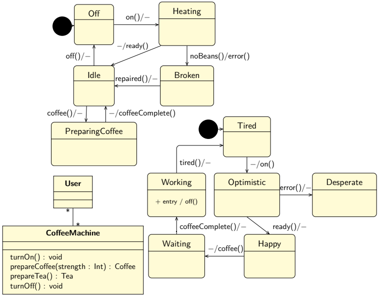
Towards Scenario-Based Testing of UML Diagrams

Magdalena Widl

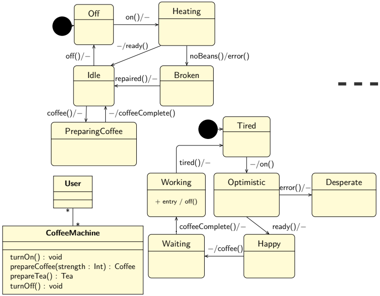
Knowledge-Based Systems Group
Vienna University of Technology

Model-Driven Engineering

Model-Driven Engineering

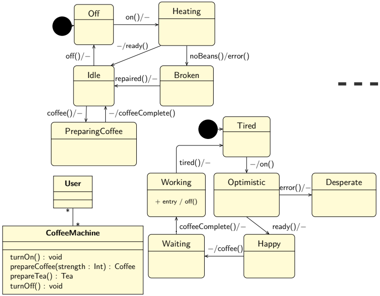


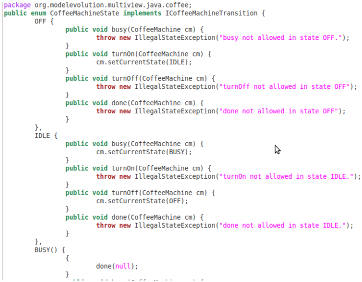
Model-Driven Engineering




```java
package org.modelevolution.multiview.java.coffee;
public enum CoffeeMachineState implements ICoffeeMachineTransition {
        OFF {
                public void busy(CoffeeMachine cm) {
                        throw new IllegalStateException("busy not allowed in state OFF.");
                }
                public void turnOn(CoffeeMachine cm) {
                        cm.setCurrentState(IDLE);
                }
                public void turnOff(CoffeeMachine cm) {
                        throw new IllegalStateException("turnOff not allowed in state OFF");
                }
                public void done(CoffeeMachine cm) {
                        throw new IllegalStateException("done not allowed in state OFF");
                }
        },
        IDLE {
                public void busy(CoffeeMachine cm) {
                        cm.setCurrentState(BUSY);
                }
                public void turnOn(CoffeeMachine cm) {
                        throw new IllegalStateException("turnOn not allowed in state IDLE.");
                }
                public void turnOff(CoffeeMachine cm) {
                        cm.setCurrentState(OFF);
                }
                public void done(CoffeeMachine cm) {
                        throw new IllegalStateException("done not allowed in state IDLE.");
                }
        },
        BUSY() {
                {
                        done(null);
                }
```

