iUML-B
State Machine Diagrams

Dr Colin Snook

Motivation

Provide a more approachable interface for newcomers to Event-B

Provide diagrams to help visualise models

Provide extra modelling features to Event-B

- Sequencing of Events
- Lifting of Behaviour to a set of instances (O-O)

N.b. not trying to formalise UML

What is iUML-B?

A Graphical front-end for Event-B

- Plug-in for Rodin

Not UML …

- Has its own meta-model (abstract syntax)
- Semantics inherited from translation to Event-B

… but it has some similarities with UML

- Class Diagrams *Coming soon!*
- State Machine Diagrams

Translator generates Event-B automatically

- Into the same machine (generated is read only)
- Can also write standard Event-B in the same machine + events

What are the benefits?

Visualisation

- Helps understanding
- communication

Faster modelling

- One drawing node = several lines of B
- Extra information inferred from position (containment) of elements
- Experiment with different abstractions

finding useful abstractions is hard

Provides structuring constructs

- Hierarchical state-machines

  Event-B has no *event sequencing* mechanism
- Class Lifted state-machines

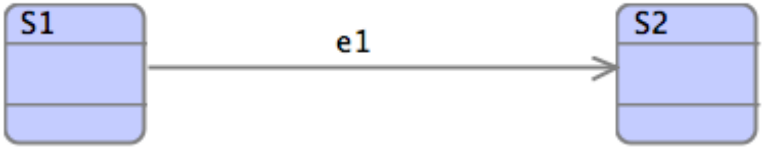
  Event-B has no *lifting* mechanism

State Machines

State machines provide a way to model behaviour (transitions)

Constrained by some data (source state)

The transition's behaviour is to change the data (to target state)

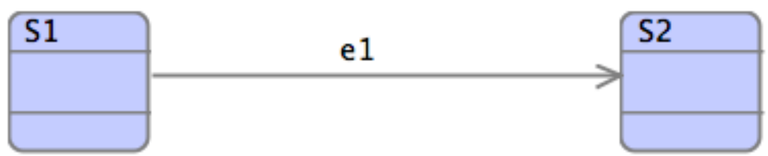


Transition e1 can only fire when the state is S1

when e1 fires it changes the state to S2

How could we represent this in Event-B?

State Machines to Events



EVENTS

e1 ≙ WHEN <in S1> THEN <becomes S2> END

where, <in S1> and <becomes S2> depend on the data that represents state

State machine as a type

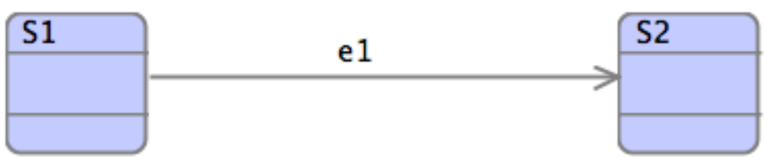
We could treat the whole statemachine as an enumerated type. The current state is given by a variable of that type. (Called Enumeration translation in iUML-B).

VARIABLES

sm ∈ sm_STATES

SETS

sm_STATES = {S1,S2}



EVENTS

e1 ≙ WHEN sm = S1 THEN sm := S2 END

State machine collection of variables

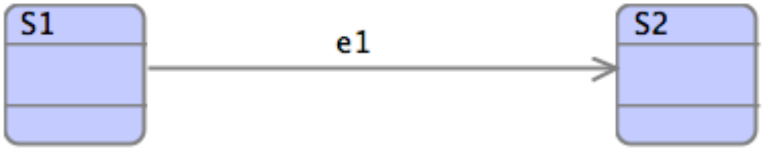
Or we could treat each state as a separate variable. (Called Variables translation in iUML-B)

