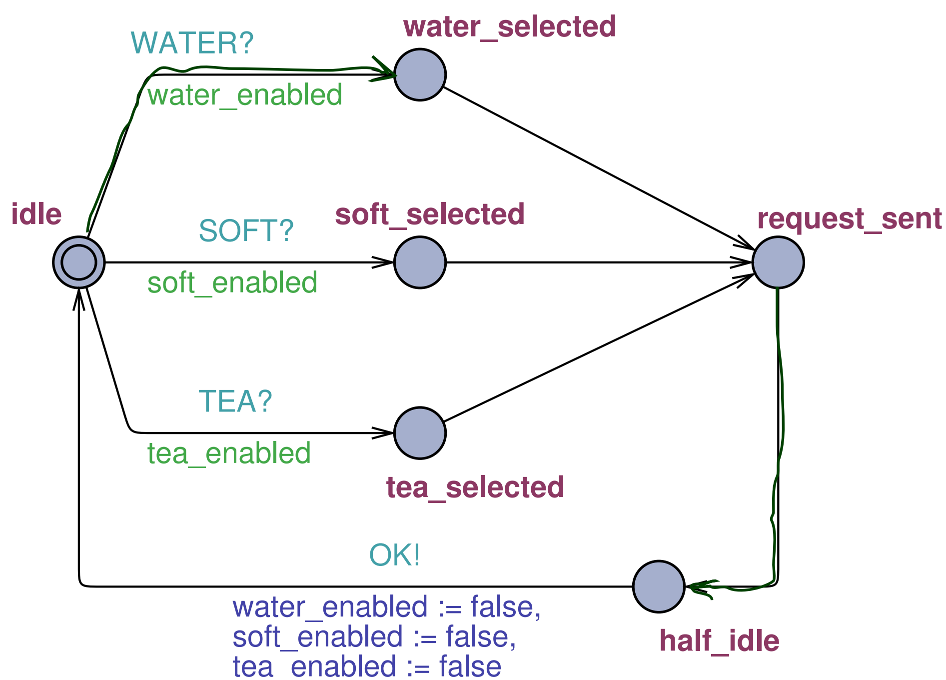
Example

$L = \{idle, water\_selected, \dots\}$

$B = \{WATER, OK, \dots\}$

$V = \{water\_enabled, \dots\}$

ChoicePanel:
(simplified)



$$E = \{ (idle, WATER?, water\_enabled, \langle\rangle, water\_selected),$$
$$(request\_sent, \tau, true, \langle\rangle, half\_idle), \dots \}$$

$$\in L \qquad \in B_{?!} \qquad \in \Phi(V) \qquad \in R(V)^* \qquad \in L$$

$l_{ini} = idle$

Definition.
Let $\mathcal{A}_i = (L_i, B_i, V_i, E_i, \ell_{ini,i})$, $1 \leq i \leq n$, be communicating finite automata.

The **operational semantics** of the **network** of CFA $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$
is the labelled transition system

$$\mathcal{T}(\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)) = (Conf, \underbrace{\mathsf{Chan} \cup \{\tau\}}_{\text{labels}}, \{\xrightarrow{\lambda} \mid \lambda \in \mathsf{Chan} \cup \{\tau\}\}, C_{ini})$$

where

- $V = \bigcup_{i=1}^{n} V_i$, $\in L_1 \times L_2 \times \dots \times L_n$
- $Conf = \{\langle \vec{\ell}, \nu \rangle \mid \ell_i \in L_i, \nu : V \to \mathscr{D}(V)\}$, valuation of 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

- $\nu : V \to \mathscr{D}(V)$ is a **valuation** of the variables,
- A valuation ν of the variables canonically assigns an integer value $\nu(\varphi)$ to each integer expression $\varphi \in \Phi(V)$.

  $\nu(v),\ \nu(w) \rightsquigarrow \nu(v+w$

- $\models \subseteq (V \to \mathscr{D}(V)) \times \Phi(V)$ is the canonical **satisfaction relation** between valuations and integer expressions from $\Phi(V)$.

  $\nu \models v > w$

- **Effect of modification** $r \in R(V)$ **on** ν, 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.

$\vec{r}$ $\quad w := w+1,\ w := 0$

$\nu(w) = 27$

$\nu[\vec{r}] := 0$

Operational Semantics of Networks of CFA

$(\langle\vec{\ell},\nu\rangle, \langle\vec{\ell}',\nu'\rangle) \in \xrightarrow{\tau} \left(\in \{ \xrightarrow{\lambda} \mid \lambda \text{ label} \} \right)$

