NEXT TUTORIAL: MONDAY

Softwaretechnik / Software-Engineering

Lecture 13: Behavioural Software Modelling

2015-06-29

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

Albert-Ludwigs-Universität Freiburg, Germany

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**

Introduction	L 1:	20.4., Mo
	T 1:	23.4., Do
Development Process, Metrics	L 2:	27.4., Mo
	L 3:	30.4., Do
	L 4:	4.5., Mo
	T 2:	7.5., Do
	L 5:	11.5., Mo
Requirements Engineering	-	14.5., Do
	L 6:	18.5., Mo
	L 7:	21.5., Do
	-	25.5., Mo
	-	28.5., Do
	T 3:	1.6., Mo
	-	4.6., Do
	L 8:	8.6., Mo
	L 9:	11.6., Do
	L 10:	15.6., Mo
	T 4:	18.6., Do
Architecture & Design, Software Modelling	L 11:	22.6., Mo
	L 12:	25.6., Do
	L 13:	29.6., Mo
	L 14:	2.7., Do
	T 5:	6.7., Mo
Quality Assurance	L 15:	9.7., Do
	L 16:	13.7., Mo
Invited Talks	L 17:	16.7., Do
	T 6:	20.7., Mo
Wrap-Up	L 18:	23.7., Do

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 $(a, b \in)$ 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.

  By $(\varphi \in)\ \Psi(V)$ we denote the set of **integer expressions** over V using function symbols $+, -, \dots$, $>$, $\neq$, ...

- A **modification** on v is

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

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

- By $\vec{r}$ we denote a finite list $\langle r_1, \dots, r_n \rangle$, $n \in \mathbb{N}_0$, 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.

Communicating Finite Automata

plural

singular

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 set of **directed edges** such that

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

boolean

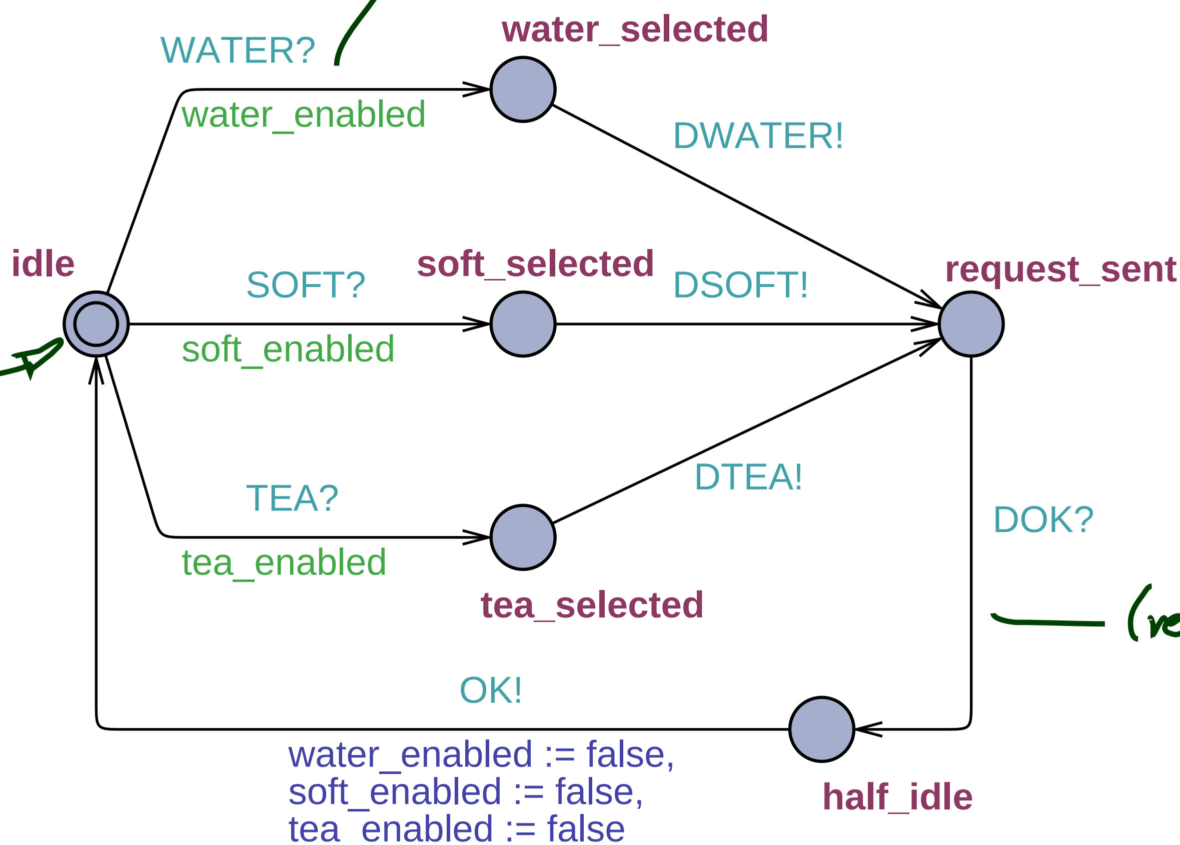
Edges $(\ell, \alpha, \varphi, \vec{r}, \ell')$ from location ℓ to ℓ' are labelled with an **action** α, a **guard** φ, and a list $\vec{r}$ of **modifications**.