Model-Driven Engineering

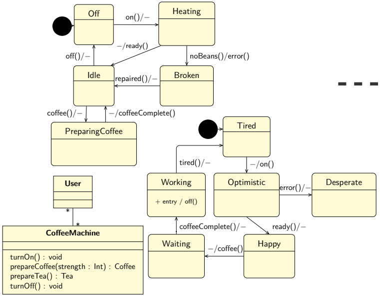


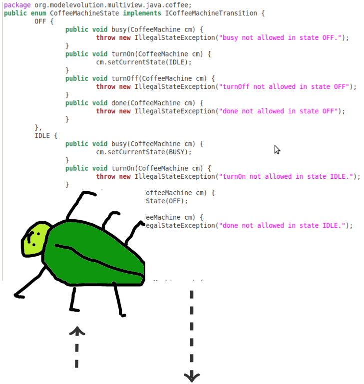


```
package org.modelevolution.multiview.java.coffee;
public enum CoffeeMachineState implements ICoffeeMachineTransition {
        OFF {
                public void busy(CoffeeMachine cm) {
                        throw new IllegalStateException("busy not allowed in state OFF.");
                }
                public void turnOn(CoffeeMachine cm) {
                        cm.setCurrentState(IDLE);
                }
                public void turnOff(CoffeeMachine cm) {
                        throw new IllegalStateException("turnOff not allowed in state OFF");
                }
                public void done(CoffeeMachine cm) {
                        throw new IllegalStateException("done not allowed in state OFF");
                }
        },
        IDLE {
                public void busy(CoffeeMachine cm) {
                        cm.setCurrentState(BUSY);
                }
                public void turnOn(CoffeeMachine cm) {
                        throw new IllegalStateException("turnOn not allowed in state IDLE.");
                }
                public void turnOff(CoffeeMachine cm) {
                        cm.setCurrentState(OFF);
                }
                public void done(CoffeeMachine cm) {
                        throw new IllegalStateException("done not allowed in state IDLE.");
                }
        },
        BUSY() {
                {
                        done(null);
                }
```