VARIABLES

S1 ∈ BOOL

S2 ∈ BOOL

where, one of S1, S2 is TRUE at any moment



EVENTS

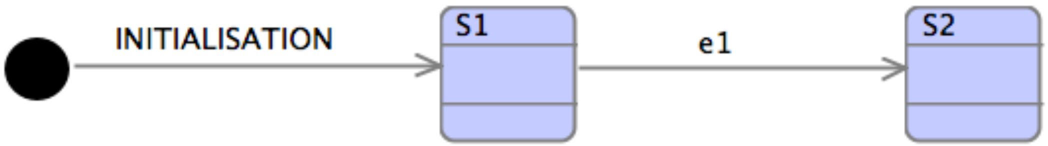
e1 ≙ WHEN S1 = TRUE

THEN S1 := FALSE

S2 := TRUE

END

Initial transition



Enumeration translation

INITIALISATION
```
sm := S1
```

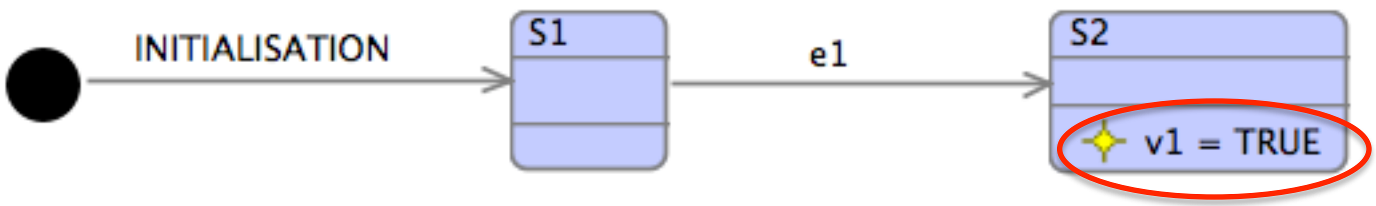
or

Variables translation

INITIALISATION
```
S1 := TRUE
S2 := FALSE
```

State Invariant

Something that must be true whenever the system is in that state.



Enumeration translation

INVARIANTS

$(sm=S2) \Rightarrow (v1=TRUE)$

or

Variables translation

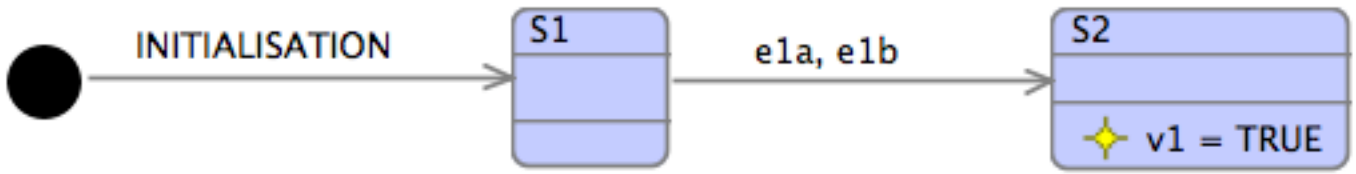
INVARIANTS

$(S2=TRUE) \Rightarrow (v1=TRUE)$

Transition Elaboration

Transitions ‘elaborate’ (i.e. contribute to) event(s) in the Machine.

So you can give them parameters, guards and actions etc. in the Machine




```
e1a  ≙
STATUS
 ordinary
ANY
 p
WHERE
 isin_S1   :   S1 = TRUE
 grd1   :   p > 5
THEN
 leave_S1   :   S1 ≔ FALSE
 enter_S2   :   S2 ≔ TRUE
END
```

```
e1b  ≙
STATUS
 ordinary
ANY
 p
WHERE
 isin_S1   :   S1 = TRUE
 grd1   :   p ≤ 5
THEN
 enter_S2   :   S2 ≔ TRUE
 leave_S1   :   S1 ≔ FALSE
END
```

