Model Checking of UML 2.0 Interactions

Alexander Knapp[1] Jochen Wuttke[2]

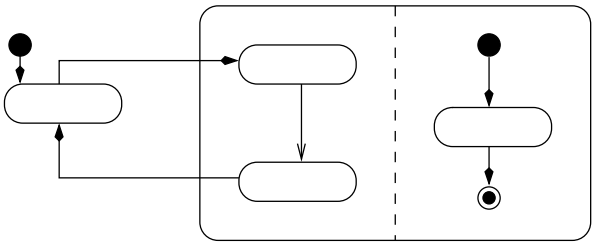
[1]Department of Computer Science
University of Munich

[2]Faculty of Informatics
University of Lugano

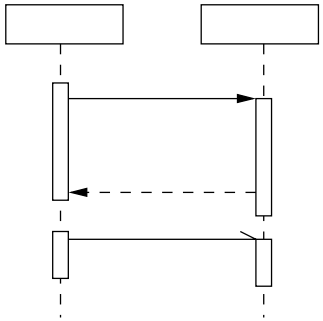
Critical Systems Development Using Modeling Languages,
CSDUML 2006

Setting the Context. . .

State machines

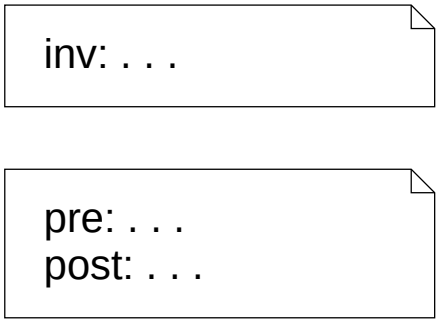


Interactions

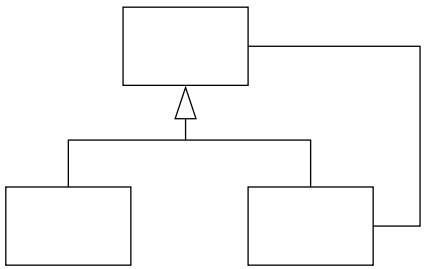


Consistency
Implementation
Refinement

OCL



Static structure



. . . with an Example

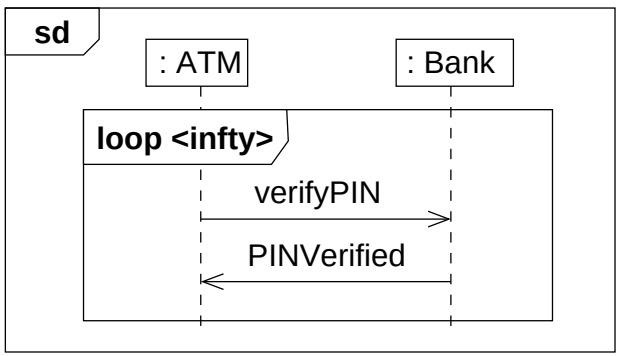
Customer/ Analyst

"An ATM should always give money to validated account holders."

. . . with an Example

Customer/ Analyst

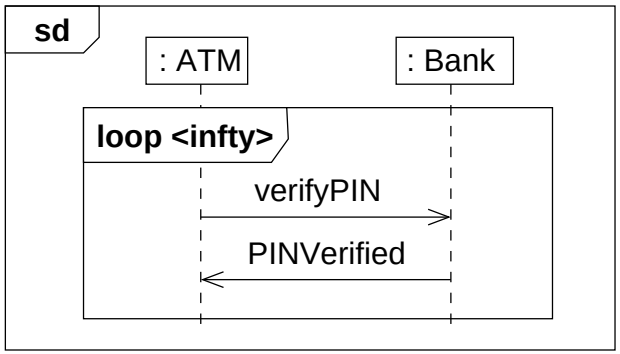
"An ATM should always give money to validated account holders."