Software verification, testing

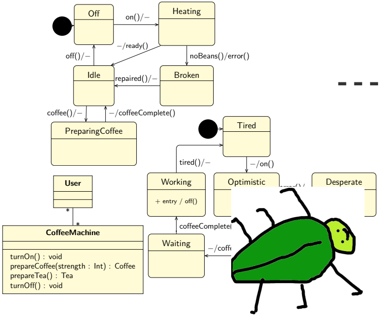
Model-Driven Engineering

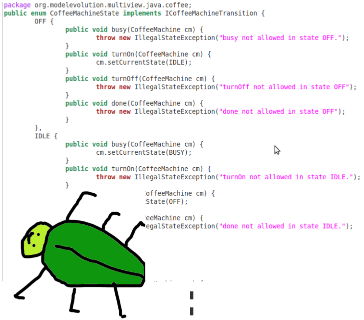




Software verification, testing

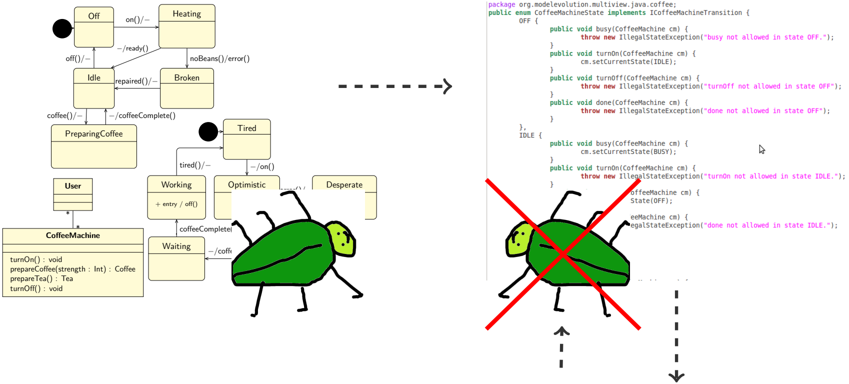
Model-Driven Engineering





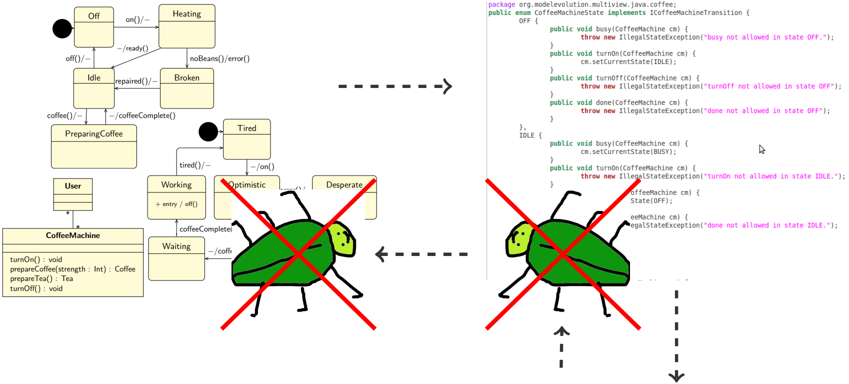
Software verification, testing

Model-Driven Engineering



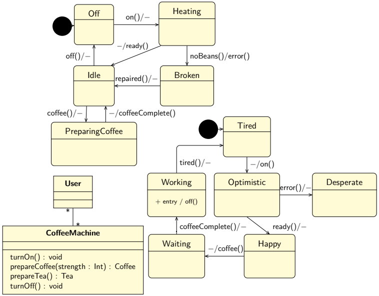
Software verification, testing

Model-Driven Engineering



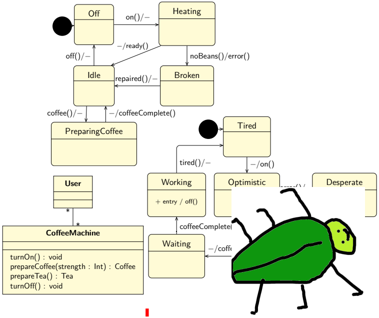
Software verification, testing

Model-Driven Engineering



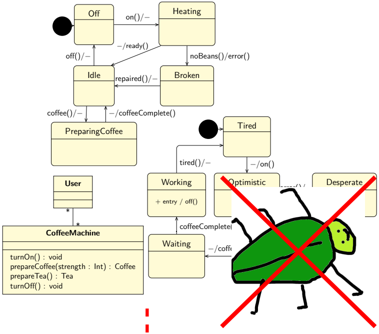
Model verification!

Model-Driven Engineering



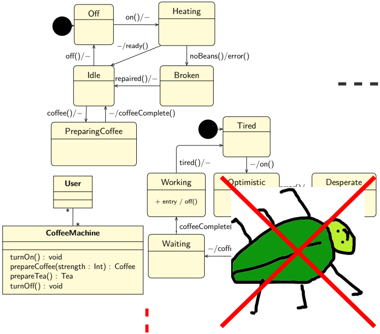
Model verification!

Model-Driven Engineering



Model verification!

Model-Driven Engineering

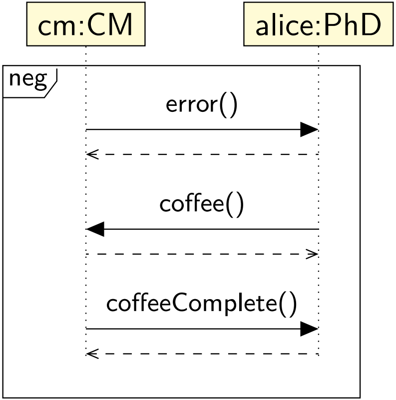



```java
package org.modelevolution.multiview.java.coffee;
public enum CoffeeMachineState implements ICoffeeMachineTransition {
        OFF {
                public void busy(CoffeeMachine cm) {
                        throw new IllegalStateException("busy not allowed in state OFF.");
                }
                public void turnOn(CoffeeMachine cm) {
                        cm.setCurrentState(IDLE);
                }
                public void turnOff(CoffeeMachine cm) {
                        throw new IllegalStateException("turnOff not allowed in state OFF");
                }
                public void done(CoffeeMachine cm) {
                        throw new IllegalStateException("done not allowed in state OFF");
                }
        },
        IDLE {
                public void busy(CoffeeMachine cm) {
                        cm.setCurrentState(BUSY);
                }
                public void turnOn(CoffeeMachine cm) {
                        throw new IllegalStateException("turnOn not allowed in state IDLE.");
                }
                public void turnOff(CoffeeMachine cm) {
                        cm.setCurrentState(OFF);
                }
                public void done(CoffeeMachine cm) {
                        throw new IllegalStateException("done not allowed in state IDLE.");
                }
        },
        BUSY() {
                {
                        done(null);
                }
```

Model verification!