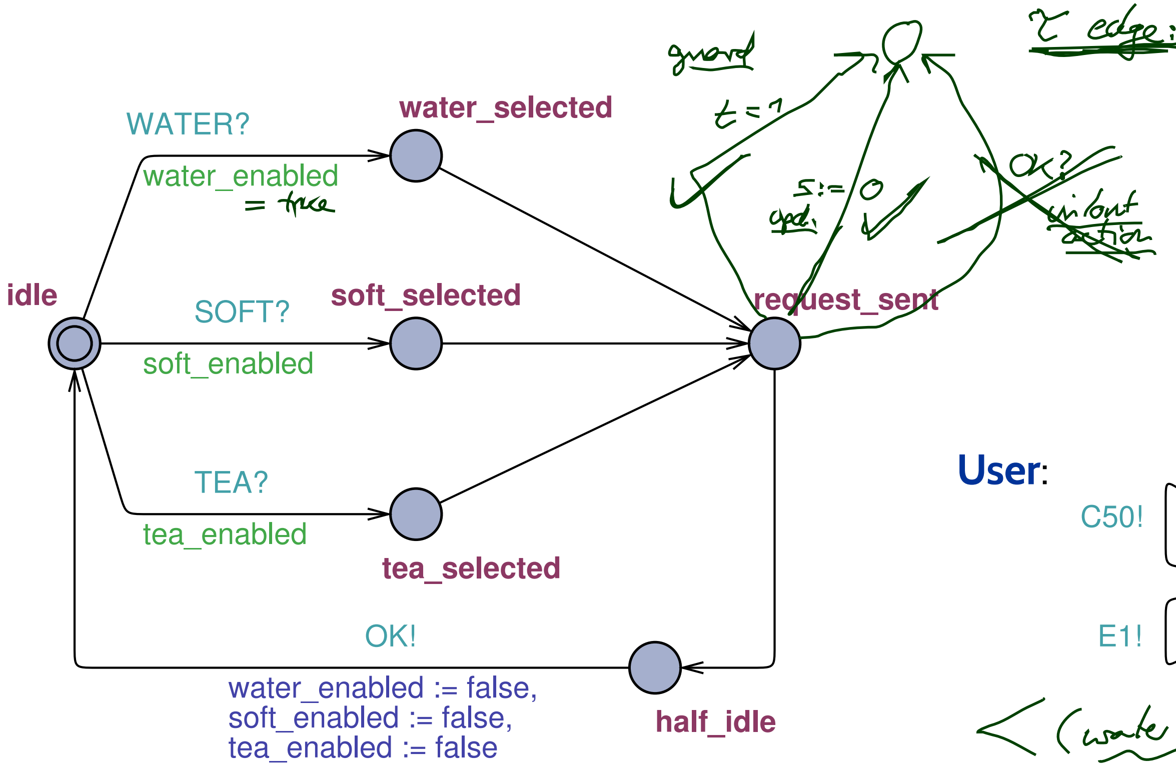
- An **internal transition** $\langle\vec{\ell},\nu\rangle \xrightarrow{\tau} \langle\vec{\ell}',\nu'\rangle$ occurs if there is $i \in \{1,\dots,n\}$ and
  - there is a τ-edge $(\ell_i, \tau, \varphi, \vec{r}, \ell_i') \in E_i$ such that — i-th component of $\vec{\ell}$
    - $\nu \models \varphi$, — "source valuation satisfies guard"
    - $\vec{\ell}' = \vec{\ell}[\ell_i := \ell_i']$, — "automaton i changes location"
    - $\nu' = \nu[\vec{r}]$, — "ν' is the result of applying $\vec{r}$ on ν"

- A **synchronisation transition** $\langle\vec{\ell},\nu\rangle \xrightarrow{b} \langle\vec{\ell}',\nu'\rangle$ occurs if there are $i, j \in \{1,\dots,n\}$ 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
    - $\nu \models \varphi_i \wedge \varphi_j$, — "source valuation satisfies guards (!)"
    - $\vec{\ell}' = \vec{\ell}[\ell_i := \ell_i'][\ell_j := \ell_j']$, — "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 ν"

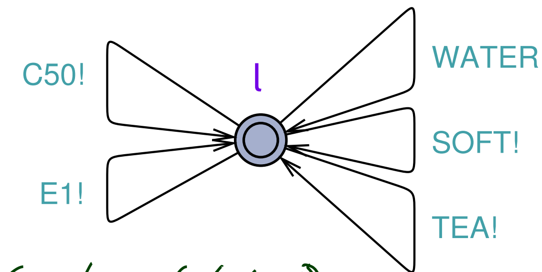
This style of communication is known under the names "**rendezvous**", "**synchronous**", "**blocking**" communication (and possibly many others).

Example

ChoicePanel:
(simplified)



User:



< (water_selected, ℓ), w = 2, s = 0, t = 1 >

~~τ~~

Conf:

< (idle, ℓ), water_enabled = 0, s = 0, t = 0 > --~~WATER~~--> < (water_selected, ℓ), w = 1, s = 1, t = 0 >

τ

< (idle, ℓ), w = 1, s = 1, t = 0 > --WATER-->

< (request_sent, ℓ), w = 1, s = 1, t = 0 >

Transition Sequences

(starting at / from $\langle \vec{\ell}_0, \nu_0 \rangle$)

- A **transition sequence** of $\mathcal{C}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ 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

  - $\langle \vec{\ell}_0, \nu_0 \rangle = C_{ini}$, (without "start / from")
  - for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ in $\mathcal{T}(\mathcal{C}(\mathcal{A}_1, \ldots, \mathcal{A}_n))$ with $\langle \vec{\ell}_i, \nu_i \rangle \xrightarrow{\lambda_{i+1}} \langle \vec{\ell}_{i+1}, \nu_{i+1} \rangle$.

Deadlock

- A **configuration** $\langle \ell, \nu \rangle$ of $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ is called **deadlock** if and only if there are no transitions from $\langle \ell, \nu \rangle$, i.e. if

  labels

  $$\neg(\exists\, \lambda \in \Lambda\; \exists\, \langle \ell', \nu' \rangle \in \mathit{Conf} \bullet \langle \ell, \nu \rangle \xrightarrow{\lambda} \langle \ell', \nu' \rangle).$$

  The **network** $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ is said to **have a deadlock** if and only if there is a reachable configuration $\langle \ell, \nu \rangle$ which is a deadlock.

Reachability

- A **configuration** $\langle \vec{\ell}, \nu \rangle$ is called **reachable** (in $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$) **from** $\langle \vec{\ell}_0, \nu_0 \rangle$ if and only if there is a transition sequence of the form

$$\langle \vec{\ell}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{\ell}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{\ell}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \dots \xrightarrow{\lambda_n} \langle \vec{\ell}_n, \nu_n \rangle = \langle \vec{\ell}, \nu \rangle.$$

- A **configuration** $\langle \vec{\ell}, \nu \rangle$ is called **reachable** (without "from"!) if and only if it is reachable from C_{ini}.

- A **location** $\ell \in L_i$ is called **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 ν such that $\ell_i = \ell$ and $\langle \vec{\ell}, \nu \rangle$ is reachable.

Uppaal

(Larsen et al., 1997; Behrmann et al., 2004)

Tool Demo

The Uppaal Query Language

Consider $\mathcal{N} = \mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ over data variables V.

- **basic formula**:

$$atom ::= \mathcal{A}_i.\ell \mid \varphi \mid \texttt{deadlock}$$

  where $\ell \in L_i$ is a location and φ an expression over V.