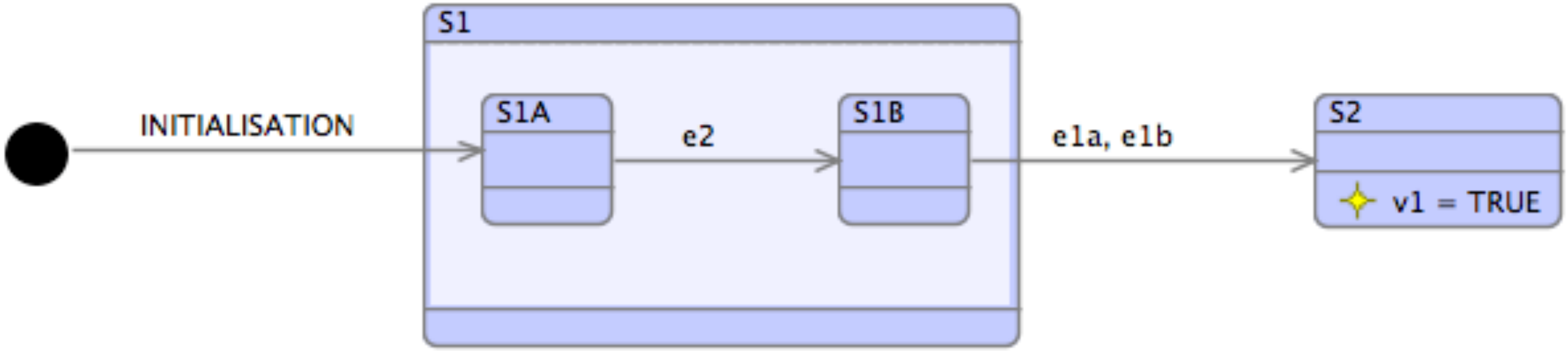
Nested Statemachines

Statemachines can be nested inside states

a) so that we can put an invariant on the superstate

b) so that a transition can leave from any substate

c) adding detail in a refinement



REFINEMENT of State-machines:

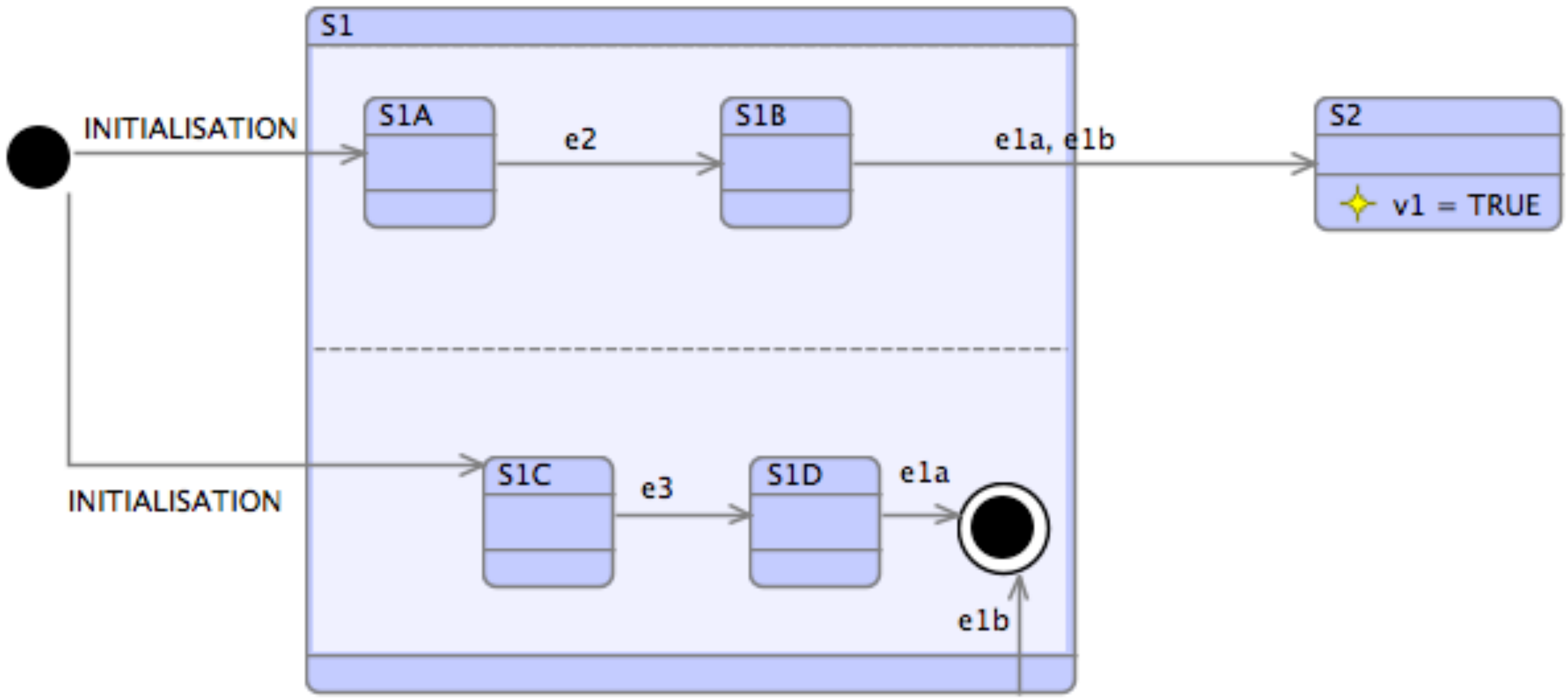
We often add nested statemachines in refinements.

