kbs Knowledge-Based Systems Group
BIG Business Informatics Group
TU WIEN
JKU JOHANNES KEPLER UNIVERSITÄT LINZ
dbai
RiSE
Vienna University of Technology
WWTF WIENER WISSENSCHAFTS-, FORSCHUNGS- UND TECHNOLOGIEFONDS

A SAT-based Debugging Tool for State Machines and Sequence Diagrams

Petra Kaufmann, Martin Kronegger,
Andreas Pfandler, Martina Seidl,
Magdalena Widl

State Machines

Image courtesy of “Piled Higher and Deeper” by Jorge Cham `www.phdcomics.com`

State Machines

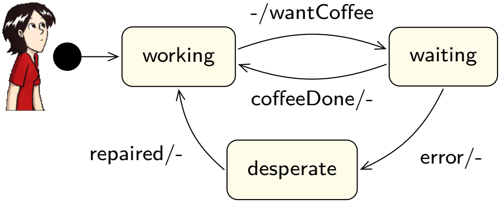


Image courtesy of “Piled Higher and Deeper” by Jorge Cham `www.phdcomics.com`

State Machines

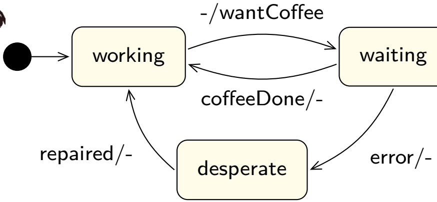
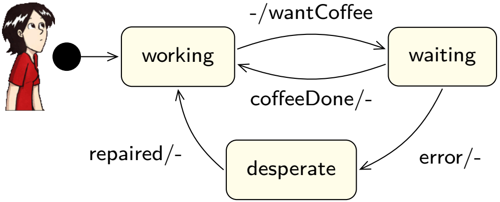