- ℓ_{ini} is the **initial location**.

Example

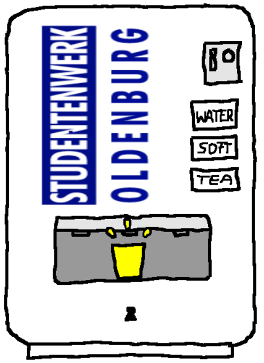
(idle, WATER?, water_enabled, <>, water-selected)

ChoicePanel:



initial location

(req_sent, DOK?, true, <>, half_idle)



$L = \{ \text{idle}, \text{w\_selected}, \ldots \}$

Operational Semantics of Networks of FCA

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 FCA $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ is the labelled transition system

configurations labels labelled transition relations

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

initial configuration

where

- $V = \bigcup_{i=1}^{n} V_i$,
- $\mathit{Conf} = \{\langle \overbrace{\vec{\ell}}^{(\ell_1, \ell_2, \dots, \ell_n)}, \nu \rangle \mid \ell_i \in L_i, \overbrace{\nu : V \to \mathscr{D}(V)}^{\text{a valuation of the variables in } V}\}$,
- $C_{ini} = \langle \vec{\ell}_{ini}, \nu_{ini} \rangle$ 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 $\varphi \in \Phi(V)$.
- $\models \subseteq (V \to \mathscr{D}(V)) \times \Phi(V)$ is the canonical satisfaction relation between valuations and integer expressions from $\Phi(V)$.

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

  $\nu(\varphi) = 13$

- $\nu \models x \neq 0$

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

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

- **Effect of modification** $r \in R(V)$ **on** ν, denoted by $\nu[r]$:

$$\nu[v := \varphi](a) := \begin{cases} \nu(\varphi), \text{ if } a = v, & ① \\ \nu(a), \text{ otherwise} & ② \end{cases}$$

$\nu[x := 0](x) = 0$ ①

$\nu[x := y - x](x) = 7$ ①

$\nu[x := 0](y) = 10$ ②

- We set $\nu[\langle r_1, \dots, r_n \rangle] := \nu[r_1] \dots [r_n] = (((\nu[r_1])[r_2]) \dots)[r_n]$.

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

$\nu[x := 0] = \{x \mapsto 0, y \mapsto 10\}$

$\nu[\langle x := 31, y := x, x := 7 \rangle] = \{x \mapsto 7, y \mapsto 31\}$

Operational Semantics of Networks of FCA

$(\ell_1, \dots, \ell_i, \dots, \ell_n)$

- 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
    - $\nu \models \varphi$, "source valuation satisfies guard"
    - $\vec{\ell'} = \vec{\ell}[\ell_i := \ell'_i]$, "automaton i changes location"
    - $\nu' = \nu[\vec{r}]$, "ν' is ν modified by $\vec{r}$"

- A **synchronisation transition** $\langle \vec{\ell}, \nu \rangle \xrightarrow{b} \langle \vec{\ell'}, \nu' \rangle$ occurs if there are $i, j \in \{1, \dots, n\}$ 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
    - $\nu \models \varphi_i \wedge \varphi_j$,
    - $\vec{\ell'} = \vec{\ell}[\ell_i := \ell'_i][\ell_j := \ell'_j]$,
    - $\nu' = \nu[\vec{r}_i][\vec{r}_j]$, "output first, then input"

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

Transition Sequences, Reachability

- A **transition sequence** of $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ is any (in)finite sequence of the form

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

  with

  - $\langle \ell_0, \nu_0 \rangle = C_{ini}$,
  - for all $i \in \mathbb{N}$, there is $\xrightarrow{\lambda_{i+1}}$ in $\mathcal{T}(\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n))$ 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 $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$) if and only if there is a transition sequence of the form

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

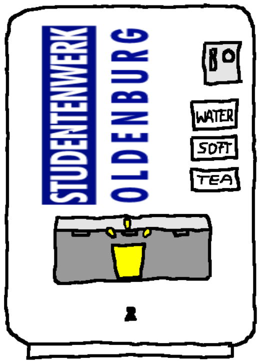
- A **location** ℓ_i is called **reachable** if and only if **any** configuration $\langle \vec{\ell}, \nu \rangle$ is reachable, i.e. there exists a valuation ν such that $\langle \vec{\ell}, \nu \rangle$ is reachable.