Scenario-based Testing

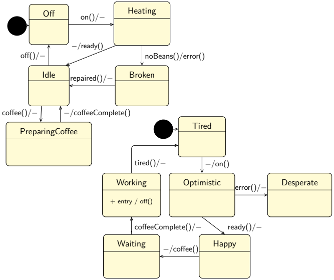
Scenarios can be represented by models



The Big Picture

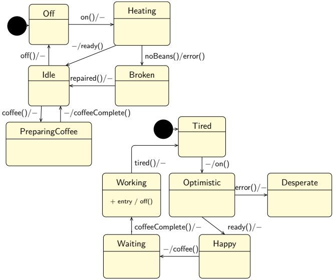
The Big Picture

Model to generate code
(e.g. state machines)

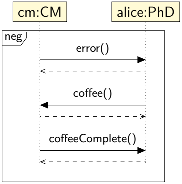


The Big Picture

Model to generate code (e.g. state machines)

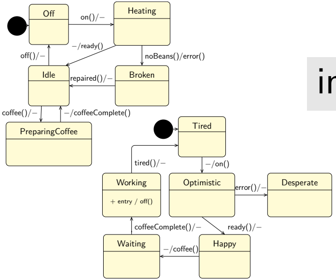


Scenario



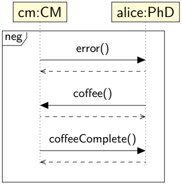
The Big Picture

Model to generate code
(e.g. state machines)



implements

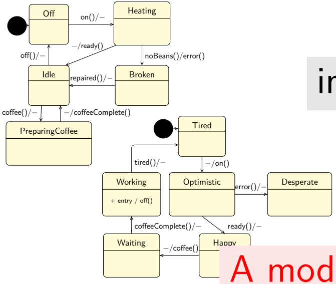
Scenario



?

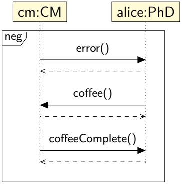
The Big Picture

Model to generate code
(e.g. state machines)



implements

Scenario

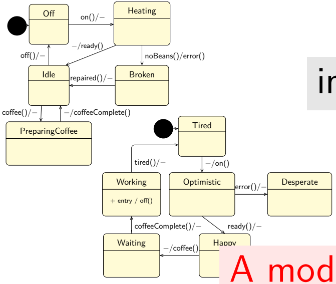


?

A model checking problem!

The Big Picture

Model to generate code
(e.g. state machines)



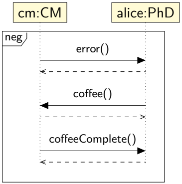
$\mathcal{K}$

Kripke structure

implements

A model checking problem!

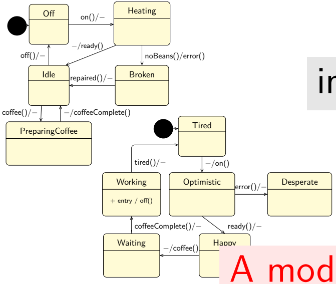
Scenario



?

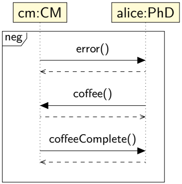
The Big Picture

Model to generate code
(e.g. state machines)



implements

Scenario



?

A model checking problem!

$\mathcal{K}$

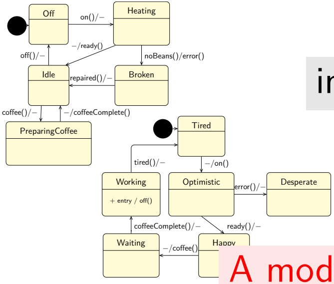
Kripke structure

ϕ

Temporal logic

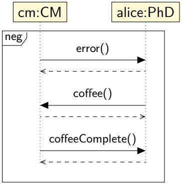
The Big Picture

Model to generate code (e.g. state machines)



implements

Scenario



?

A model checking problem!

$\mathcal{K}$

Kripke structure

$\models$

ϕ

Temporal logic

?

Idea & Challenges

Use model checking to verify a set of state machines against a set of sequence diagrams.

Idea & Challenges

Use model checking to verify a set of state machines against a set of sequence diagrams.