Image courtesy of "Piled Higher and Deeper" by Jorge Cham `www.phdcomics.com`

State Machines



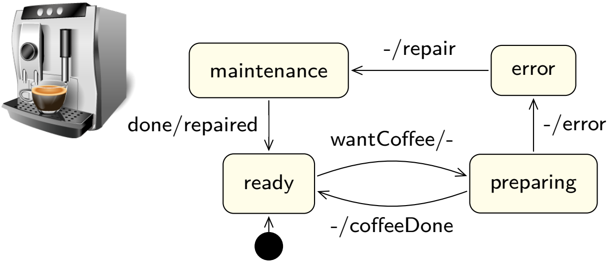


Image courtesy of “Piled Higher and Deeper” by Jorge Cham `www.phdcomics.com`

State Machines

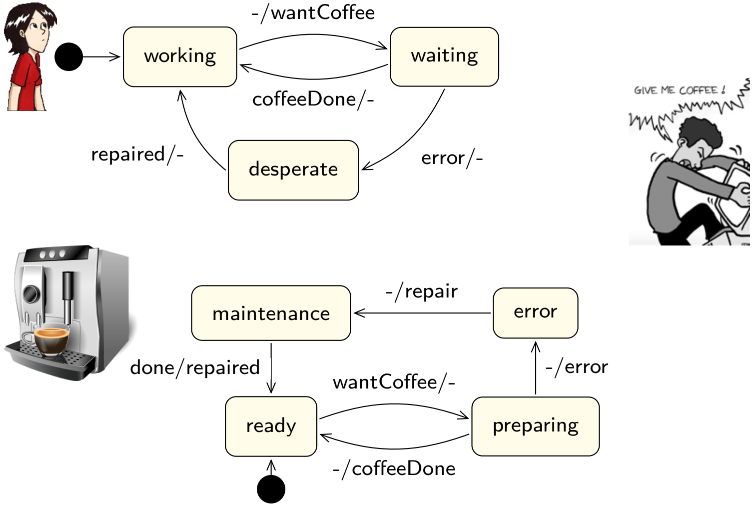


Image courtesy of “Piled Higher and Deeper” by Jorge Cham `www.phdcomics.com`

State Machines

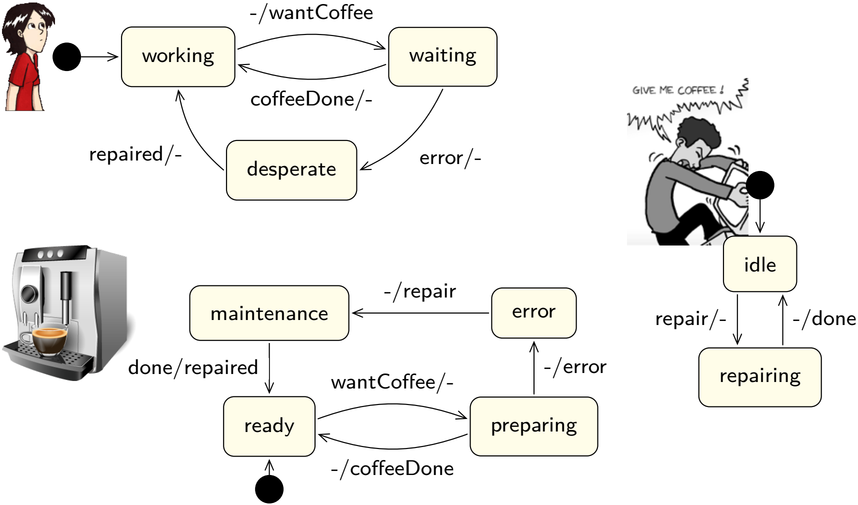


Image courtesy of “Piled Higher and Deeper” by Jorge Cham `www.phdcomics.com`

State Machines

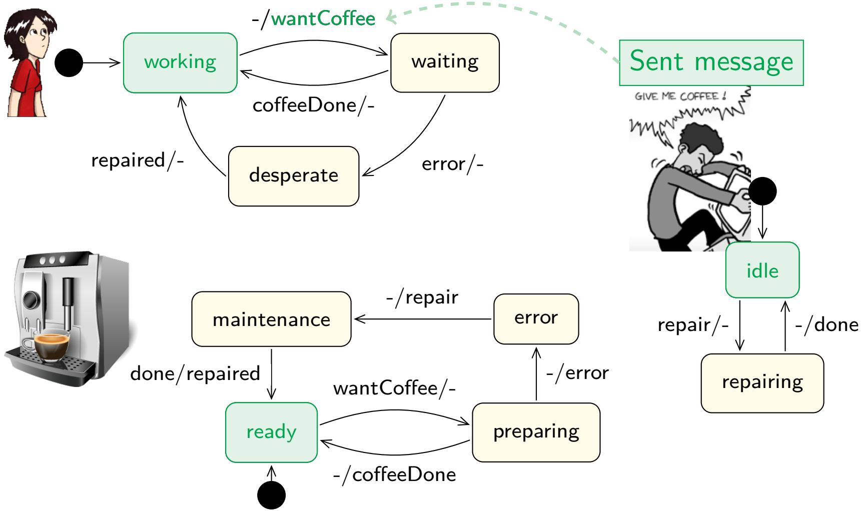


Image courtesy of "Piled Higher and Deeper" by Jorge Cham www.phdcomics.com

State Machines

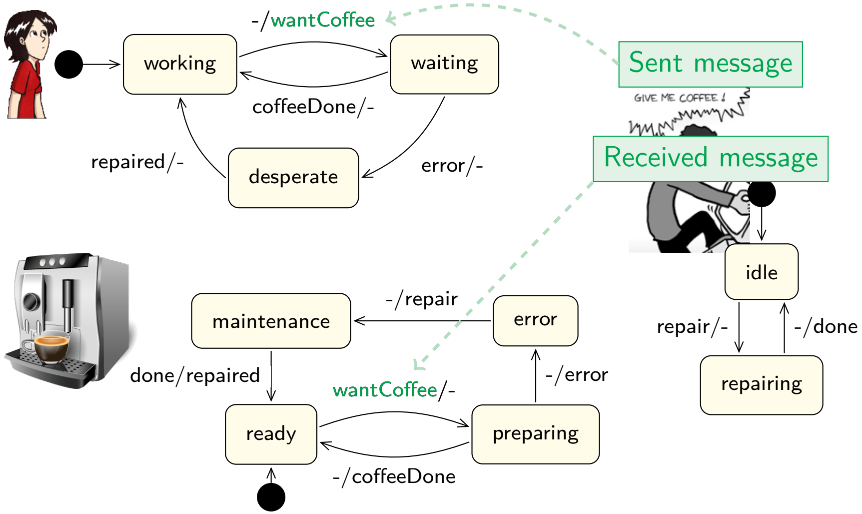
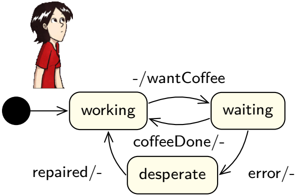
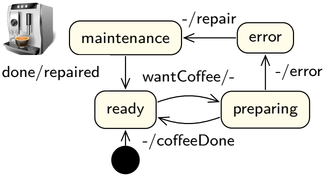
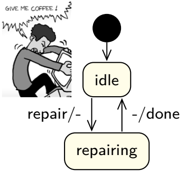