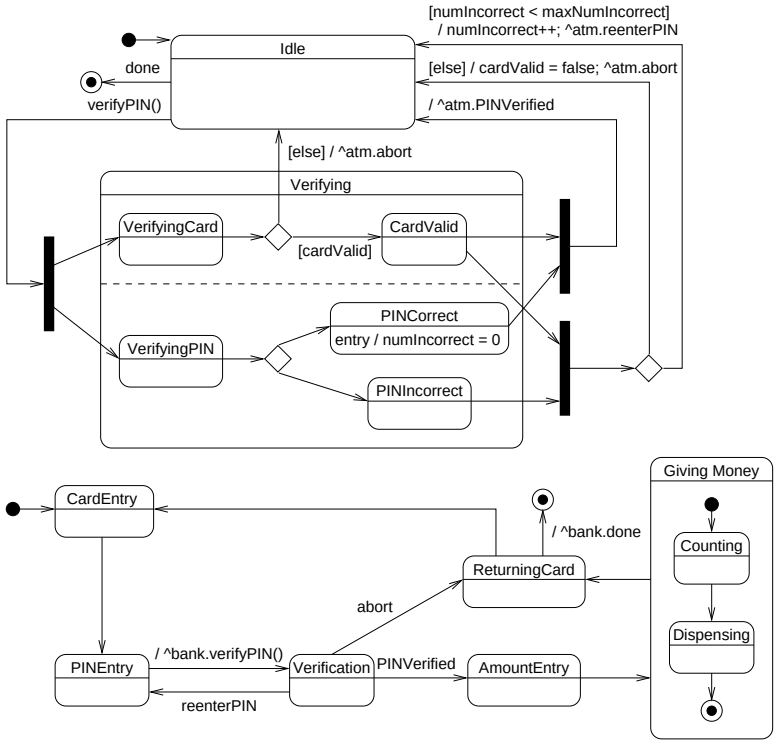
. . . with an Example

Customer/ Analyst

"An ATM should always give money to validated account holders."