1. Formal problem definition, formal semantics of models

Idea & Challenges

Use model checking to verify a set of state machines against a set of sequence diagrams.

1. Formal problem definition, formal semantics of models
2. Translation to model checker

Idea & Challenges

Use model checking to verify a set of state machines against a set of sequence diagrams.

1. Formal problem definition, formal semantics of models
2. Translation to model checker
3. Presentation of result (counterexample)

Idea & Challenges

Use model checking to verify a set of state machines against a set of sequence diagrams.

1. **Formal problem definition, formal semantics of models**
2. Translation to model checker
3. Presentation of result (counterexample)

Problem Instance

A triple $(\mathcal{A}, \mathcal{M}, \mathcal{S})$, where

- $\mathcal{A}$ a set of actions (vocabulary)
- $\mathcal{M}$ a set of state machines, and
- $\mathcal{S}$ a set of sequence diagrams.

Problem Instance

A triple $(\mathcal{A}, \mathcal{M}, \mathcal{S})$, where

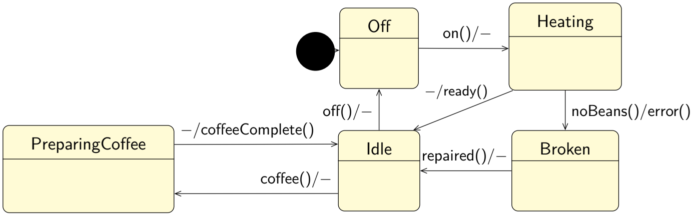
- $\mathcal{A}$ a set of actions (vocabulary)
- $\mathcal{M}$ a set of state machines, and
- $\mathcal{S}$ a set of sequence diagrams.

Problem Instance

A triple $(\mathcal{A}, \mathcal{M}, \mathcal{S})$, where

- $\mathcal{A}$ a set of actions (vocabulary)
- $\mathcal{M}$ a set of state machines, and
- $\mathcal{S}$ a set of sequence diagrams.

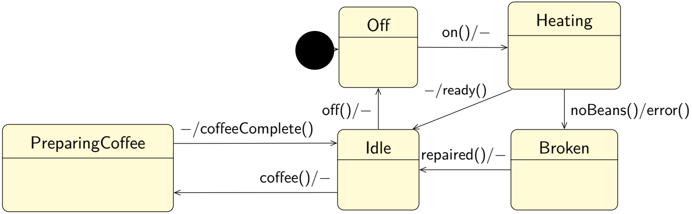
State Machine



A quintuple $(S, \iota, A^{tr}, A^{eff}, T)$, where

- S is a set of states
- $\iota \in S$ is an initial state
- $A^{tr} \in \mathcal{A}$ is a set of triggers
- $A^{eff} \in \mathcal{A}$ is a set of effects
- $T \subseteq S \times A^{tr} \times P(A^{eff}) \times S$ is a set of transitions

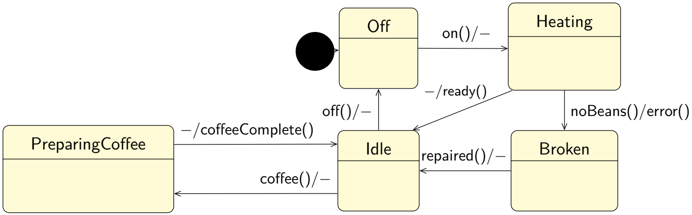
State Machine



A quintuple $(S, \iota, A^{tr}, A^{eff}, T)$, where

- S is a set of states
- $\iota \in S$ is an initial state
- $A^{tr} \in \mathcal{A}$ is a set of triggers
- $A^{eff} \in \mathcal{A}$ is a set of effects
- $T \subseteq S \times A^{tr} \times P(A^{eff}) \times S$ is a set of transitions