Can also split events into cases (e.g. e1 to e1a,e1b) and add invariants to states

BUT must NOT add states to an existing statemachine as this would break the type/partition invariants.

Parallel Nested Statemachines

Several Statemachines can be nested inside states



Transition Properties

Transitions can own event properties:

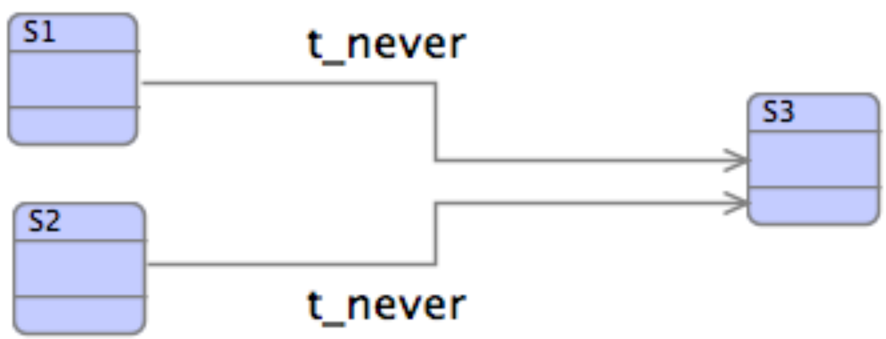
parameters, witnesses, guards and actions

Transition properties are copied into each elaborated event when the statemachine is generated.

Why? :

a. You can do all the modelling in the diagram, no need to switch to the Event-B editor
b. If several events are elaborated you only write the guards/actions etc. once

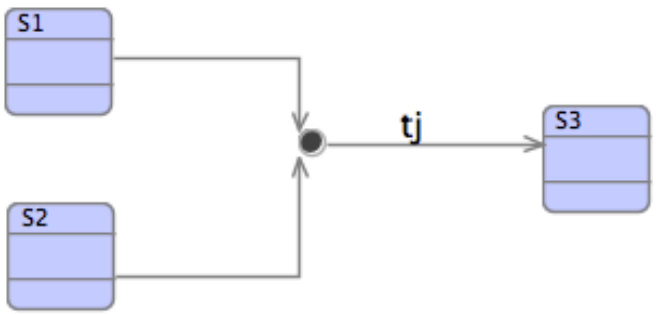
Warning: Make sure transitions are possible!!




```
o   t_never:     not extended ordinary ›
    WHERE
    o   isin_S2:     S2 = TRUE not theorem ›
    o   isin_S1:     S1 = TRUE not theorem ›
    THEN
    o   leave_S2:    S2 ≔ FALSE ›
    o   leave_S1:    S1 ≔ FALSE ›
    o   enter_S3:    S3 ≔ TRUE ›
    END
```