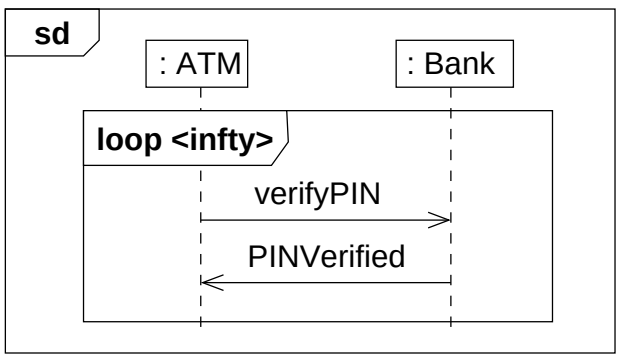
System Designer



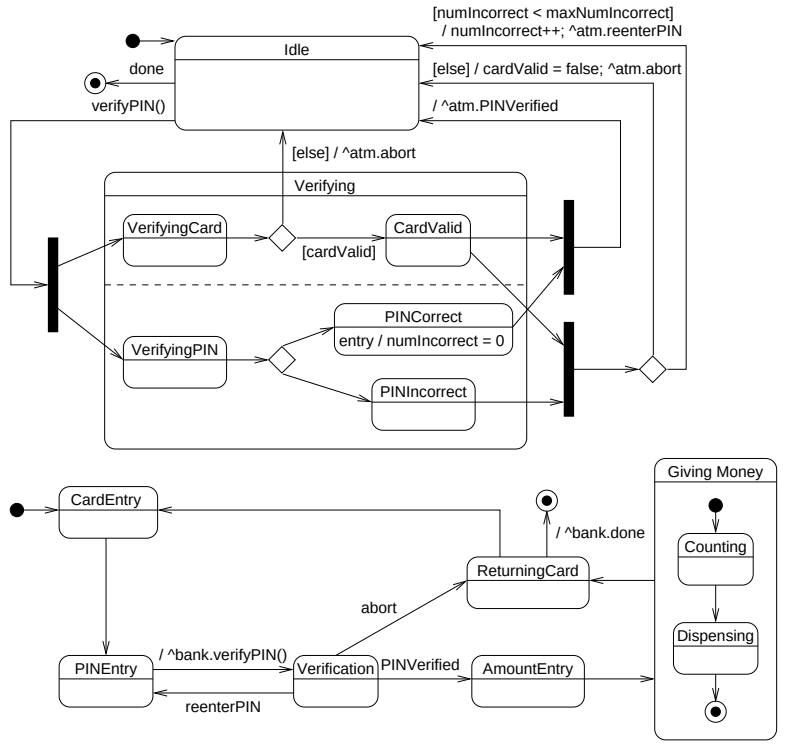
. . . with an Example

Customer/ Analyst

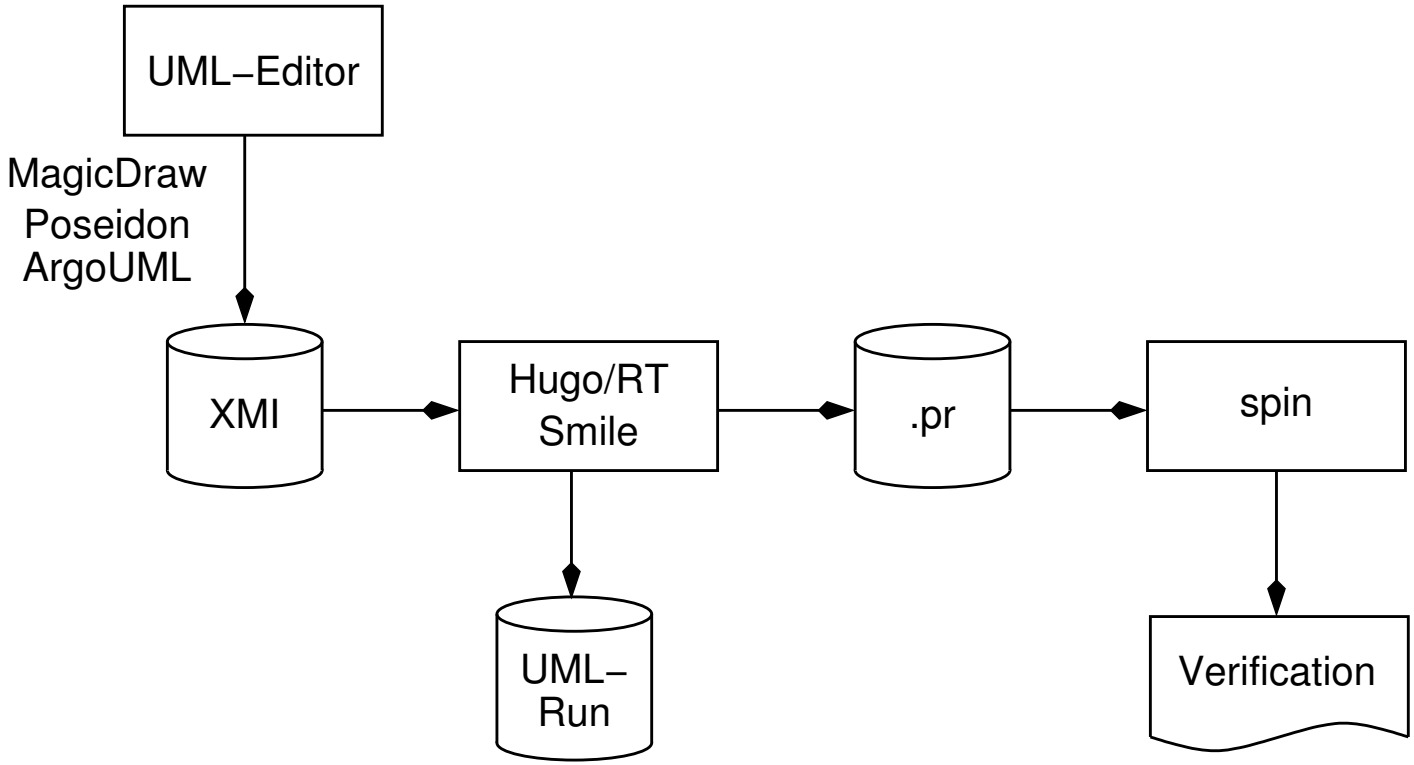
"An ATM should always give money to validated account holders."



System Designer



The Hugo/RT Verification Process



Outline

Previous Work

- Operational and denotational semantics for UML interactions.
  - → Cengarle and Knapp (2004)

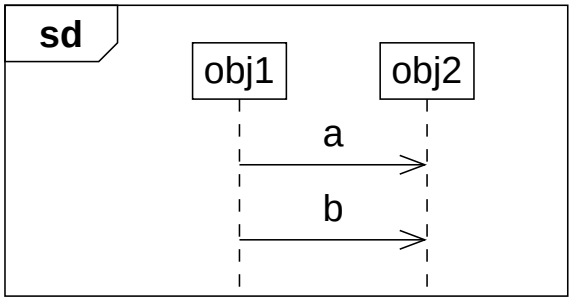
Previous Work

- Operational and denotational semantics for UML interactions.
  - → Cengarle and Knapp (2004)
- Basic unwinding of Life Sequence Charts (LSC).
  - → Brill et al. (2004)