Image courtesy of “Piled Higher and Deeper” by Jorge Cham `www.phdcomics.com`

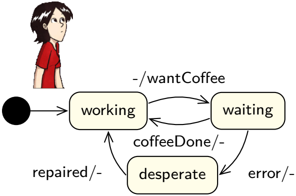
Sequence Diagram

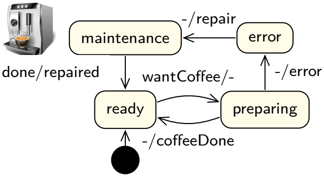


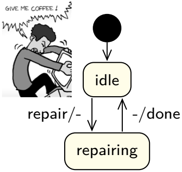


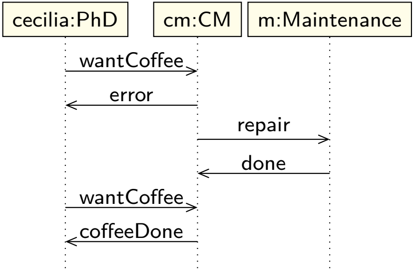


Sequence Diagram

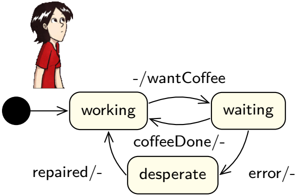


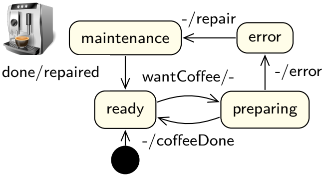


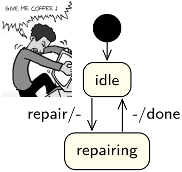


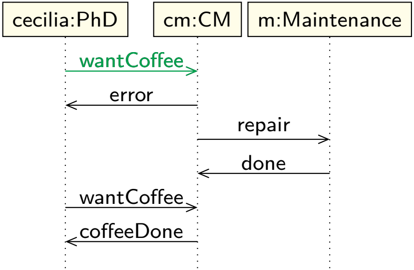


Sequence Diagram

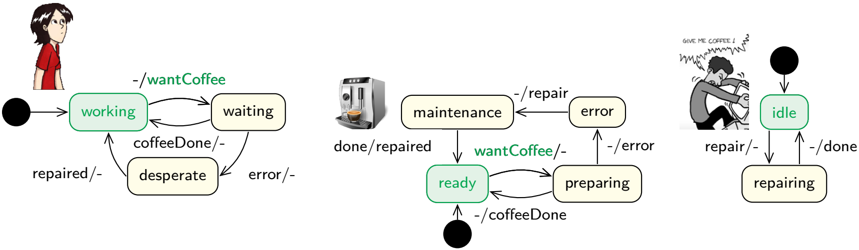


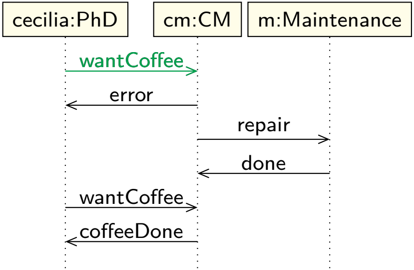




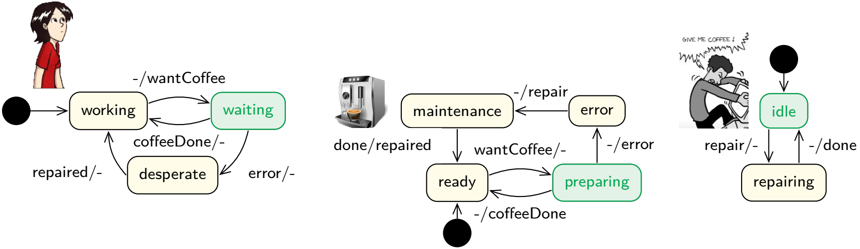


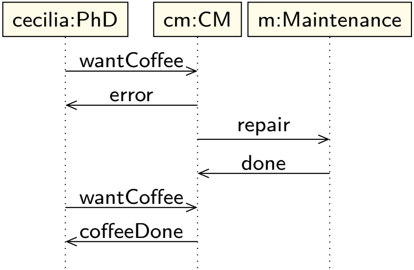
Sequence Diagram



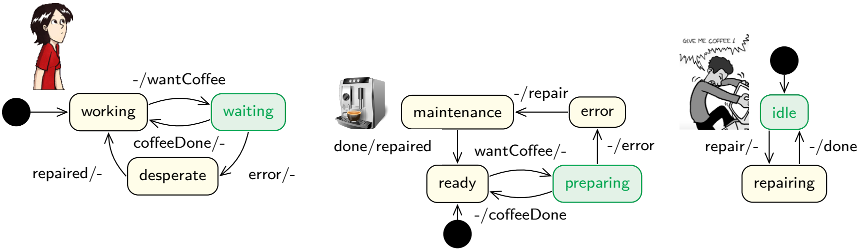


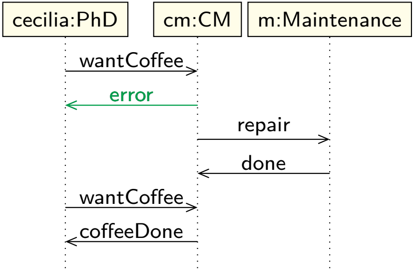
Sequence Diagram



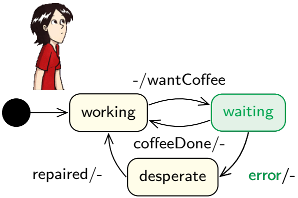


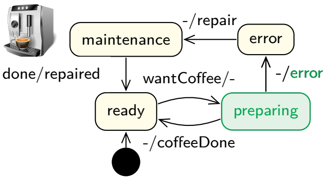
Sequence Diagram

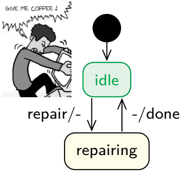


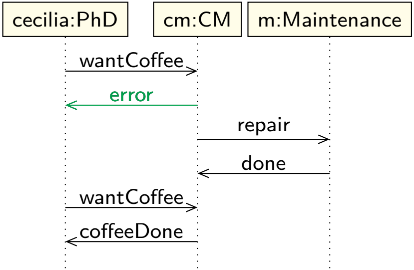


Sequence Diagram

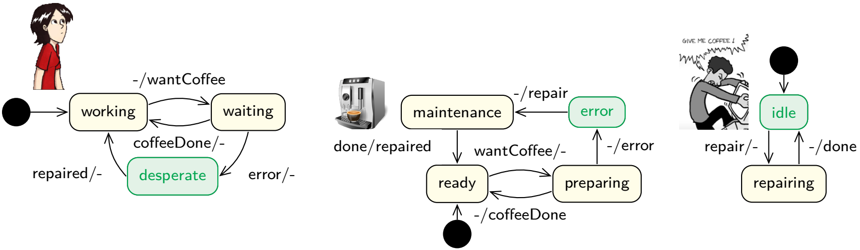


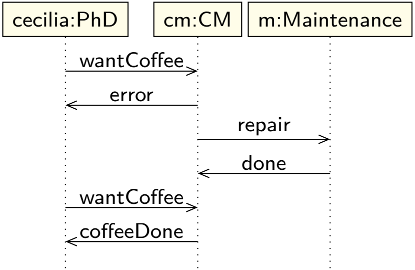