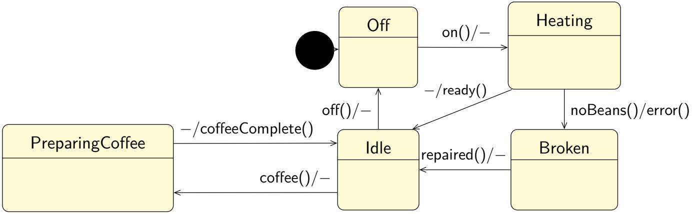
State Machine



A quintuple $(S, \iota, A^{tr}, A^{eff}, T)$, where

- S is a set of states
- $\iota \in S$ is an initial state
- $A^{tr} \in \mathcal{A}$ is a set of triggers
- $A^{eff} \in \mathcal{A}$ is a set of effects
- $T \subseteq S \times A^{tr} \times P(A^{eff}) \times S$ is a set of transitions

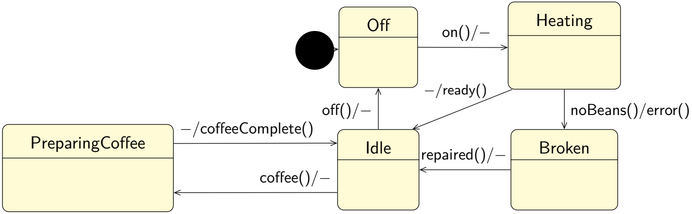
State Machine



A quintuple $(S, \iota, A^{tr}, A^{eff}, T)$, where

- S is a set of states
- $\iota \in S$ is an initial state
- $A^{tr} \in \mathcal{A}$ is a set of triggers
- $A^{eff} \in \mathcal{A}$ is a set of effects
- $T \subseteq S \times A^{tr} \times P(A^{eff}) \times S$ is a set of transitions

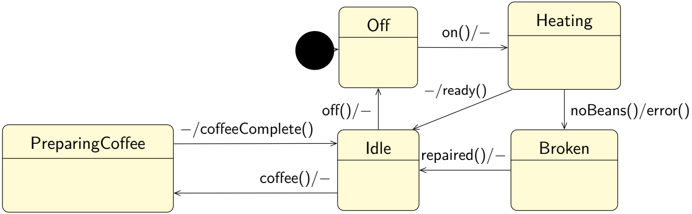
State Machine



A quintuple $(S, \iota, A^{tr}, A^{eff}, T)$, where

- S is a set of states
- $\iota \in S$ is an initial state
- $A^{tr} \in \mathcal{A}$ is a set of triggers
- $A^{eff} \in \mathcal{A}$ is a set of effects
- $T \subseteq S \times A^{tr} \times P(A^{eff}) \times S$ is a set of transitions

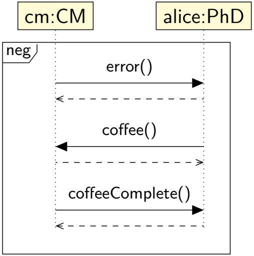
State Machine



A quintuple $(S, \iota, A^{tr}, A^{\mathit{eff}}, T)$, where

- S is a set of states
- $\iota \in S$ is an initial state
- $A^{tr} \in \mathcal{A}$ is a set of triggers
- $A^{\mathit{eff}} \in \mathcal{A}$ is a set of effects
- $T \subseteq S \times A^{tr} \times P(A^{\mathit{eff}}) \times S$ is a set of transitions

Sequence Diagram

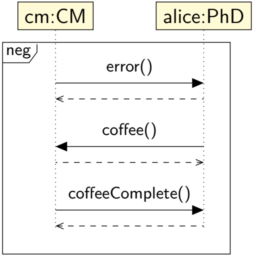


A triple (L, m, N), where

- L a set of lifelines
- $m : L \to \mathcal{M}$ a function assigning a state machine to each lifeline
- N a sequence of elements in $L \times \mathcal{A} \times L$

Only *neg* fragments.

Sequence Diagram

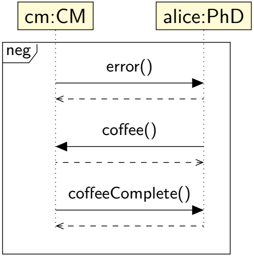


A triple (L, m, N), where

- L a set of lifelines
- $m : L \to \mathcal{M}$ a function assigning a state machine to each lifeline
- N a sequence of elements in $L \times \mathcal{A} \times L$

Only *neg* fragments.