Previous Work

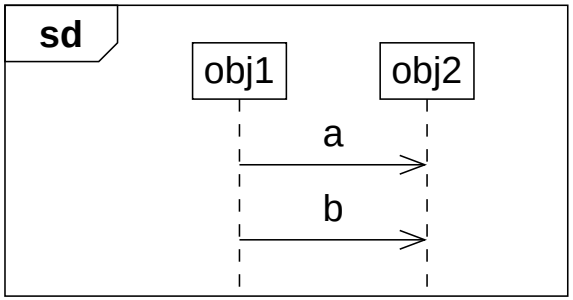
- Operational and denotational semantics for UML interactions.
  → Cengarle and Knapp (2004)
- Basic unwinding of Life Sequence Charts (LSC).
  → Brill et al. (2004)
- Undecidability of the model checking problem for Message Sequence Charts.
  → Alur and Yannakakis (1999)

The Basic Translation Idea



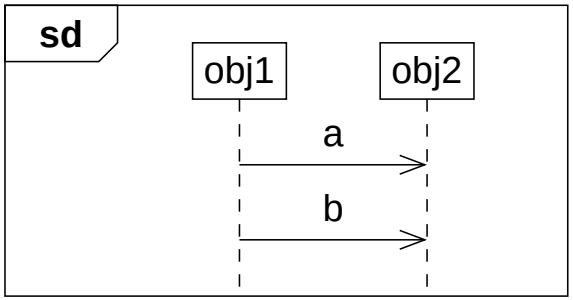
- Basic Interactions consist of events.

The Basic Translation Idea



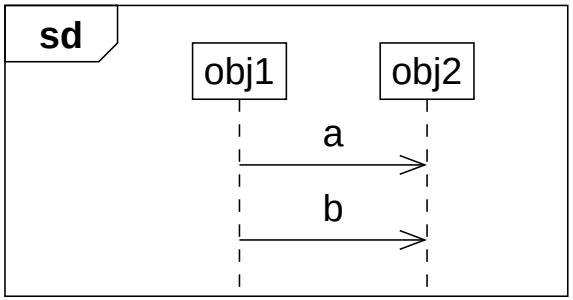
- Basic Interactions consist of events.
- Send- and Receive events are atomic.

The Basic Translation Idea

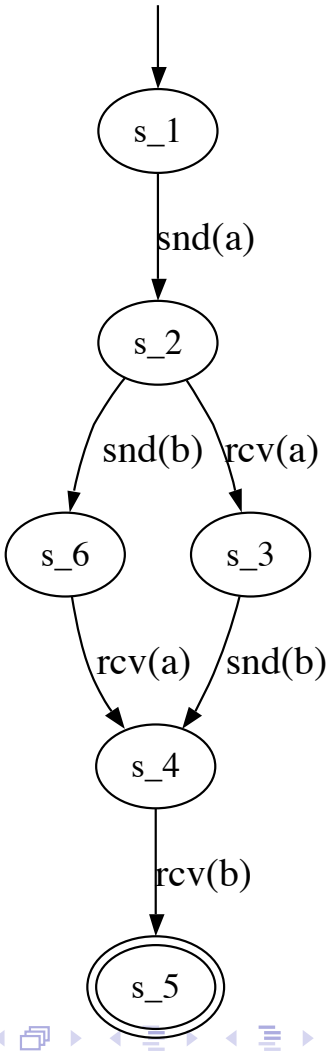


- Basic Interactions consist of events.
- Send- and Receive events are atomic.
- An event history tracks partial order prerequisites.

The Basic Translation Idea



- Basic Interactions consist of events.
- Send- and Receive events are atomic.
- An event history tracks partial order prerequisites.
- The resulting automaton is always deterministic.



Introducing UML 2.0 Operators

- Parallel composition
  - Standard parallel composition of Büchi automata.

Introducing UML 2.0 Operators

- Parallel composition
  - Standard parallel composition of Büchi automata.
- Weak and strict sequencing
  - Strict sequencing attaches the two automata unchanged.
  - Weak sequencing adds partial order constraint to parallel composition.