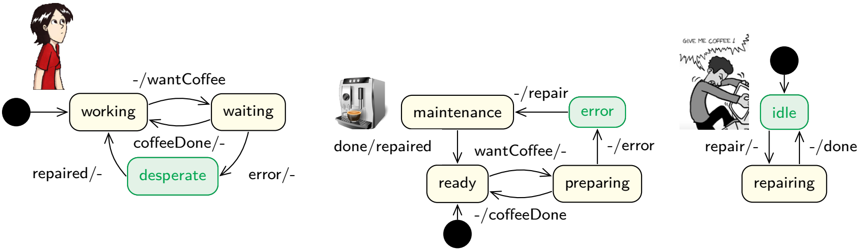


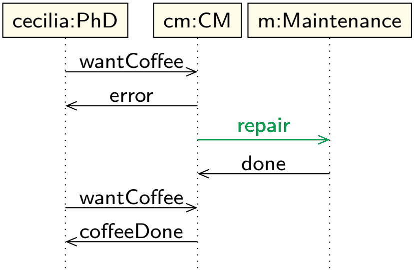
Sequence Diagram



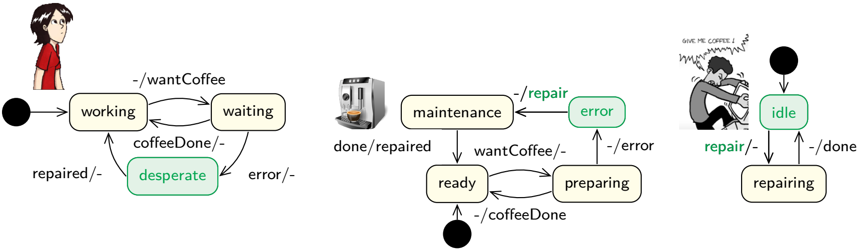


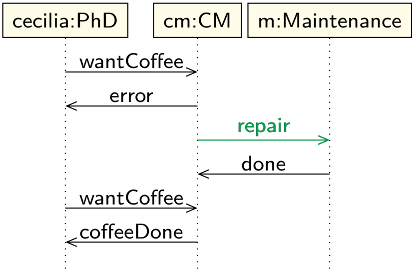
Sequence Diagram



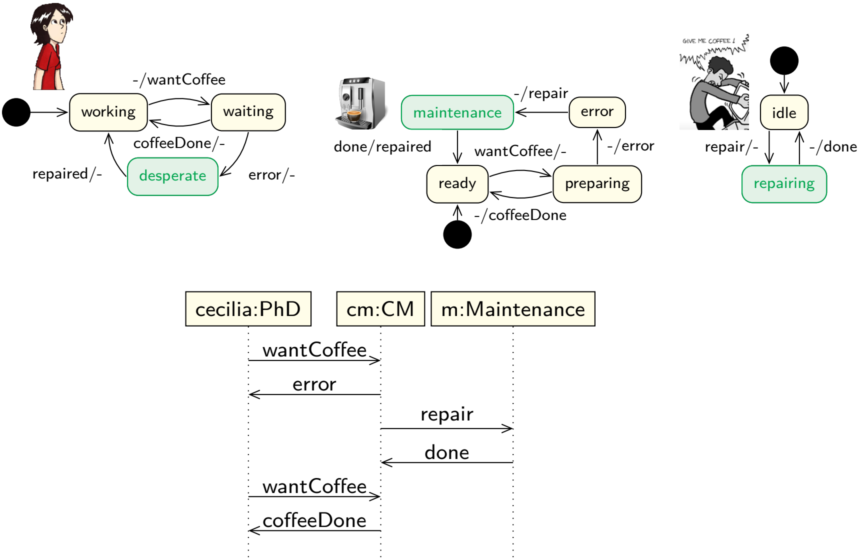


Sequence Diagram

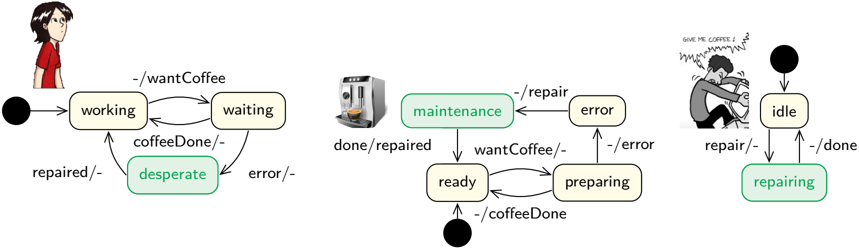


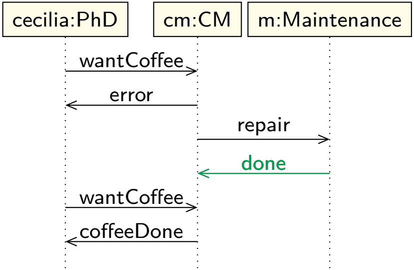


Sequence Diagram

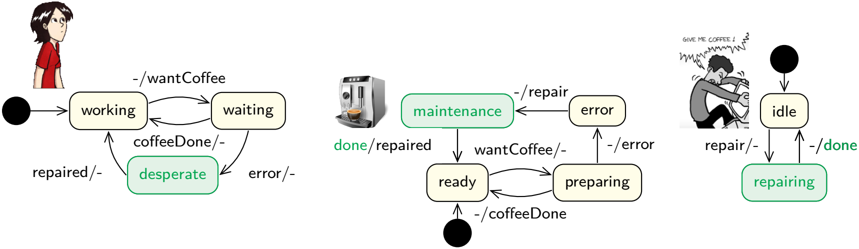


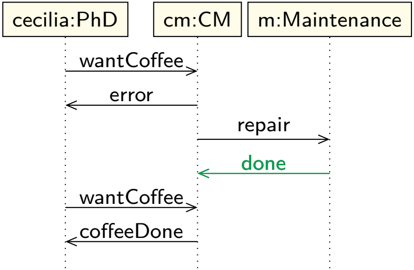
Sequence Diagram



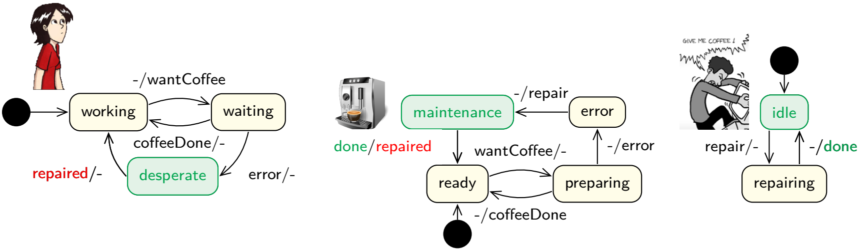


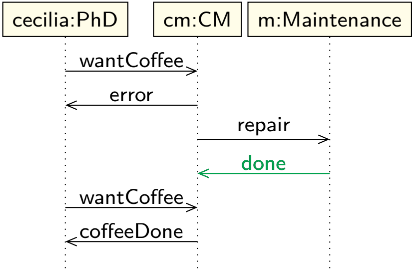
Sequence Diagram



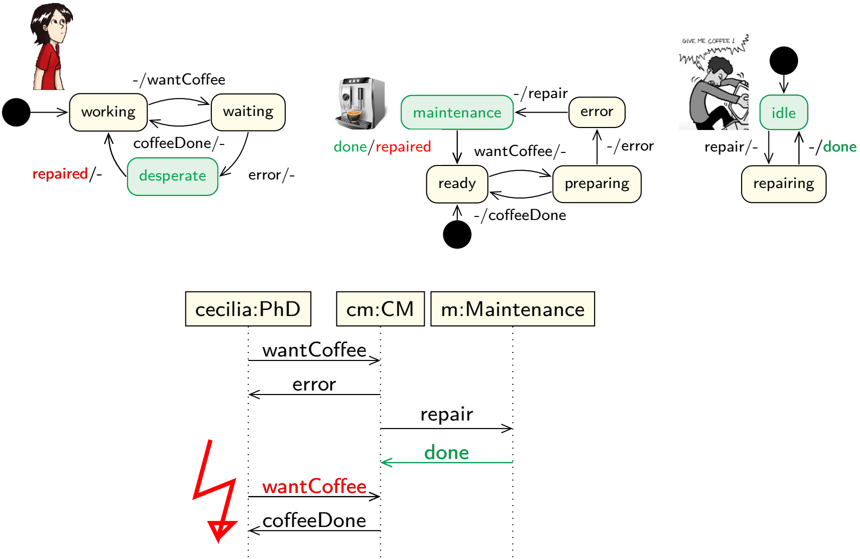


Sequence Diagram

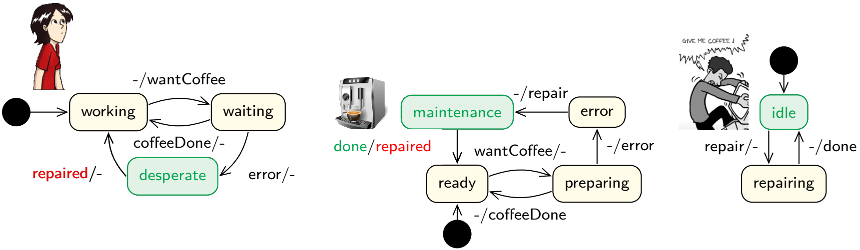


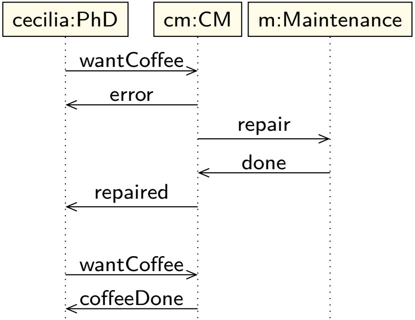


Sequence Diagram

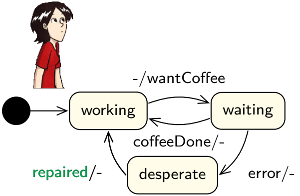


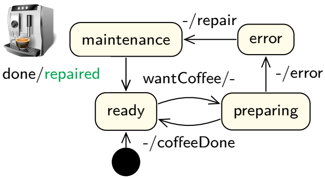
Sequence Diagram

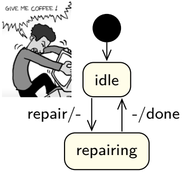


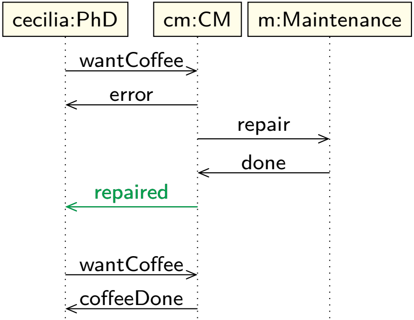


Sequence Diagram

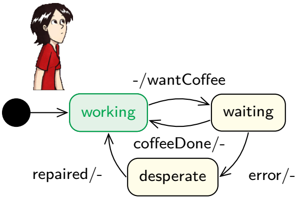


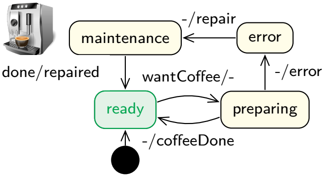


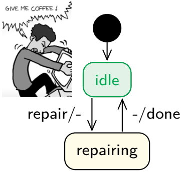