- **configuration formulae**:

$$term ::= atom \mid \texttt{not}\ term \mid term_1\ \texttt{and}\ term_2$$

- **existential path formulae**:

$$\begin{aligned} e\text{-}formula ::= &\ \exists\Diamond\ term && \text{(exists finally)} \\ \mid &\ \exists\Box\ term && \text{(exists globally)} \end{aligned}$$

- **universal path formulae**:

$$\begin{aligned} a\text{-}formula ::= &\ \forall\Diamond\ term && \text{(always finally)} \\ \mid &\ \forall\Box\ term && \text{(always globally)} \\ \mid &\ term_1\ \texttt{-->}\ term_2 && \text{(leads to)} \end{aligned}$$

- **formulae** (or **queries**):

$$F ::= e\text{-}formula \mid a\text{-}formula$$

Satisfaction of Uppaal Queries by Configurations

- The **satisfaction relation**

$$\langle \vec{\ell}, \nu \rangle \models F$$

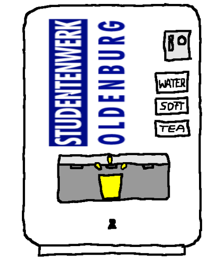
between **configurations**

$$\langle \vec{\ell}, \nu \rangle = \langle (\ell_1, \ldots, \ell_n), \nu \rangle$$

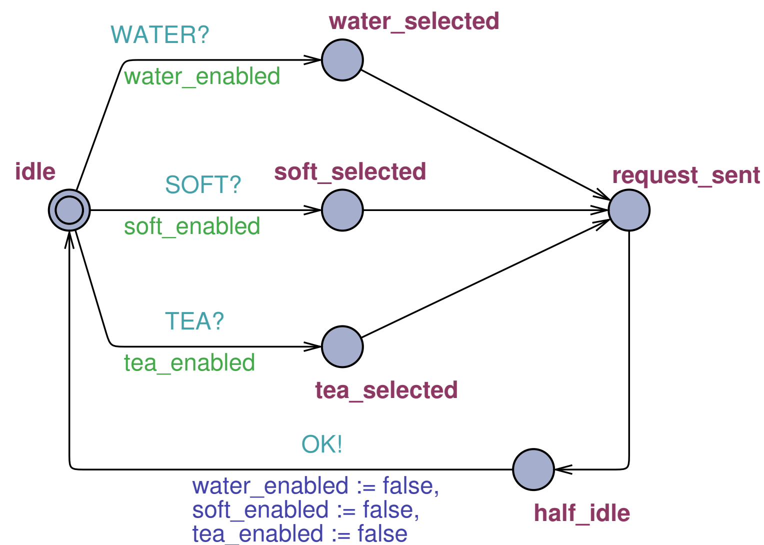
of a network $\mathcal{C}(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ and **formulae** F of the Uppaal logic is defined **inductively** as follows:

- $\langle \vec{\ell}, \nu \rangle \models$ `deadlock` iff $\langle \vec{\ell}, \nu \rangle$ is a deadlock configuration
- $\langle \vec{\ell}, \nu \rangle \models \mathcal{A}_i.\ell$ iff $\vec{\ell}_i = \ell$
- $\langle \vec{\ell}, \nu \rangle \models \varphi$ iff $\nu \models \varphi$
- $\langle \vec{\ell}, \nu \rangle \models$ `not` $term$ iff $\langle \vec{\ell}, \nu \rangle \not\models term$
- $\langle \vec{\ell}, \nu \rangle \models term_1$ `and` $term_2$ iff $\langle \vec{\ell}, \nu \rangle \models term_i$, $i = 1, 2$

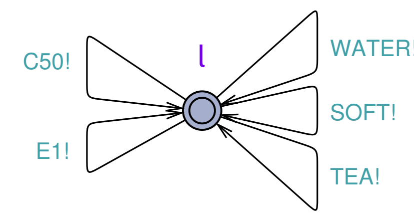
Example: Computation Paths vs. Computation Tree



ChoicePanel:



User:

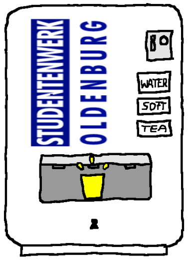


$$\left\langle (\mathsf{idle}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle$$

WATER: $\left\langle (\mathsf{water\_selected}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \xrightarrow{\tau} \left\langle (\mathsf{request\_sent}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \xrightarrow{\tau} \left\langle (\mathsf{half\_idle}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle$

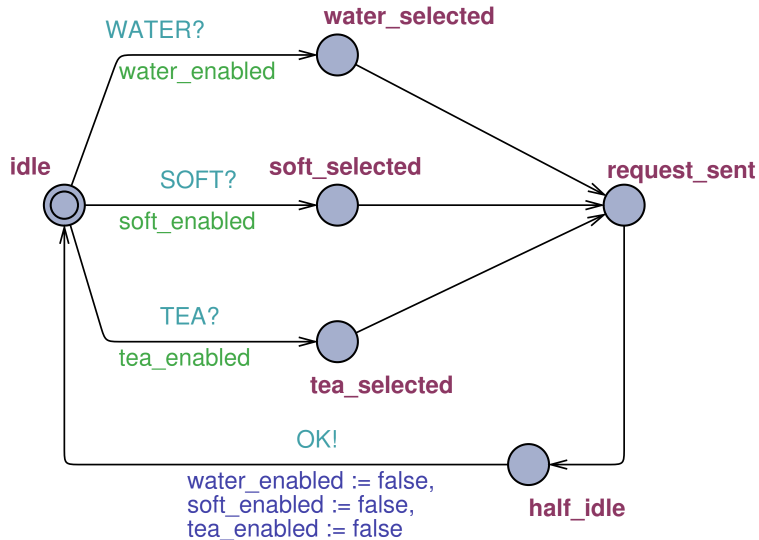
SOFT: $\left\langle (\mathsf{soft\_selected}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \xrightarrow{\tau} \left\langle (\mathsf{request\_sent}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \xrightarrow{\tau} \left\langle (\mathsf{half\_idle}, \mathsf{l}), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle$