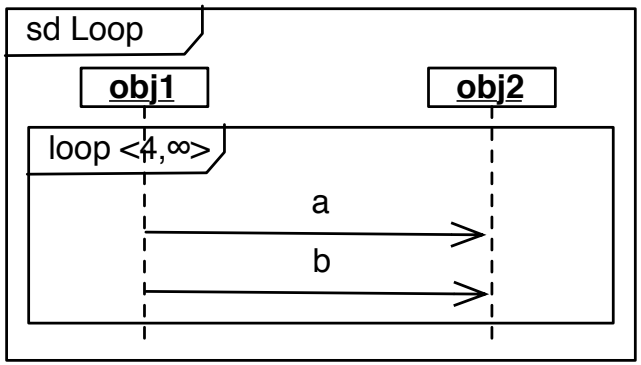
Introducing UML 2.0 Operators

- Parallel composition
  - Standard parallel composition of Büchi automata.
- Weak and strict sequencing
  - Strict sequencing attaches the two automata unchanged.
  - Weak sequencing adds partial order constraint to parallel composition.
- Alternative fragments (conditional branches)
  - Operand automata reachable through guarded transitions from a common new initial state.

Introducing UML 2.0 Operators

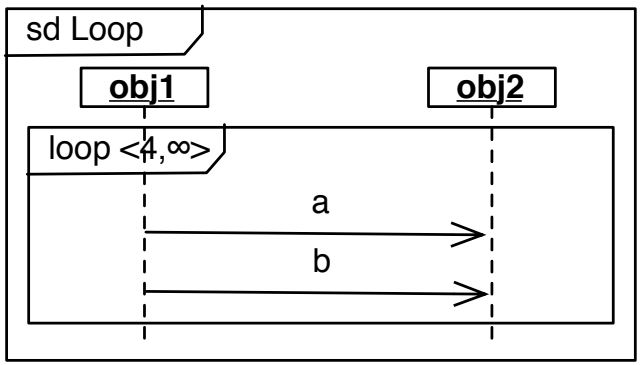
- Parallel composition
  - Standard parallel composition of Büchi automata.
- Weak and strict sequencing
  - Strict sequencing attaches the two automata unchanged.
  - Weak sequencing adds partial order constraint to parallel composition.
- Alternative fragments (conditional branches)
  - Operand automata reachable through guarded transitions from a common new initial state.
- Ignoring unimportant events
  - Adds transitions that “swallow” specified events.

Loop Translation



Weak sequencing can “wrap around” before the basic interaction is complete.

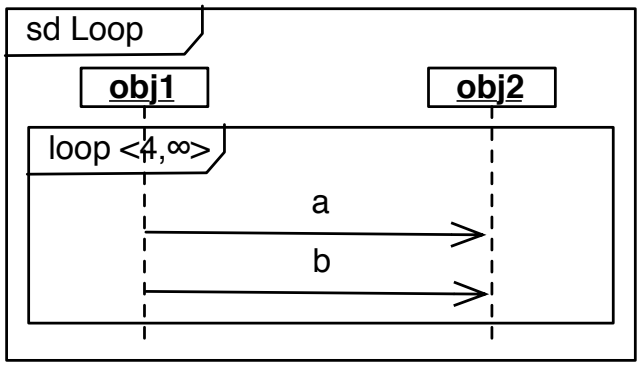
Loop Translation



Weak sequencing can “wrap around” before the basic interaction is complete.