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

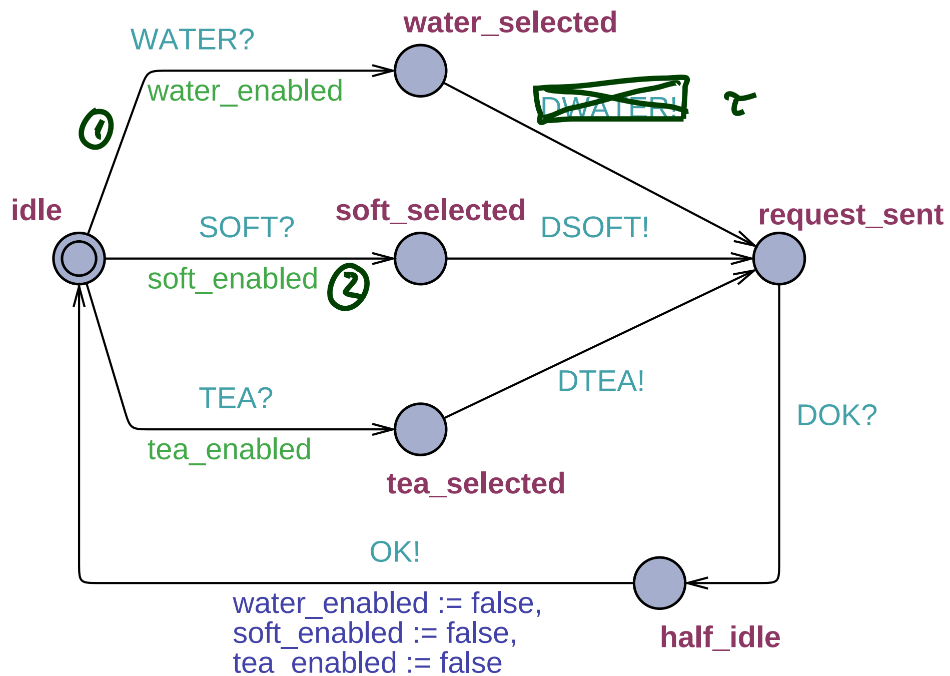
$$\nexists \xrightarrow{\lambda} \in \mathcal{T}(\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)), \langle \ell', \nu' \rangle \in Conf \bullet \langle \ell, \nu \rangle \xrightarrow{\lambda} \langle \ell', \nu' \rangle.$$

NO DEADLOCK

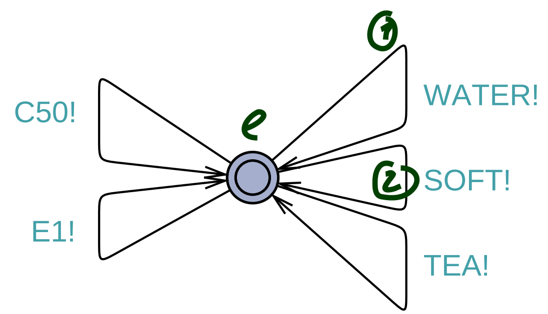
Example



ChoicePanel:



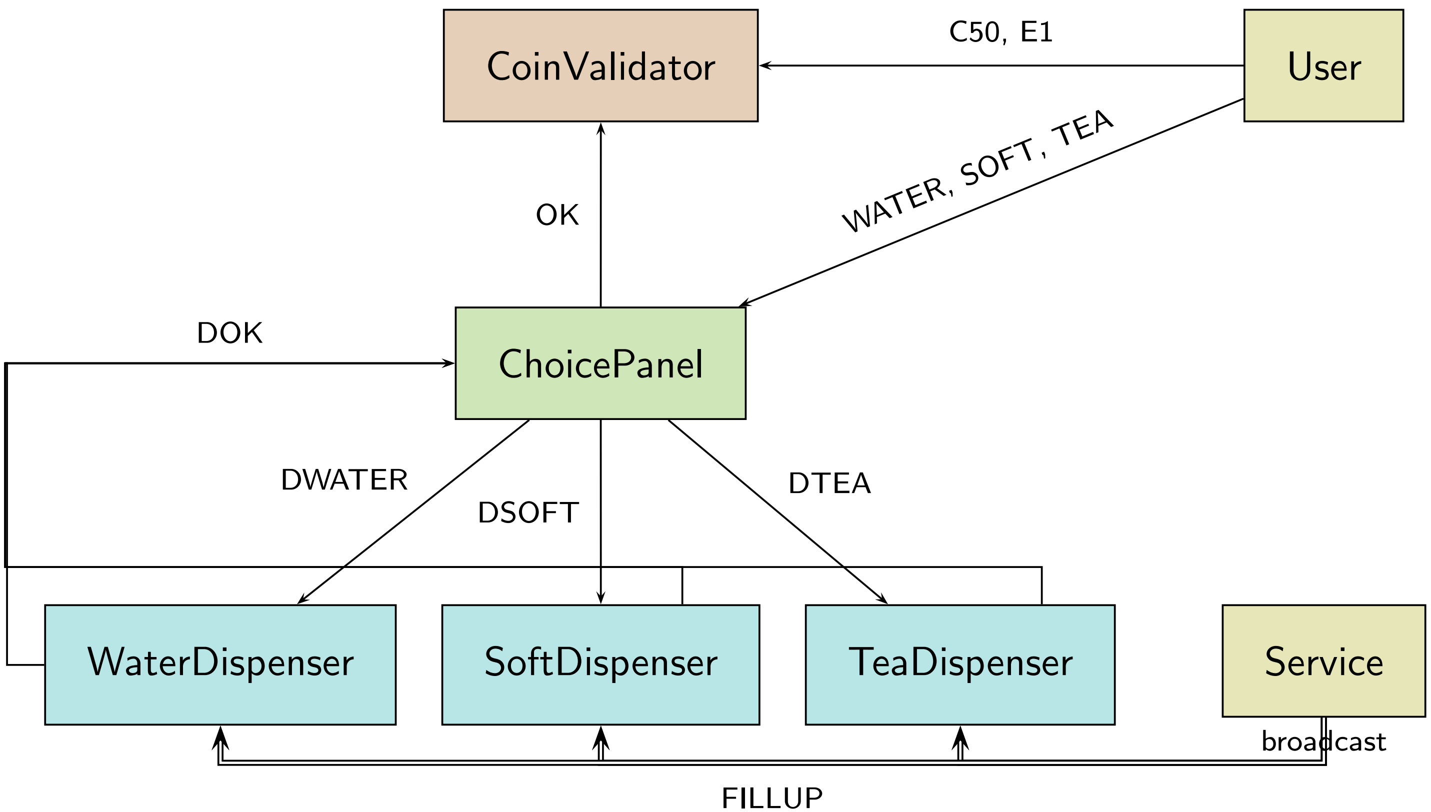
User:



$$\langle (idle, e), \overbrace{w\_en = 1, s\_en = 1, t\_en = 0}^{=:\nu} \rangle \xrightarrow{WATER} \langle (w\_sel, e), \nu \rangle \xrightarrow{\tau} \langle (req\_s, e), \nu \rangle$$

$$\xrightarrow{SOFT} \langle (s\_sel, e), \nu \rangle \longleftarrow \text{deadlock}$$

Model Architecture — Who Talks What to Whom



- **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

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 $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ be a network of CFA.
- $\Sigma = \mathit{Conf}$
- $A = \mathsf{Chan} \cup \{\tau\}$
- $[\![\mathcal{C}]\!] = \{\pi = \langle \vec{\ell}_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle \vec{\ell}_1, \nu_1 \rangle \xrightarrow{\lambda_2} \langle \vec{\ell}_2, \nu_2 \rangle \xrightarrow{\lambda_3} \cdots \mid \pi \text{ is a computation path of } \mathcal{C}\}$.

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

Uppaal

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

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

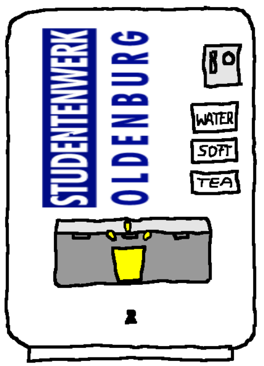
$$(\mathcal{C}, F) \in \models.$$

$\mathcal{C} \models F$

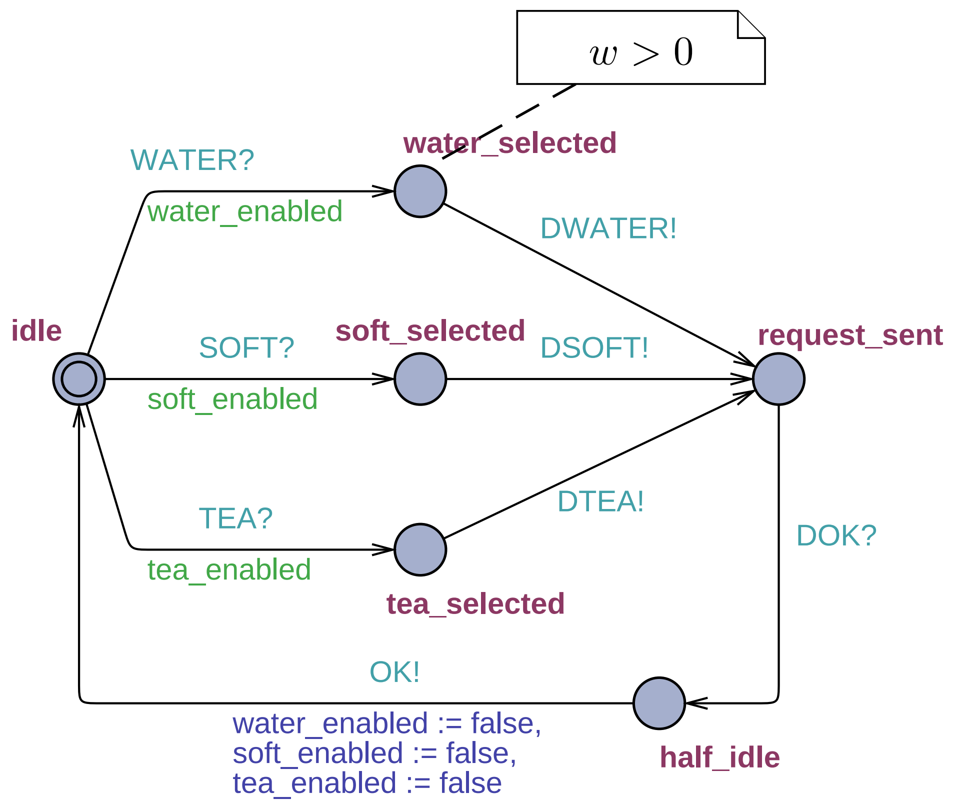
$VM \models E\Diamond\, w=0$

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

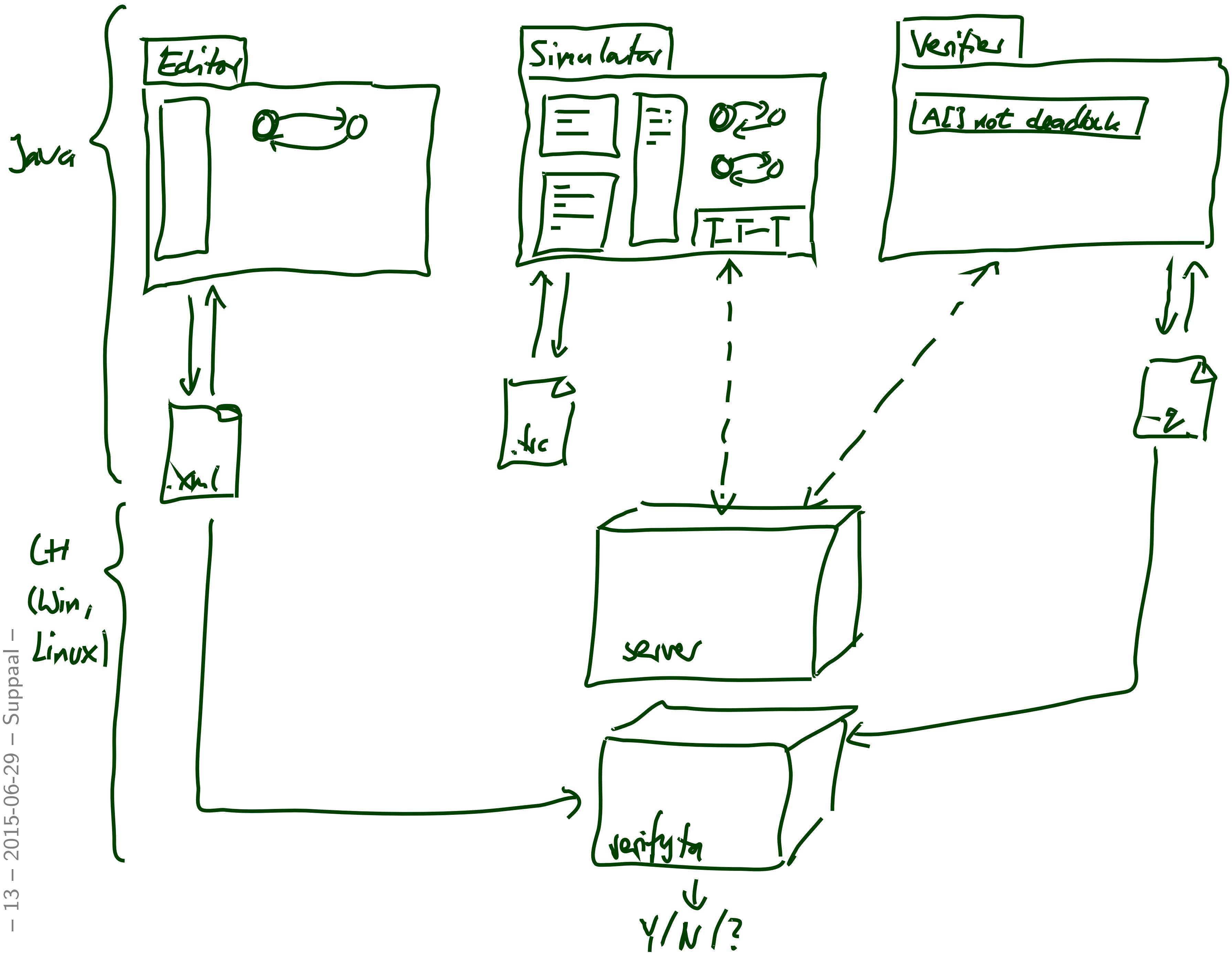
Example: Invariants in the Model



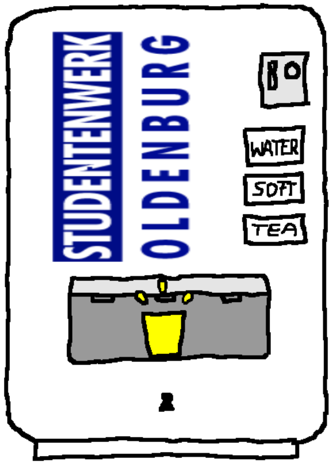
ChoicePanel:

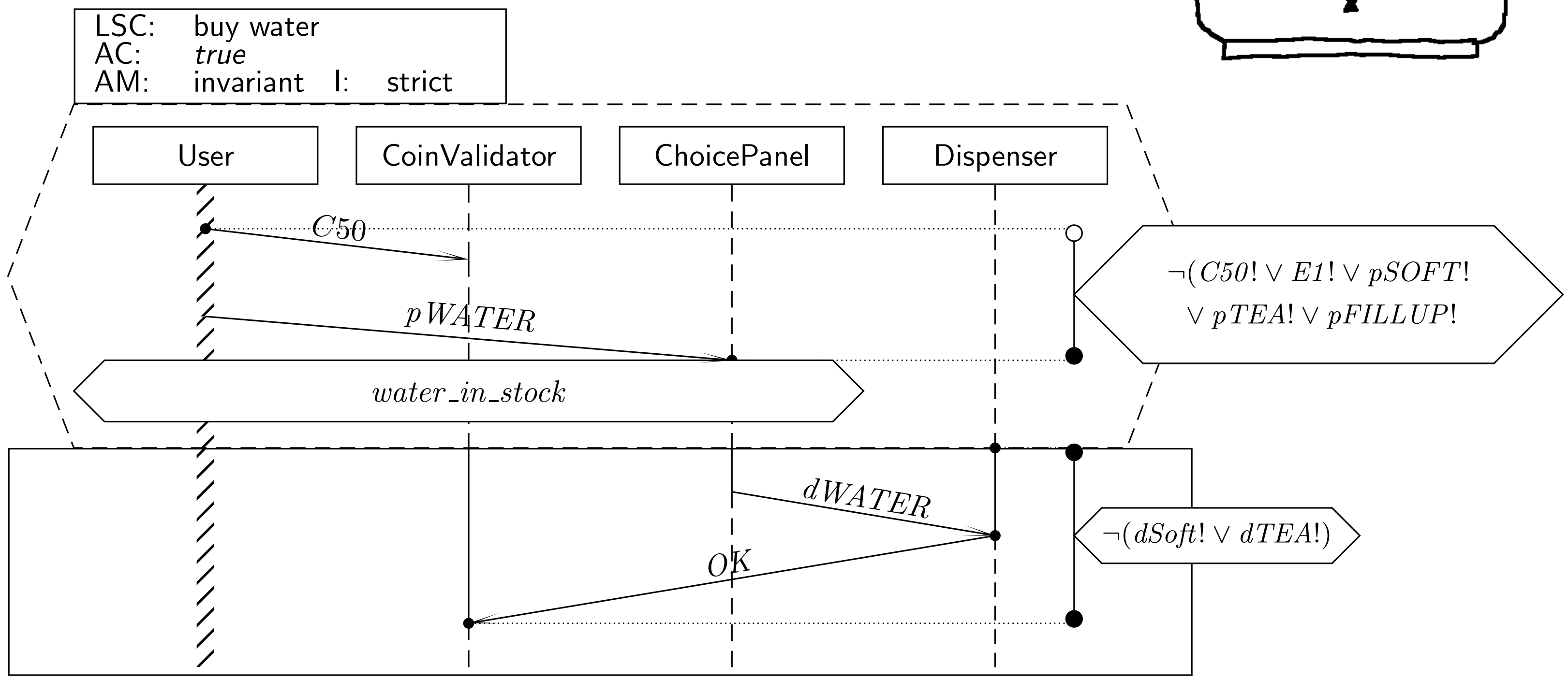


Uppaal Architecture



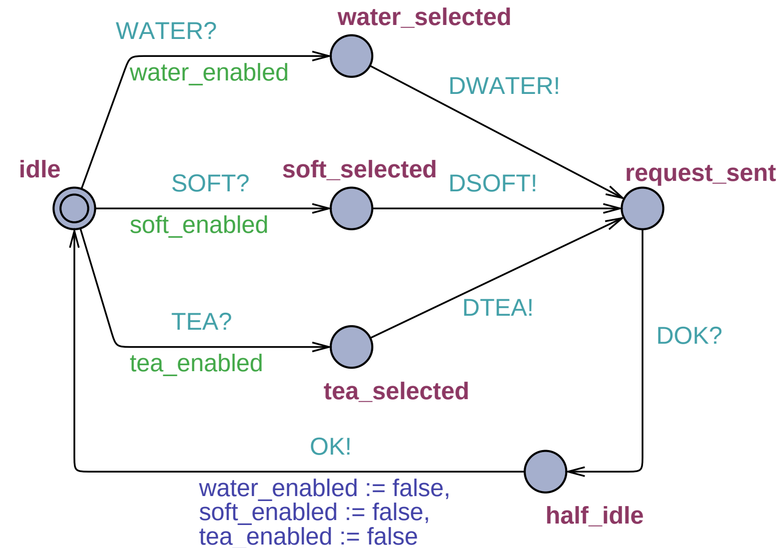
Recall: Universal LSC Example

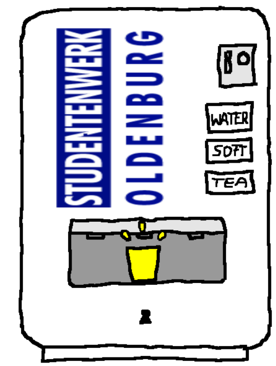




Implementing Communicating Finite Automata

Implementing CFA

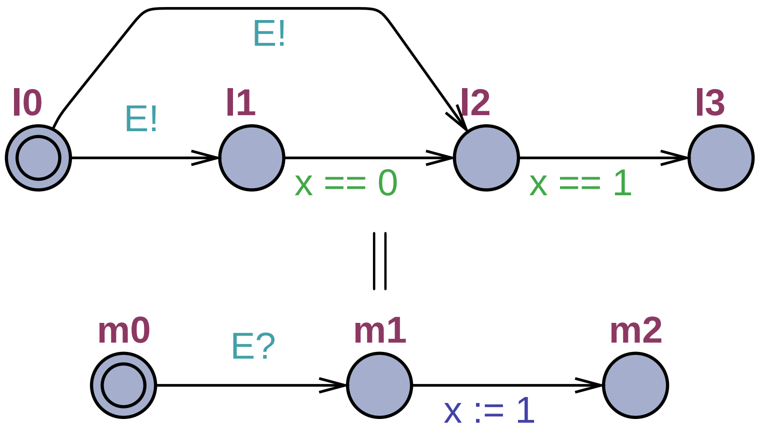


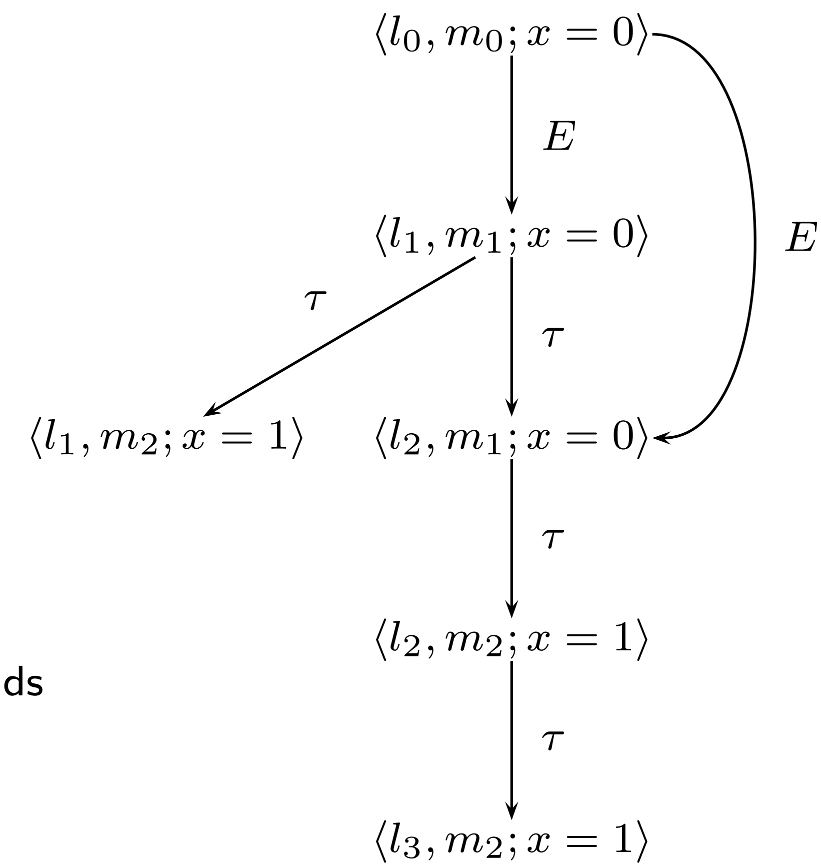



```
st : { idle, wsel, ssel, tsel, reqs, half };

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...



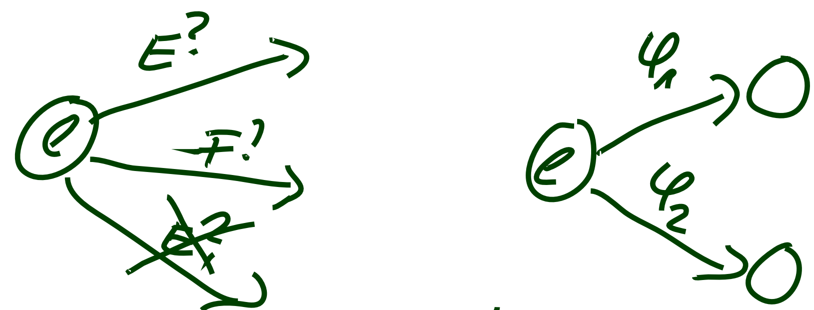


- How are we supposed to implement that?
  - There is **non-determinism** in the upper automaton,
  - internal transitions can **interleave**, one interleaving leads to a deadlock.
- We are not!
- We define
  - **deterministic** CFA,
  - a **greedy** semantics for internal transitions.

and only implement deterministic CFA using the greedy semantics.

Deterministic CFA

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



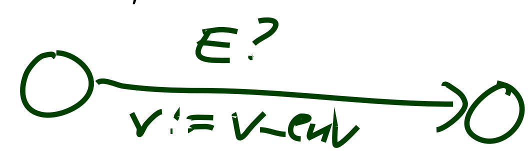
$(\varphi_1 \wedge \varphi_2) \leftrightarrow \mathit{false}$

Deterministic CFA

- The communicating finite automaton $\mathcal{A} = (L, B, V, E, \ell_{ini})$ is called **deterministic** if and only if