Example: Computation Paths vs. Computation Graph

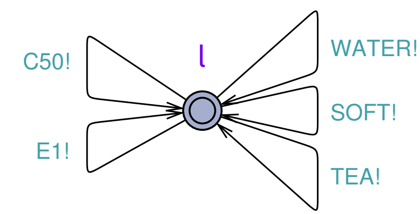


(or: Transition Graph)

ChoicePanel:



User:



$$
\begin{tikzcd}[column sep=small]
& \left\langle \left(\mathsf{idle}, \mathfrak{l}\right), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \arrow[dl, "\mathsf{WATER}"'] \arrow[dr, "\mathsf{SOFT}"] & \\
\left\langle \left(\mathsf{water\_selected}, \mathfrak{l}\right), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \arrow[dr, "\tau"'] & & \left\langle \left(\mathsf{soft\_selected}, \mathfrak{l}\right), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \arrow[dl, "\tau"] \\
& \left\langle \left(\mathsf{request\_sent}, \mathfrak{l}\right), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle \arrow[d, "\tau"'] & \\
& \left\langle \left(\mathsf{half\_idle}, \mathfrak{l}\right), \begin{smallmatrix} we=1 \\ se=1 \\ te=0 \end{smallmatrix} \right\rangle &
\end{tikzcd}
$$

Satisfaction of Uppaal Queries by Configurations

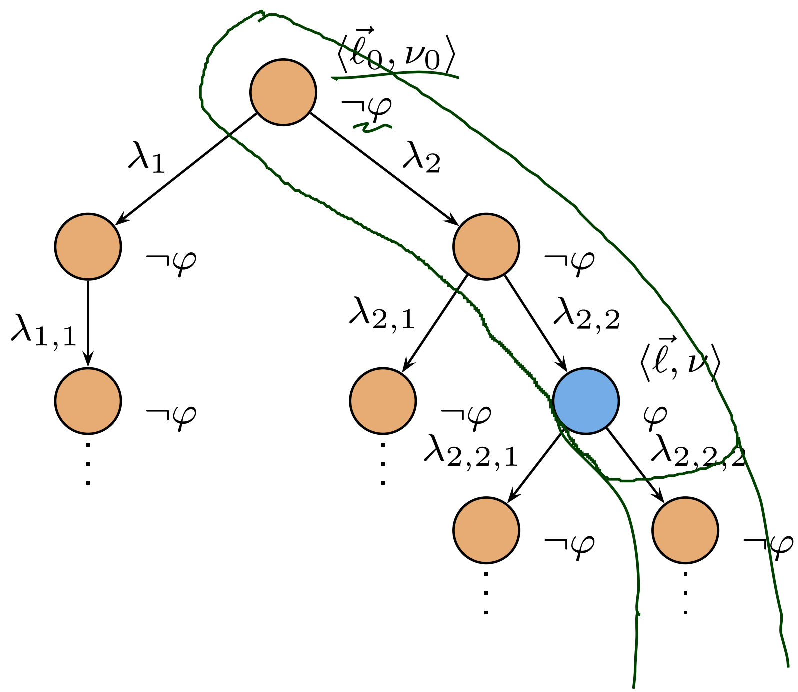
Exists finally:

i-th config. in ξ

- $\langle \vec{\ell}_0, \nu_0 \rangle \models \exists\Diamond\, term$ iff $\exists$ path ξ of $\mathcal{N}$ starting in $\langle \vec{\ell}_0, \nu_0 \rangle$
  $\exists\, i \in \mathbb{N}_0 \bullet \xi^i \models term$

"some configuration satisfying $term$ is reachable"

Example: $\langle \vec{\ell}_0, \nu_0 \rangle \models \exists\Diamond\, \varphi$



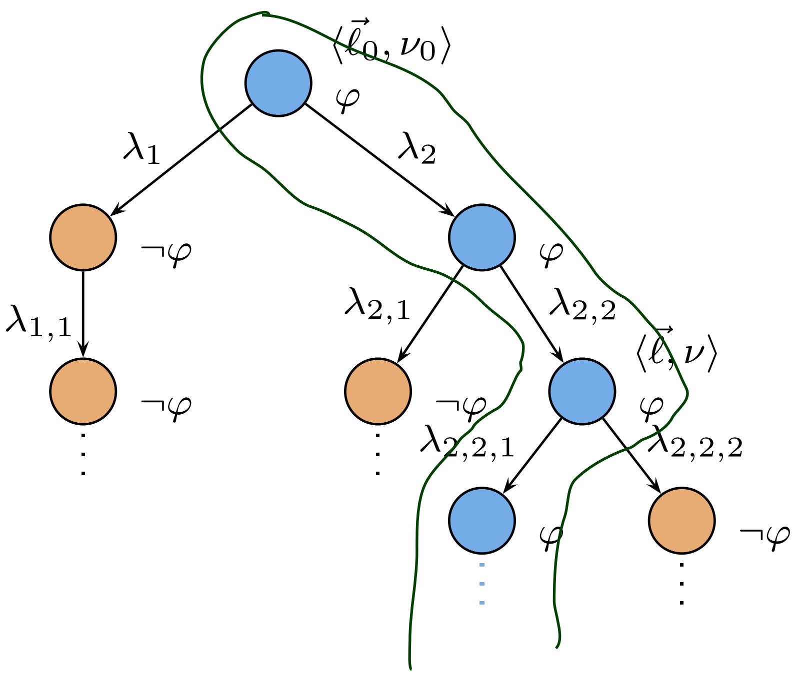
Satisfaction of Uppaal Queries by Configurations

Exists globally:

- $\langle \vec{\ell}_0, \nu_0 \rangle \models \exists\Box\, term$ iff $\exists$ path ξ of $\mathcal{N}$ starting in $\langle \vec{\ell}_0, \nu_0 \rangle$
  $\forall\, i \in \mathbb{N}_0 \bullet \xi^i \models term$