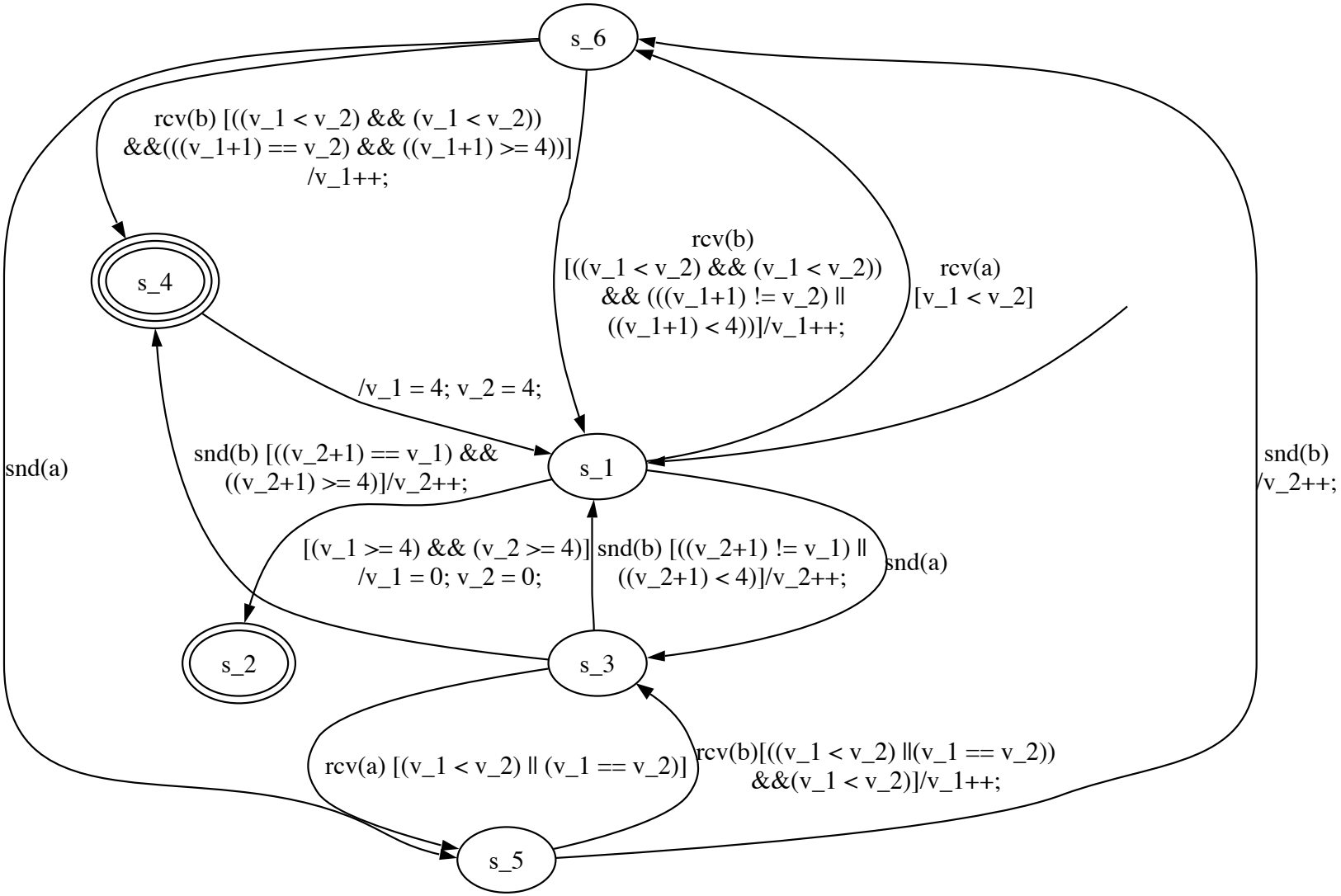
- A simple history is not enough to track progress.
  - Counter variables track the number of iterations on each lifeline.
  - Guards and actions augment transition annotations.

Loop Translation

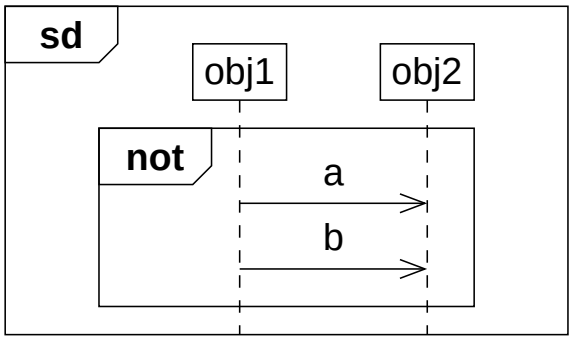


Weak sequencing can “wrap around” before the basic interaction is complete.

- A simple history is not enough to track progress.
  - Counter variables track the number of iterations on each lifeline.
  - Guards and actions augment transition annotations.
- Weak sequencing loops create possibly infinite state spaces.