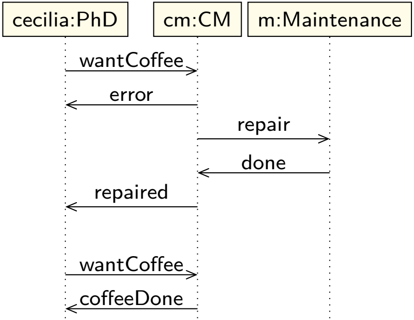


Sequence Diagram

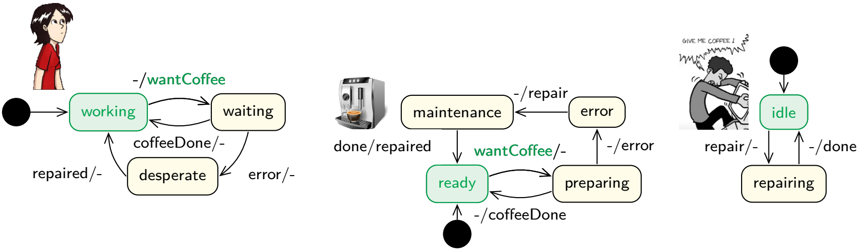


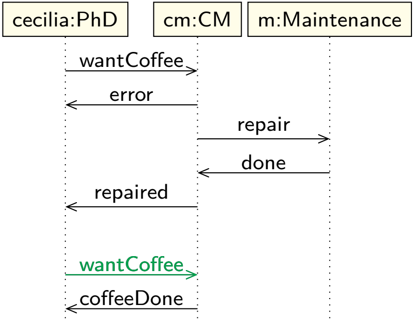




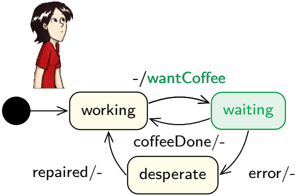


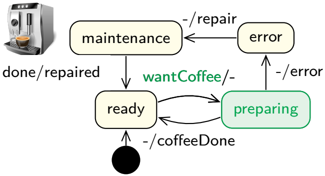
Sequence Diagram

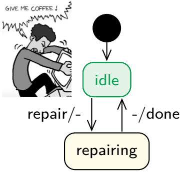


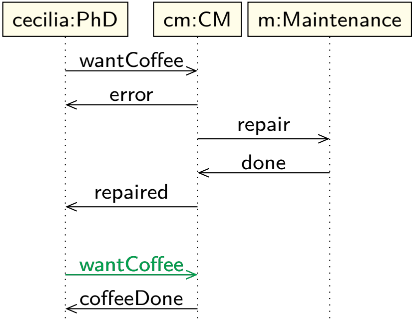


Sequence Diagram

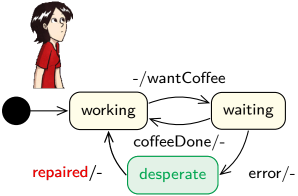


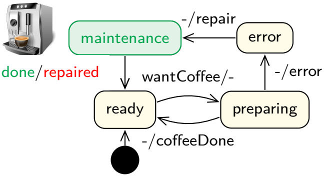


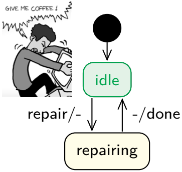


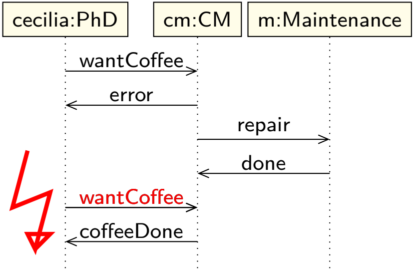


Sequence Diagram

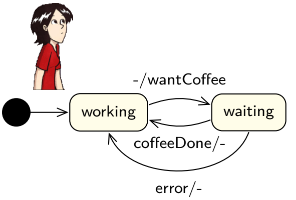


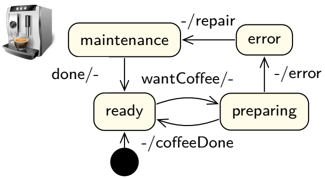


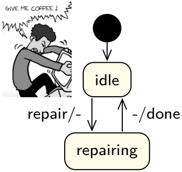


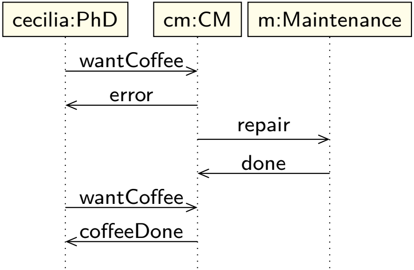


Sequence Diagram









The Problem

Check whether a sequence diagram is consistent with a set of state machines: Can the sequence diagram be executed (not necessarily from the initial states)?

The Problem

Check whether a sequence diagram is consistent with a set of state machines: Can the sequence diagram be executed (not necessarily from the initial states)?

Why not use a **model checker**?

The Problem

Check whether a sequence diagram is consistent with a set of state machines: Can the sequence diagram be executed (not necessarily from the initial states)?

Why not use a **model checker**?

Semantic differences!

The Problem

Check whether a sequence diagram is consistent with a set of state machines: Can the sequence diagram be executed (not necessarily from the initial states)?

Why not use a **model checker**?

Semantic differences!

Direct encoding to **SAT**

Propositional Satisfiability (SAT)

Propositional Satisfiability (SAT)

Instance:	Propositional formula φ.
Question:	Is there a satisfying truth assignment for φ?

Propositional Satisfiability (SAT)

Instance:	Propositional formula φ.
Question:	Is there a satisfying truth assignment for φ?

Example

$\varphi = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$

φ satisfiable?

Propositional Satisfiability (SAT)

Instance:	Propositional formula φ.
Question:	Is there a satisfying truth assignment for φ?