"on some computation path, all configurations satisfy $term$"

Example: $\langle \vec{\ell}_0, \nu_0 \rangle \models \exists\Box\, \varphi$



Satisfaction of Uppaal Queries by Configurations

- **Always globally**:
  - $\langle \vec{\ell}_0, \nu_0 \rangle \models \forall\Box\, term$ iff $\langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists\Diamond\, \neg term$

    "not (some configuration satisfying $\neg term$ is reachable)"

    or: "all reachable configurations satisfy $term$"

- **Always finally**:
  - $\langle \vec{\ell}_0, \nu_0 \rangle \models \forall\Diamond\, term$ iff $\langle \vec{\ell}_0, \nu_0 \rangle \not\models \exists\Box\, \neg term$

    "not (on some computation path, all configurations satisfy $\neg term$)"

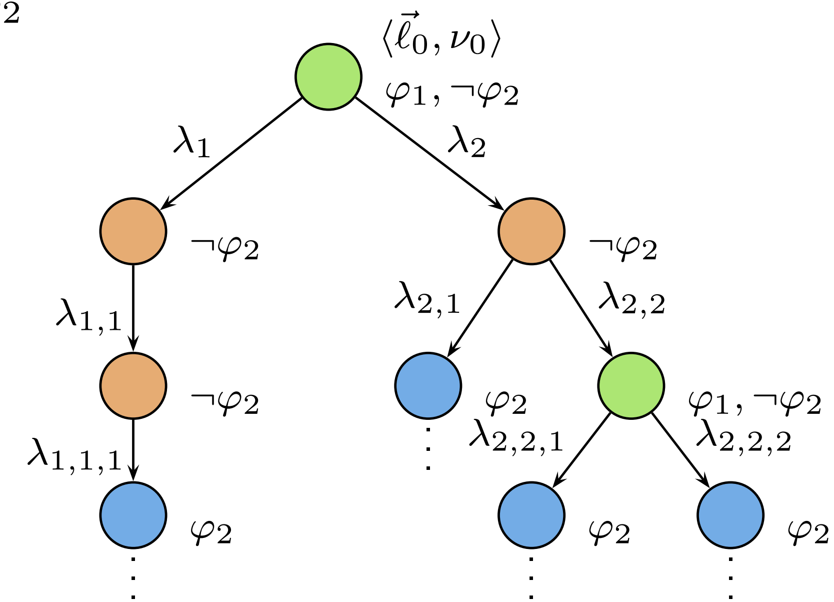
    or: "on all computation paths, there is a configuration satisfying $term$"

Leads to:

- $\langle \vec{\ell}_0, \nu_0 \rangle \models term_1 \longrightarrow term_2$ iff $\forall$ path ξ of $\mathcal{N}$ starting in $\langle \vec{\ell}_0, \nu_0 \rangle$ $\forall\, i \in \mathbb{N}_0 \bullet$
  $\xi^i \models term_1 \implies \xi^i \models \forall\Diamond\, term_2$

"on all paths, from each configuration satisfying $term_1$,
a configuration satifying $term_2$ is reachable" (**response pattern**)

Example: $\langle \vec{\ell}_0, \nu_0 \rangle \models \varphi_1 \longrightarrow \varphi_2$



CFA Model-Checking

Definition. Let $\mathcal{N} = \mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ 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 $(\mathcal{N}, F) \in \models$.

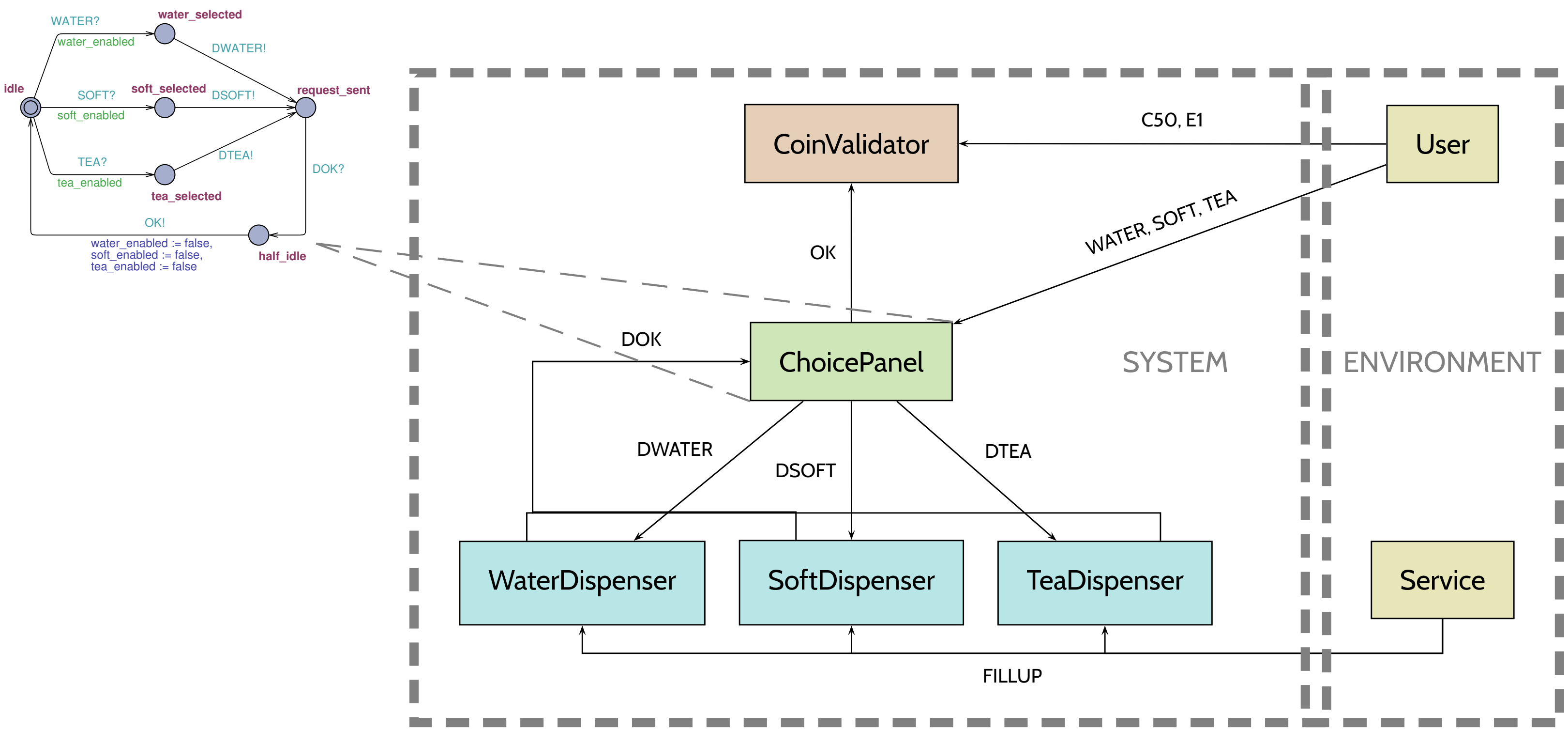
Proposition.
The model-checking problem for communicating finite automata is **decidable**.