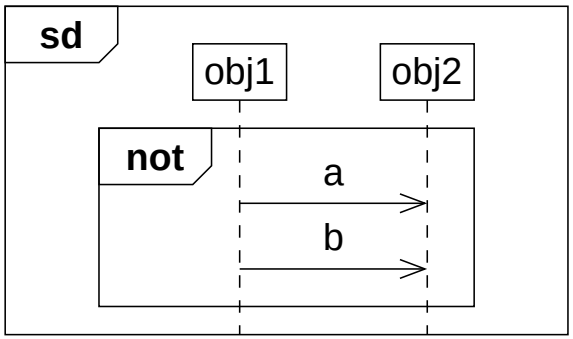

s_6
rcv(b) [((v_1 < v_2) && (v_1 < v_2))
&&(((v_1+1) == v_2) && ((v_1+1) >= 4))]
/v_1++;
s_4
rcv(b)
[((v_1 < v_2) && (v_1 < v_2))
&& (((v_1+1) != v_2) ||
((v_1+1) < 4))]/v_1++;
rcv(a)
[v_1 < v_2]
/v_1 = 4; v_2 = 4;
snd(a)
snd(b) [((v_2+1) == v_1) &&
((v_2+1) >= 4)]/v_2++;
s_1
snd(b)
/v_2++;
[(v_1 >= 4) && (v_2 >= 4)]
/v_1 = 0; v_2 = 0;
snd(b) [((v_2+1) != v_1) ||
((v_2+1) < 4)]/v_2++;
snd(a)
s_2
s_3
rcv(a) [(v_1 < v_2) || (v_1 == v_2)]
rcv(b)[((v_1 < v_2) ||(v_1 == v_2))
&&(v_1 < v_2)]/v_1++;
s_5

Translating Negation



Replace UML's `neg` with binary logic negation, i.e, the operator `not` accepts all those traces that are invalid for its operand.

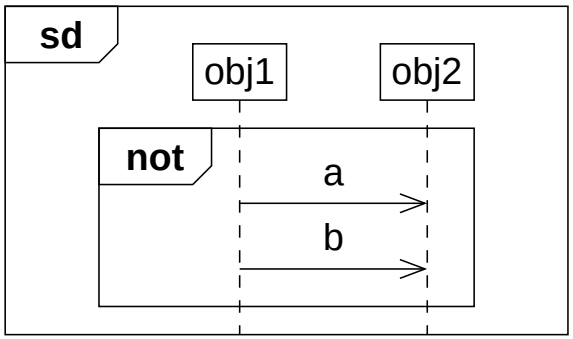
Translating Negation



- The accepting state of the operand becomes a normal state.

Replace UML's `neg` with binary logic negation, i.e, the operator `not` accepts all those traces that are invalid for its operand.

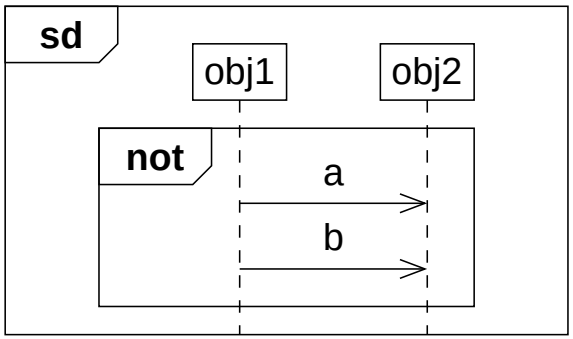
Translating Negation



- The accepting state of the operand becomes a normal state.
- All other states become accepting.

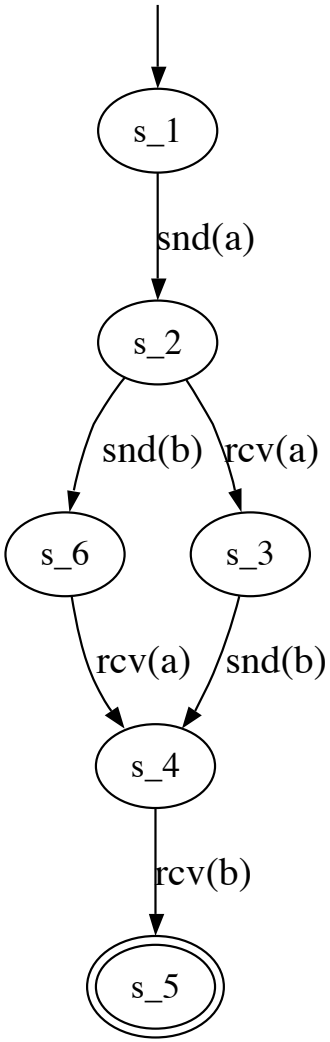
Replace UML's `neg` with binary logic negation, i.e, the operator `not` accepts all those traces that are invalid for its operand.