Sequence Diagram

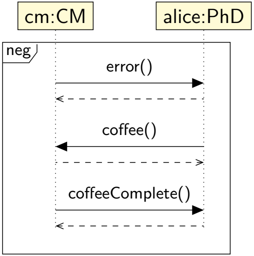


A triple (L, m, N), where

- L a set of lifelines
- $m : L \to \mathcal{M}$ a function assigning a state machine to each lifeline
- N a sequence of elements in $L \times \mathcal{A} \times L$

Only *neg* fragments.

Sequence Diagram

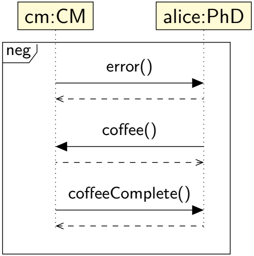


A triple (L, m, N), where

- L a set of lifelines
- $m : L \to \mathcal{M}$ a function assigning a state machine to each lifeline
- N a sequence of elements in $L \times \mathcal{A} \times L$

Only *neg* fragments.

Sequence Diagram



A triple (L, m, N), where

- L a set of lifelines
- $m : L \to \mathcal{M}$ a function assigning a state machine to each lifeline
- N a sequence of elements in $L \times \mathcal{A} \times L$

Only *neg* fragments.

Problem Definition

Given a set $\mathcal{M}$ of state machines

Problem Definition

Given a set $\mathcal{M}$ of state machines and a set $\mathcal{S}$ of *neg* sequences (scenarios),

Problem Definition

Given a set $\mathcal{M}$ of state machines and a set $\mathcal{S}$ of *neg* sequences (scenarios), does any $S \in \mathcal{S}$ occur on any path of the *composition* of $\mathcal{M}$?

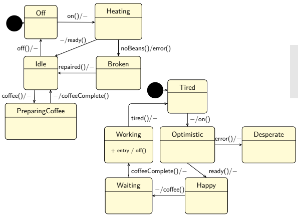
Idea & Challenges

Use model checking to verify a set of state machines against a set of sequence diagrams.

1. Formal problem definition, formal semantics of models
2. **Translation to model checker**
3. Presentation of result (counterexample)

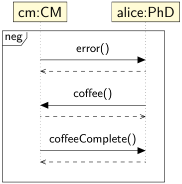
Model Checking with SPIN

State machines



implements

Scenario



?

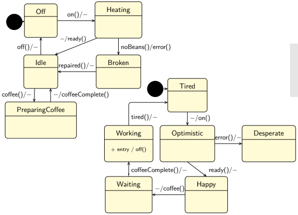
$\mathcal{K}$ $\models$ ϕ ?

Kripke structure

Temporal logic

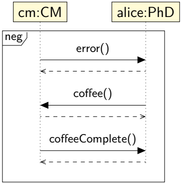
Model Checking with SPIN

State machines



implements

Scenario



?

$\mathcal{K}$ $\models$ ϕ ?