Example

$\varphi = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$

φ satisfiable? Yes! For example: $x_1 = x_3 =$ true, $x_2 =$ false.

Propositional Satisfiability (SAT)

Instance:	Propositional formula φ.
Question:	Is there a satisfying truth assignment for φ?

Example

$\varphi = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$
φ satisfiable? Yes! For example: $x_1 = x_3 =$ true, $x_2 =$ false.

- SAT solvers are very powerful – they handle millions of variables.
- Using SAT as a "programming language" is very successful.

Propositional Satisfiability (SAT)

Instance:	Propositional formula φ.
Question:	Is there a satisfying truth assignment for φ?

Example

$\varphi = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$

φ satisfiable? Yes! For example: $x_1 = x_3 =$ true, $x_2 =$ false.

- SAT solvers are very powerful – they handle millions of variables.
- Using SAT as a "programming language" is very successful.

Use SAT for our problem!

Encoding

Variables

Variables representing **transitions** at positions:

$$t^i$$

Encoding

Variables

Variables representing **transitions** at positions:

$$t^i$$

Variables representing **message symbols** at positions:

$$\mathit{eff}^i,\ tr^i,\ \mathsf{trg}(t)^i,\ a^i,\ \mathsf{symb}(m)^i$$

Encoding

Variables

Variables representing **transitions** at positions:

$$t^i$$

Variables representing **message symbols** at positions:

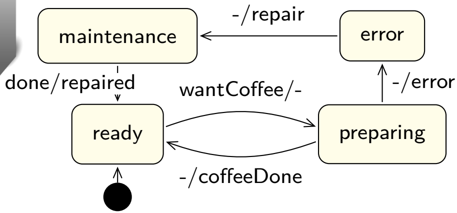
$$eff^i,\ tr^i,\ \mathsf{trg}(t)^i,\ a^i,\ \mathsf{symb}(m)^i$$

Variables representing **states** at positions:

$$\mathsf{src}(t)^i, \mathsf{tgt}(t)^i,\ \mathsf{int}(t)^i$$

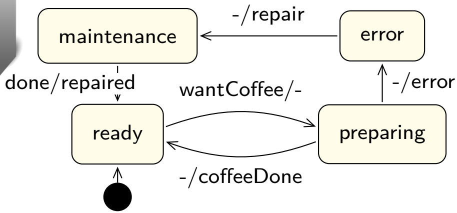
Encoding

Extended State Machine



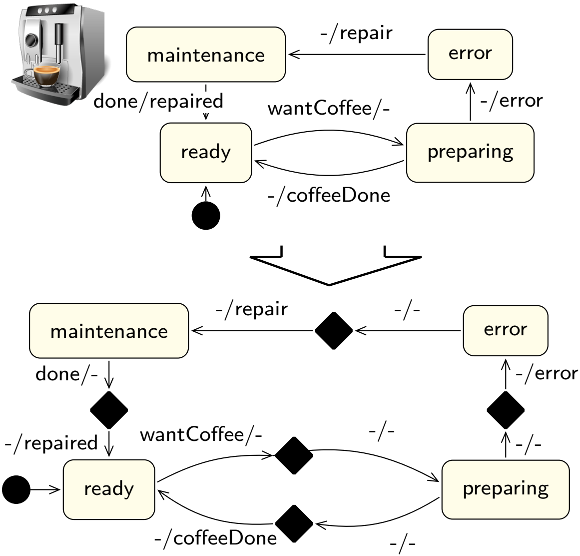
Encoding

Extended State Machine



Encoding

Extended State Machine



Encoding

Subformulas

- Initial state

$$\bigwedge_{i=1}^{l} \left(\iota_i^0 \wedge \bigwedge_{s \in S_i \cup \mathcal{S}_i^*, s \neq \iota_i} \overline{s}^0 \right) \wedge \bigwedge_{a \in \mathcal{A}} \overline{a}^0$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{t \in \mathcal{T}} \Big[t^i \rightarrow \Big(\mathsf{src}(t)^i \wedge \mathsf{int}(t)^{i+1} \wedge \mathsf{trg}(t)^i \wedge \overline{\mathsf{trg}(t)}^{i+1} \wedge \\ \wedge \bigwedge_{\mathit{eff} \in \mathsf{eff}(t)} \Big(\overline{\mathit{eff}}^i \wedge \mathit{eff}^{i+1} \Big) \Big) \Big]$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition
- There must be a cause whenever the polarity of a symbol changes

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{trg \in \mathcal{A}} \left[trg^i \wedge \overline{trg}^{i+1} \rightarrow \bigvee_{t \in \mathcal{T}, trg = \mathsf{trg}(t)} t^i \right]$$

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{eff \in \mathcal{A}} \left[\overline{eff}^i \wedge eff^{i+1} \rightarrow \bigvee_{t \in \mathcal{T}, eff \in \mathsf{eff}(t)} t^i \right]$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition
- There must be a cause whenever the polarity of a symbol changes
- Leaving a state is caused by a transition

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{s \in S} \left[s^i \wedge \overline{s}^{i+1} \rightarrow \bigvee_{t \in \mathcal{T}, s = \mathsf{src}(t)} t^i \right]$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition
- There must be a cause whenever the polarity of a symbol changes
- Leaving a state is caused by a transition
- Consume symbol to finish transiton

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{t \in \mathcal{T}, \mathsf{eff}(t) \neq \epsilon} \left[\left(\mathsf{int}(t)^{i} \wedge \mathsf{int}(t)^{i+1} \right) \rightarrow \mathsf{eff}(t)^{i+1} \right]$$