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).
- **Uppaal Architecture**

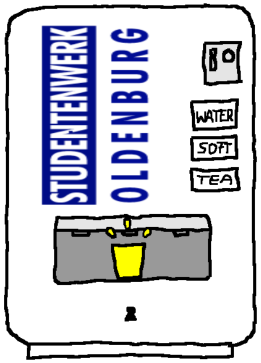
CFA and Queries at Work

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 is (at all) possible to have no water in the vending machine model?
  (Otherwise, the design is definitely broken.)

- **Approach**: Check whether a configuration satisfying

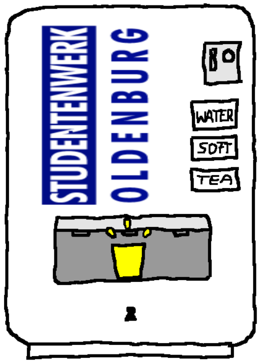
  $$w = 0$$

  is reachable, i.e. check

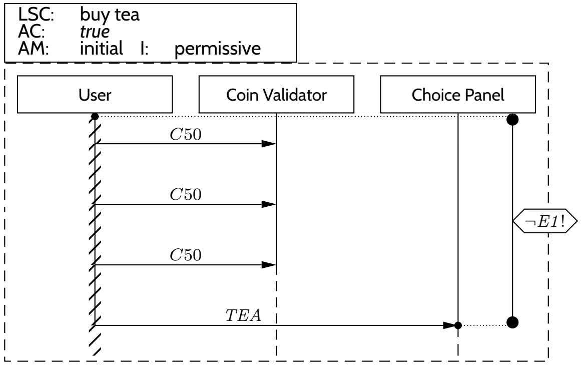
  $$\mathcal{N}_{\mathrm{VM}} \models \exists\Diamond\, w = 0.$$

  for the vending machine model $\mathcal{N}_{\mathrm{VM}}$.

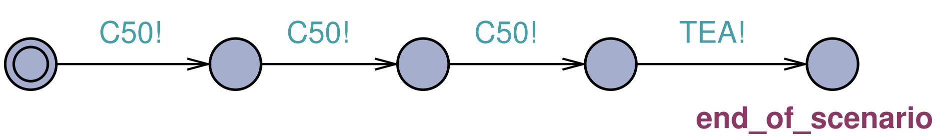
Design Check: Scenarios



- **Question**: Is the following existential LSC satisfied by the model?
  (Otherwise, the design is definitely broken.)



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



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

$$\mathcal{N}'_{\mathrm{VM}} \models \exists\Diamond\,\text{Scenario}.\texttt{end\_of\_scenario}.$$

for the modified vending machine model $\mathcal{N}'_{\mathrm{VM}}$.

Design Verification: Invariants

- **Question**: Is it the case that the "tea" button is **only** enabled if there is € 1.50 in the machine?
  (Otherwise, the design is broken.)

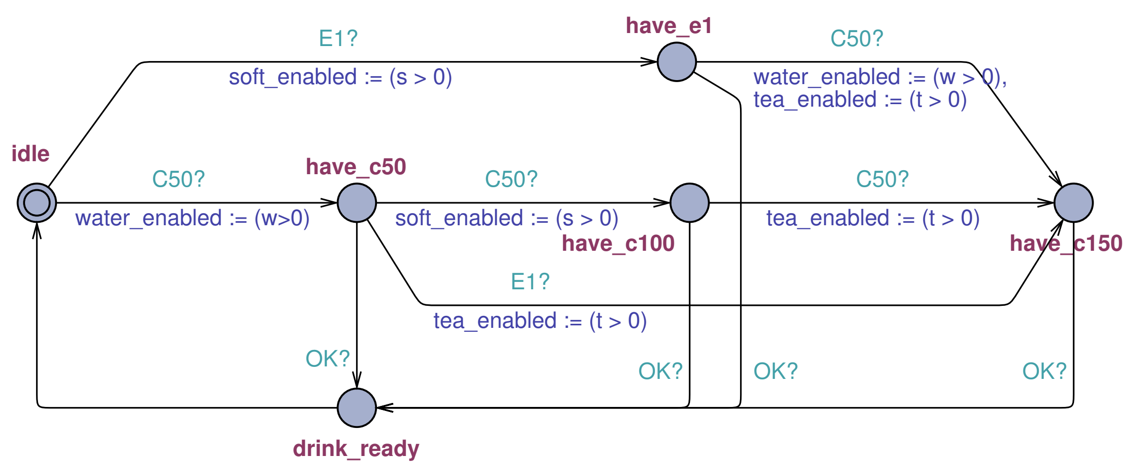
- **Approach**: Check whether the implication

$$\texttt{tea\_enabled} \implies \text{CoinValidator}.\texttt{have\_c150}$$

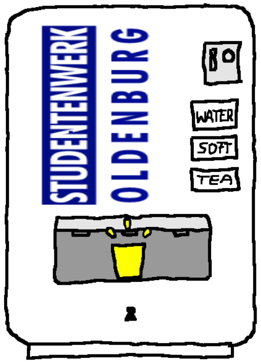
holds in all reachable configurations, i.e. check

$$\mathcal{N}_{\mathrm{VM}} \models \forall\Box\, \texttt{tea\_enabled} \quad \texttt{imply} \quad \text{CoinValidator}.\texttt{have\_c150}$$

for the vending machine model $\mathcal{N}_{\mathrm{VM}}$.