  - for each location ℓ,
    - **either** all edges with ℓ as source location have pairwise different **input actions**,
    - **or** there is no edge with an input action starting at ℓ, and all edges starting at ℓ have pairwise (logically) disjoint guards.
- Let each automaton in the network $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ be marked as either **environment** or **controller**.

  We call $\mathcal{C}$ **implementable** if and only if, for each **controller** $\mathcal{A}$ in $\mathcal{C}$,

  (i) $\mathcal{A}$ is deterministic,



  (ii) $\mathcal{A}$ 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) $\mathcal{A}$ 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:

$$(\mathcal{A}.\ell \wedge \varphi) \longrightarrow (\mathcal{A}.\ell')$$

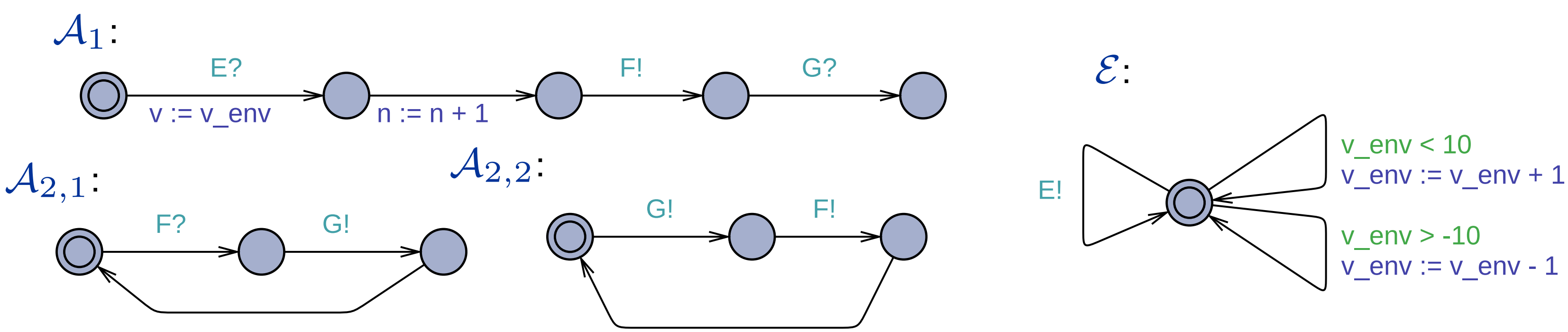
  for each edge $(\ell, \alpha, \varphi, \vec{r}, \ell')$ of $\mathcal{A}$.

Greedy CFA Semantics

- **Greedy** semantics:
  - each input synchronisation transition (plus: system start) of automaton $\mathcal{A}$ is followed by a maximal sequence of internal transitions or output transitions of $\mathcal{A}$.
  - **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:



- $\mathcal{A}_1$ is implementable in $\mathcal{C}(\mathcal{A}_1, \mathcal{A}_{2,1}, \mathcal{E})$ (environment: only $\mathcal{E}$)
  - deterministic: ✔,
  - only local variables, environment variables with input: ✔,
  - locally deadlock-free: ✔.
- $\mathcal{A}_1$ is **not** implementable in $\mathcal{C}(\mathcal{A}_1, \mathcal{A}_{2,2}, \mathcal{E})$.

Model vs. Implementation

- Now an implementable model $\mathcal{C}(\mathcal{A}_1, \dots, \mathcal{A}_n)$ has **two semantics**:
  - $[\![\mathcal{C}]\!]_{std}$ — standard semantics.
  - $[\![\mathcal{C}]\!]_{grd}$ — greedy semantics.
- Are they **related** in any way?

References

References

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

Glinz, M. (2008). Modellierung in der Lehre an Hochschulen: Thesen und Erfahrungen. *Informatik Spektrum*, 31(5):425–434.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

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.

OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.