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{t \in \mathcal{T}, \mathsf{eff}(t) \neq \epsilon} \left[\left(\mathsf{int}(t)^{i} \wedge \overline{\mathsf{int}(t)}^{i+1} \right) \rightarrow \overline{\mathsf{eff}(t)}^{i+1} \right]$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition
- There must be a cause whenever the polarity of a symbol changes
- Leaving a state is caused by a transition
- Consume symbol to finish transiton
- Every machine is in exactly one state at every time

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{j=1}^{l} \left[\left(\bigvee_{s \in (S_j \cup S_j^*)} s^i \right) \wedge \bigwedge_{\substack{s_1, s_2 \in (S_j \cup S_j^*), \\ s_1 \neq s_2}} \left(\overline{s_1}^i \vee \overline{s_2}^i \right) \right]$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition
- There must be a cause whenever the polarity of a symbol changes
- Leaving a state is caused by a transition
- Consume symbol to finish transiton
- Every machine is in exactly one state at every time
- Move to target state when effect is consumed

$$\bigwedge_{i=0}^{k'-1} \bigwedge_{t\in\mathcal{T}} \left[\left(\mathsf{int}(t)^i \wedge \bigwedge_{\mathit{eff}\in\mathsf{eff}(t)} \overline{\mathit{eff}}^{\,i+1}\right) \to \left(\overline{\mathsf{int}(t)}^{\,i+1} \wedge \mathsf{tgt}(t)^{i+1}\right)\right]$$

Encoding

Subformulas

- Initial state
- State changes after initiating a transition
- There must be a cause whenever the polarity of a symbol changes
- Leaving a state is caused by a transition
- Consume symbol to finish transiton
- Every machine is in exactly one state at every time
- Move to target state when effect is consumed
- Sequence diagram

$$\bigwedge_{\substack{i\in[1,\ldots,n],\\ j\in[k,k+4,\ldots,k+4n]}} \Bigg[\mathsf{symb}(m_i)^j \wedge \overline{\mathsf{symb}(m_i)}^{j+1} \wedge \bigvee_{\substack{t\in\mathsf{trans}(\mathsf{snd}(m_i)),\\ \mathsf{eff}(t)=m_i}} \left(\mathsf{int}(t)^j \wedge \overline{\mathsf{int}(t)}^{j+1}\right) \wedge \bigwedge_{\substack{a\in\mathcal{A}\\ a\neq m_i}} \left(\left(a^j \to a^{j+1}\right) \wedge \left(a^{j+1} \to a^{j+2}\right) \wedge \left(a^{j+2} \to a^{j+3}\right)\right)\Bigg]$$

Encoding

Interpretation

Satisfiable: Message sequence computed from logical model

Encoding

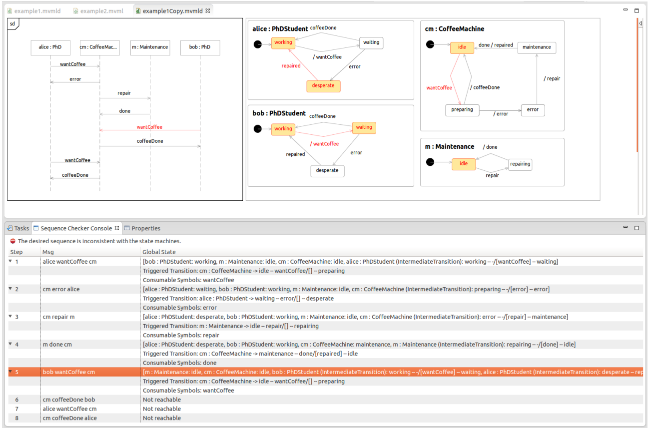
Interpretation

Satisfiable: Message sequence computed from logical model

Unsatisfiable: Sequence diagram cannot be executed

Implementation

- Multiview Modeling Language (MVML), inspired by UML
- Based on Ecore and EMF
- Graphical modeling editor



Download at `http://modelevolution.org/updatesite`

Evaluation

Random model generator

Evaluation

Random model generator

Different categories according to size

	small	medium	large
minNrStates	2	4	7
maxNrStates	3	6	10
minNrTrans	4	8	21
maxNrTrans	6	12	30
nrLifelines	3	5	8
nrMessages	4	10	20

Evaluation

Random model generator

Different categories according to size

	small	medium	large
minNrStates	2	4	7
maxNrStates	3	6	10
minNrTrans	4	8	21
maxNrTrans	6	12	30
nrLifelines	3	5	8
nrMessages	4	10	20

	small	medium	large
Encoding time *satisfiable* (ms)	11	180	2,543
Solving time *satisfiable* (ms)	4	201	9,476
Encoding time *unsatisfiable* (ms)	34	970	27,848
Solving time *unsatisfiable* (ms)	8	727	179,914
Nr variables	1,802	12,746	88,560
Nr clauses	9,652	118,245	1,700,101
Nr instances *satisfiable*	837	750	803
Nr instances *unsatisfiable*	163	250	197

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines
- Implementation, freely available

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines
- Implementation, freely available
- Thorough evaluation

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines
- Implementation, freely available
- Thorough evaluation

What we consider next

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines
- Implementation, freely available
- Thorough evaluation

What we consider next

- Tuning and optimizing of the encoding

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines
- Implementation, freely available
- Thorough evaluation

What we consider next

- Tuning and optimizing of the encoding
- Additional language concepts requiring more complex encodings

Summary and Future Work

We presented

- SAT-based theoretical framework to check consistency between sequence diagrams and state machines
- Implementation, freely available
- Thorough evaluation

What we consider next

- Tuning and optimizing of the encoding
- Additional language concepts requiring more complex encodings
- More powerful target languages such as quantified Boolean formulae