Design Verification: Sanity Check



- **Question**: Is the "tea" button **ever** enabled?
  (Otherwise, the considered invariant

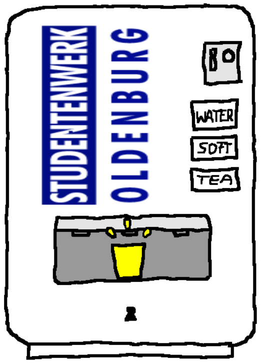
  $$\texttt{tea\_enabled} \implies \text{CoinValidator}.\texttt{have\_c150}$$

  holds vacuously.)

- **Approach**: Check whether a configuration satisfying $\texttt{water\_enabled} = 1$ is reachable.

  Exactly like we did with $w = 0$ earlier.

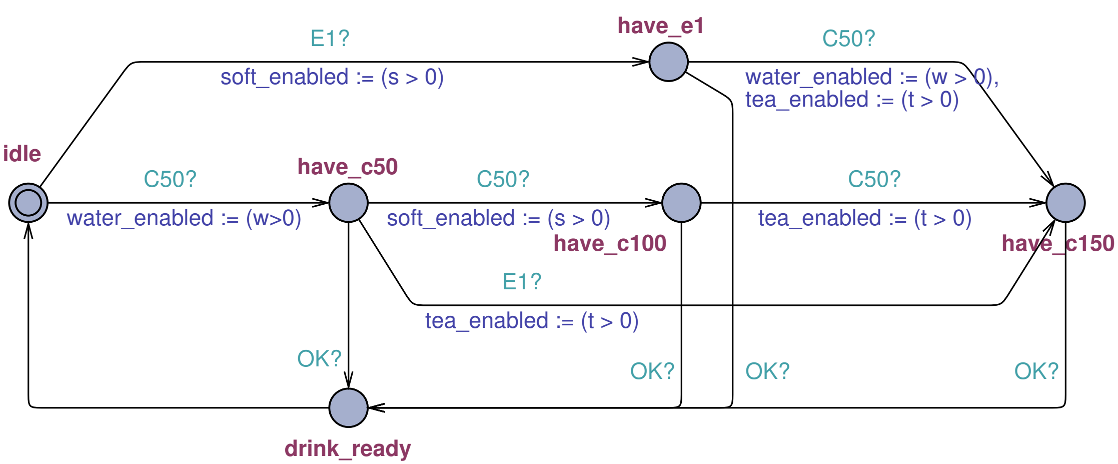
Design Verification: Another Invariant



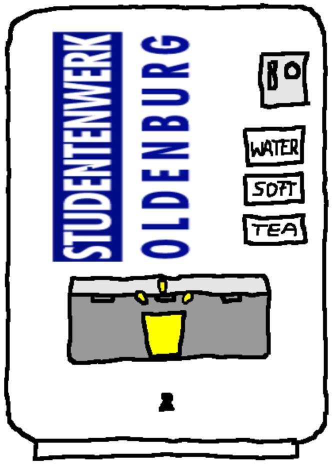
- **Question**: Is it the case that, if there is money in the machine and water in stock, that the “water” button is enabled?