Junctions

For transitions that are enabled in several states



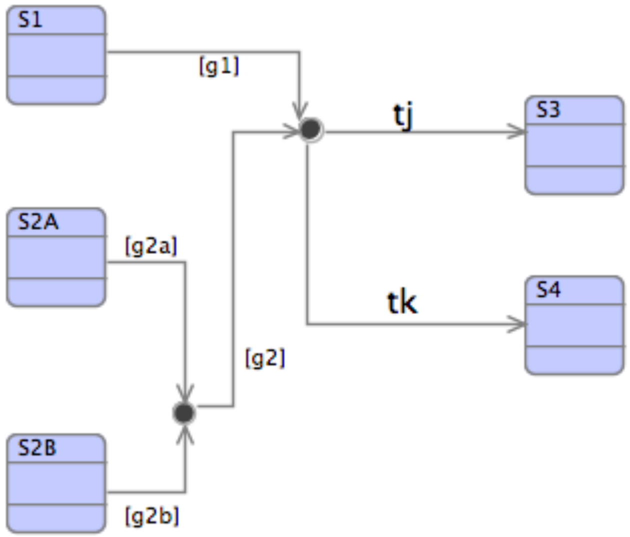

```
o   tj:  not extended ordinary ›
    WHERE
    o   isin_S1_or_isin_S2: (S1 = TRUE ∨ S2 = TRUE)
    THEN
    o   leave_S2:   S2 ≔ FALSE ›
    o   leave_S1:   S1 ≔ FALSE ›
    o   enter_S3:   S3 ≔ TRUE ›
    END
```

Rules:

1. Only the final segment elaborates an event
2. Other segments do not own actions (but can have guards)

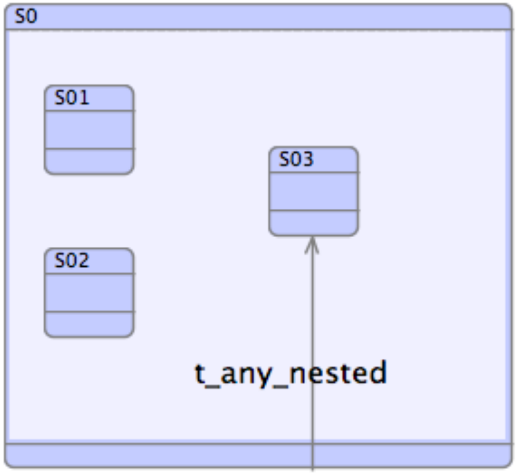
Junctions (cont.)

Can be combined



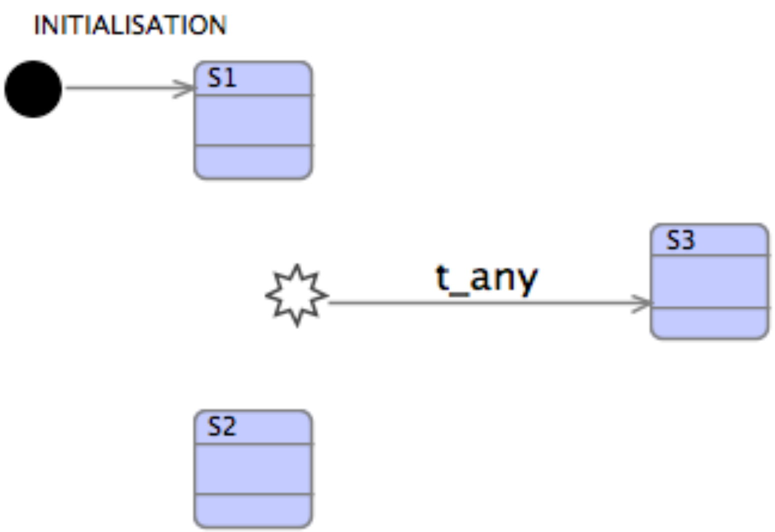

```
o   tj:  not extended ordinary ›
    WHERE
    o   isin_S1_or_isin_S2A_or_isin_S2B:    ((S1 = TRUE ∧ g1) ∨ (((S2A = TRUE ∧ g2a) ∨ (S2B = TRUE ∧ g2b)) ∧ g2))
    THEN
    o   leave_S2B:   S2B ≔ FALSE ›
    o   leave_S2A:   S2A ≔ FALSE ›
    o   leave_S1:    S1 ≔ FALSE ›
    o   enter_S3:    S3 ≔ TRUE ›
    END

o   tk:  not extended ordinary ›
    WHERE
    o   isin_S1_or_isin_S2A_or_isin_S2B:    ((S1 = TRUE ∧ g1) ∨ (((S2A = TRUE ∧ g2a) ∨ (S2B = TRUE ∧ g2b)) ∧ g2))
```