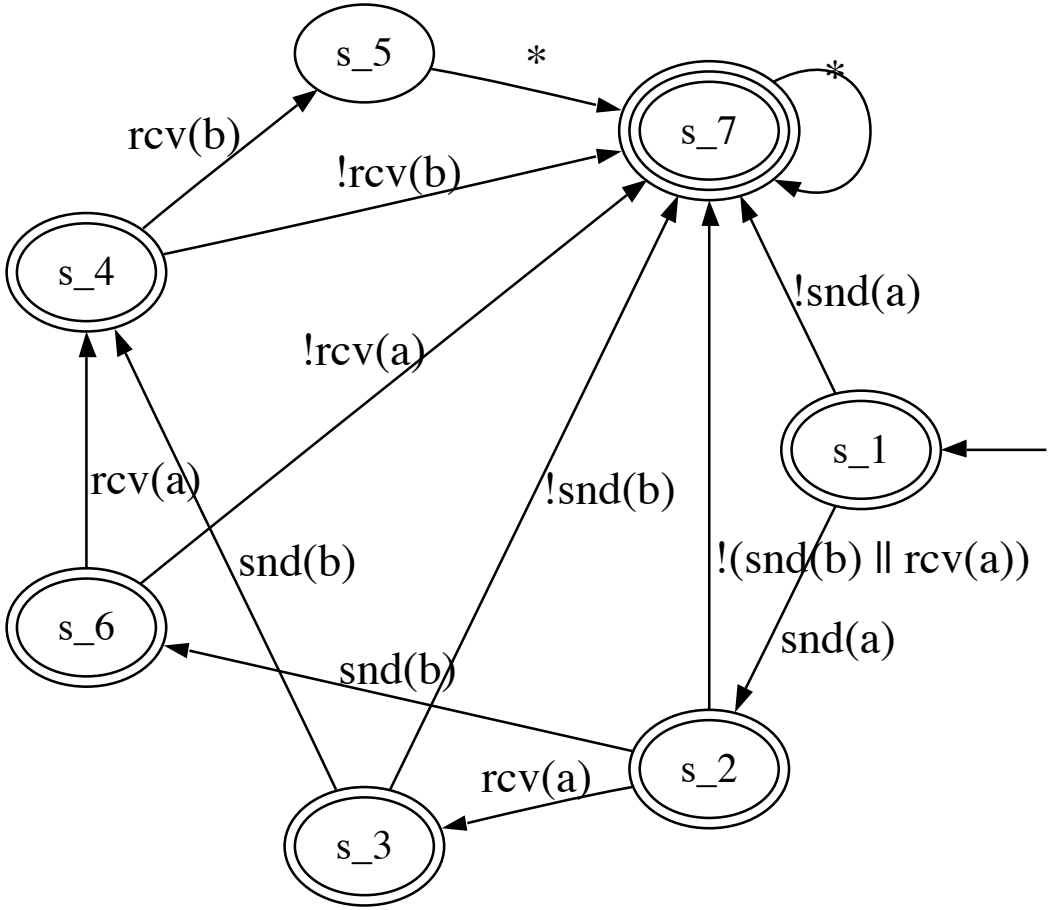
Translating Negation



Replace UML's `neg` with binary logic negation, i.e, the operator `not` accepts all those traces that are invalid for its operand.

- The accepting state of the operand becomes a normal state.
- All other states become accepting.
- Introduce a recurrent state that accepts all traces not contained in the original automaton.


s_1
snd(a)
s_2
snd(b)
rcv(a)
s_6
s_3
rcv(a)
snd(b)
s_4
rcv(b)
s_5


s_5
*
s_7
*
rcv(b)
!rcv(b)
s_4
!snd(a)
!rcv(a)
s_1
rcv(a)
!snd(b)
snd(b)
!(snd(b) || rcv(a))
s_6
snd(a)
snd(b)
rcv(a)
s_2
s_3

Translation Overview

- Basic automata are always deterministic.
- Most UML 2.0 operators introduce non-determinism.
- Negation attempts to be intuitive, binary logic negation.
- The automata are finite, but the represented state-space is not.
- Model checking automata derived from loop operators is in general undecidable.

Implementation

- Hugo/RT is 100% pure Java.
- Supports translation to PROMELA and (partially) UPPAAL.
- Input from XMI 2.0 or the proprietary UTE format.
- Available at `www.pst.ifi.lmu.de/projekte/hugo`

Summary

- We presented a detailed semantics and concrete translation from UML interactions to automata.

Summary

- We presented a detailed semantics and concrete translation from UML interactions to automata.
- Enable consistency and safety checks on specifications.

Summary

- We presented a detailed semantics and concrete translation from UML interactions to automata.
- Enable consistency and safety checks on specifications.

- Outlook

Summary

- We presented a detailed semantics and concrete translation from UML interactions to automata.
- Enable consistency and safety checks on specifications.

- Outlook
  - Include the remaining operators from UML 2.0.

Summary

- We presented a detailed semantics and concrete translation from UML interactions to automata.
- Enable consistency and safety checks on specifications.

- Outlook
  - Include the remaining operators from UML 2.0.
  - Add support for timing constraints.