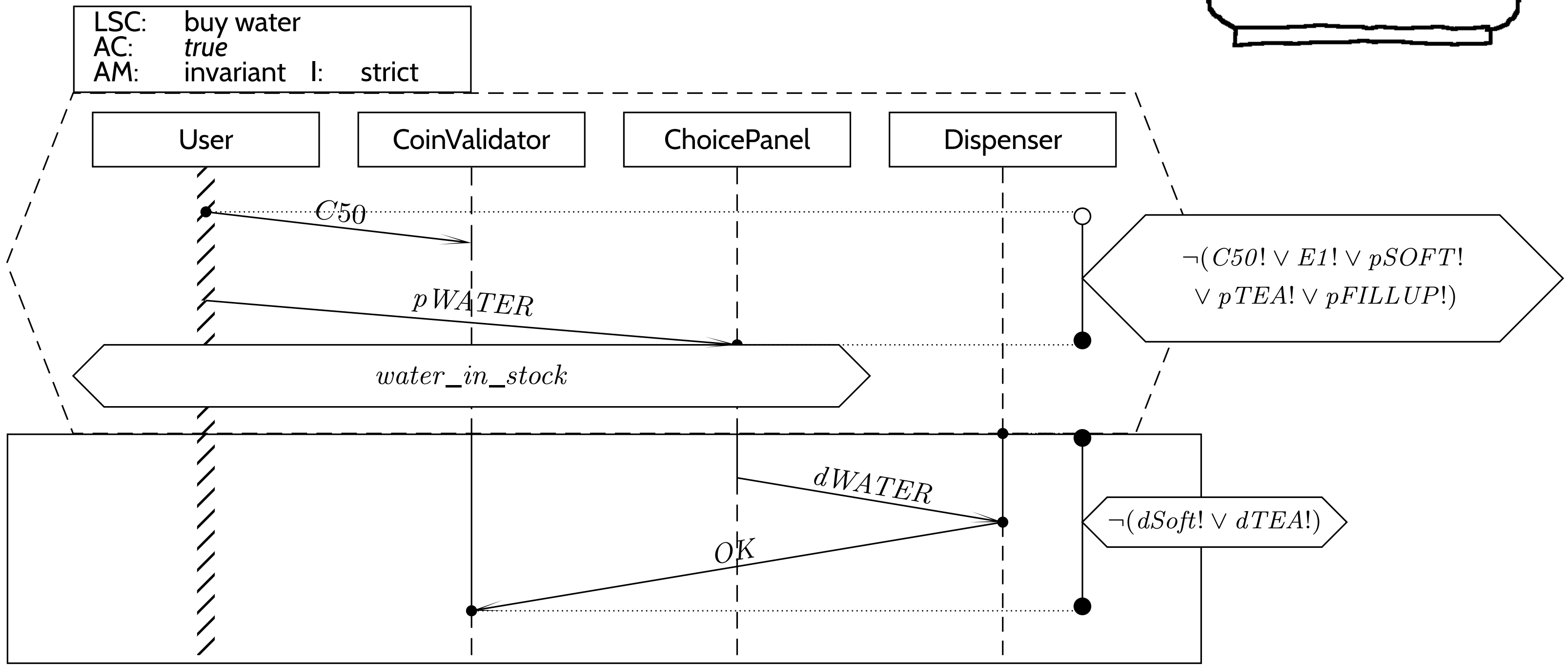
- **Approach**: Check

$$\mathcal{N}_{\mathrm{VM}} \models \forall\Box\, (\text{CoinValidator}.\texttt{have\_c50}\ \texttt{or}\ \text{CoinValidator}.\texttt{have\_c100}\ \texttt{or}\ \text{CoinValidator}.\texttt{have\_c150})\ \texttt{imply water\_enabled}.$$



Recall: Universal LSC Example



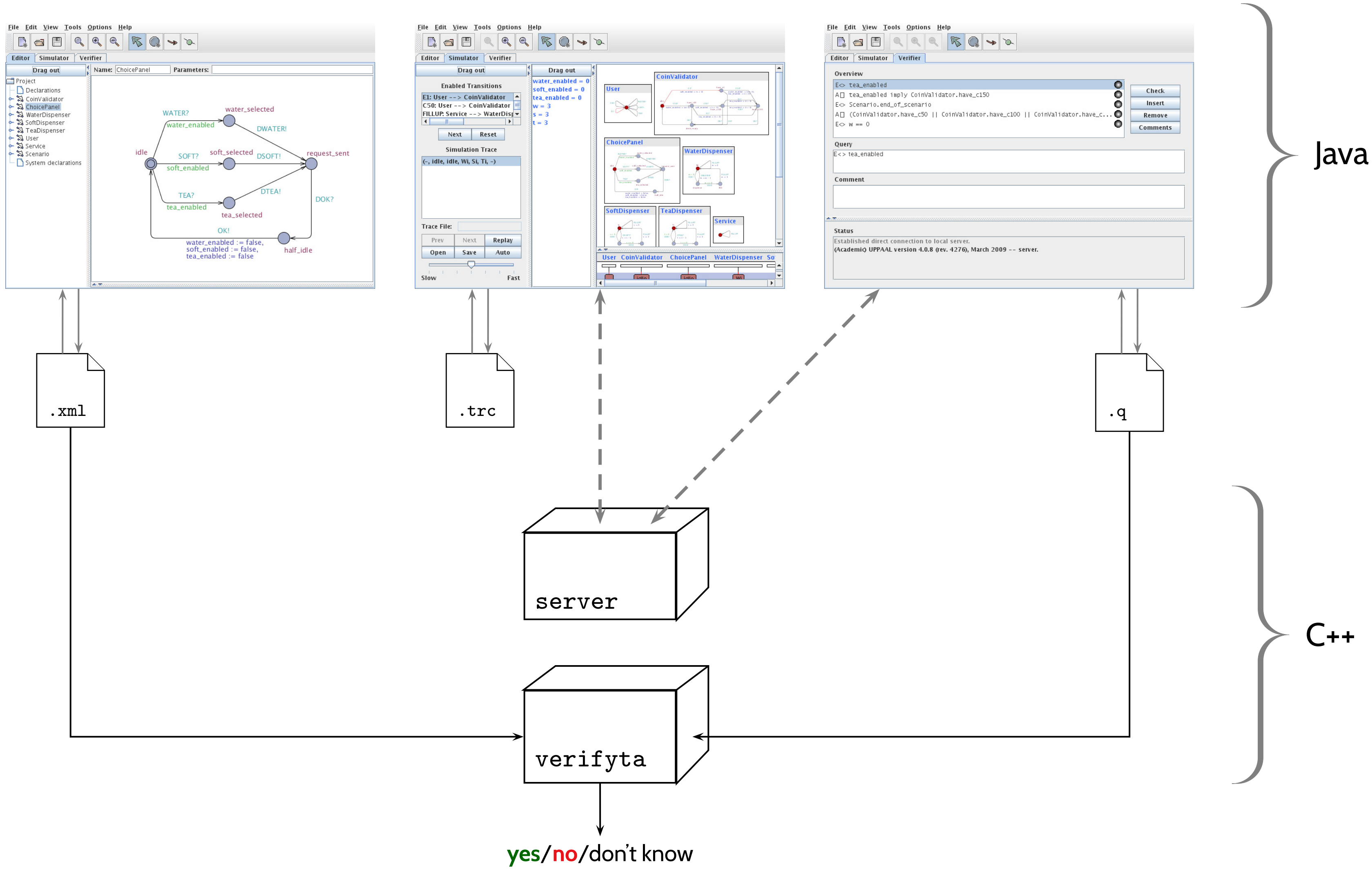


Content

- **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

Uppaal Architecture



Tell Them What You've Told Them...

- 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** ($\exists\Diamond\ CF$, $\forall\Box\ CF$, $\ldots$) and
  - **leadsto** ($CF_1 \longrightarrow CF_2$) 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 satified.

    (If not, analyse the design further using the obtained **counter-example**).

References

References

Behrmann, G., David, A., and Larsen, K. G. (2004). A tutorial on uppaal 2004-11-17. Technical report, Aalborg University, Denmark.

Larsen, K. G., Pettersson, P., and Yi, W. (1997). Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Olderog, E.-R. and Dierks, H. (2008). *Real-Time Systems - Formal Specification and Automatic Verification*. Cambridge University Press.