Kripke structure

Temporal logic

PROMELA processes

Never claims (Buchi Automata)
or (no)trace assertions

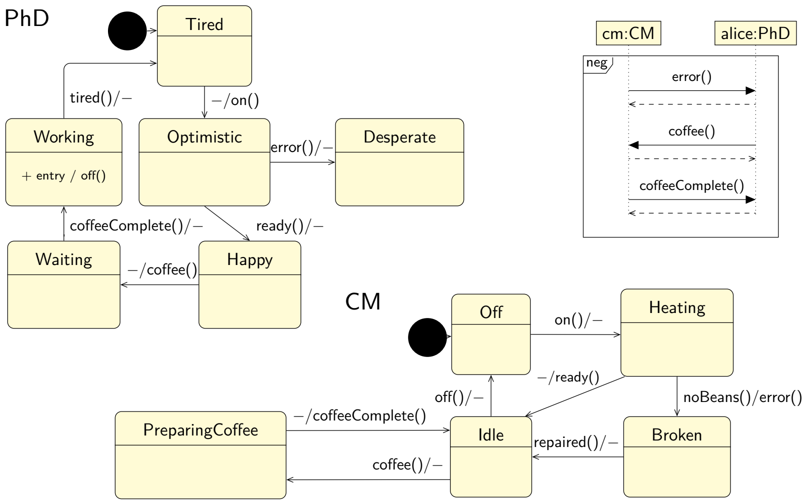
Encoding

Model Checker SPIN

- Each state machine a process
- Communication over synchronous channels
- Each sequence diagram a (no)trace assertion

Automatic code generation

Example


PhD
Tired
tired()/−
−/on()
Working
+ entry / off()
Optimistic
error()/−
Desperate
coffeeComplete()/−
ready()/−
Waiting
−/coffee()
Happy
cm:CM
alice:PhD
neg
error()
coffee()
coffeeComplete()
CM
Off
on()/−
Heating
−/ready()
off()/−
noBeans()/error()
PreparingCoffee
−/coffeeComplete()
Idle
repaired()/−
Broken
coffee()/−

Encoding

```
mtype = {on,off,ready,error,
coffee,complete};

chan phd = [0] of {mtype};
chan cm = [0] of {mtype};

active proctype PhD() {
mtype x;

Tired: atomic{cm!on;goto Optimistic;}

Optimistic: atomic{ phd?x;
    if :: x == error; goto Desperate;
       :: x == ready; goto Happy; fi;}

Desperate: goto Tired;

Happy: atomic{cm!coffee;goto Waiting;}

Waiting:
    atomic{phd?complete;goto Working;}

Working: atomic{ cm!off; goto Tired; }
}
```

```
active proctype CM() {
mtype x;

Off: atomic{ cm?on; goto Heating; }

Heating: atomic{
     if :: phd!error; goto Broken;
        :: phd!ready; goto Idle; fi;}

Broken: goto Off;

Idle: atomic{cm?x;
      if :: x == off; goto Off;
         :: x == coffee;
            goto PreparingCoffee; fi;}

PreparingCoffee:
     atomic{ phd!complete; goto Idle;}
}

notrace {    if ::    phd?error;
             cm?coffee;
             phd?complete; fi;
}
```

Challenges

1. Formal problem definition, formal semantics of models

Challenges

1. Formal problem definition, formal semantics of models
    - ✓ Definition of restricted state machine

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication
   - ✗ Semantics of non-*neg* fragments

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication
   - ✗ Semantics of non-*neg* fragments
2. Translation to model checker

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication
   - ✗ Semantics of non-*neg* fragments
2. Translation to model checker
   - ✓ Prototype in SPIN

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication
   - ✗ Semantics of non-*neg* fragments
2. Translation to model checker
   - ✓ Prototype in Spin
   - ✗ Better encodings in other model checkers

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication
   - ✗ Semantics of non-*neg* fragments
2. Translation to model checker
   - ✓ Prototype in SPIN
   - ✗ Better encodings in other model checkers
3. Presentation of result (counterexample)

Challenges

1. Formal problem definition, formal semantics of models
   - ✓ Definition of restricted state machine
   - ✗ Additional concepts like guards
   - ✓ Definition of restricted sequence diagram
   - ✗ Additional concepts like other combined fragments, asynchronous communication
   - ✗ Semantics of non-*neg* fragments
2. Translation to model checker
   - ✓ Prototype in SPIN
   - ✗ Better encodings in other model checkers
3. Presentation of result (counterexample)
   - ✗ Concrete syntax

Related Work

- Cimatti et al.: Hybrid Automata and Message Sequence Charts
- Li et al.: Petri nets and Message Sequence Charts
- CHARMY tool suite: Software Architecture with componen, state transition and sequence diagrams
- HUGO: State machines and collaboration diagrams with SPIN
- Other works in the area of synthesis (e.g. Uchitel et al.)