From any substate



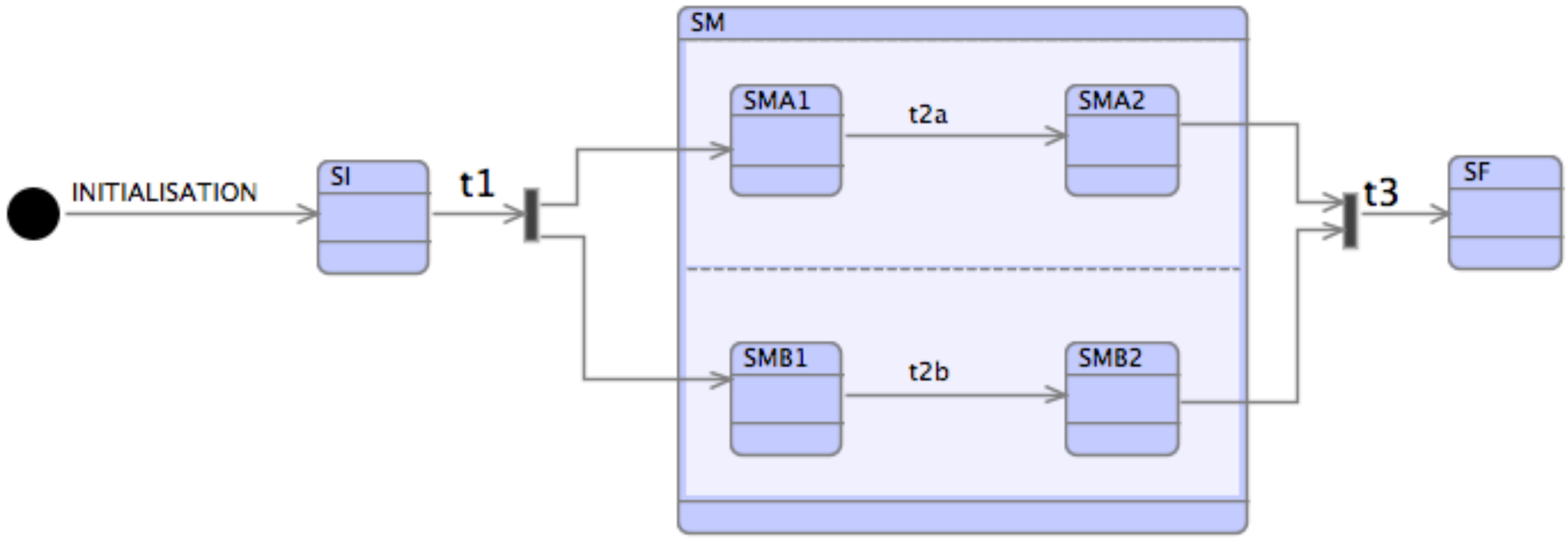

```
o   t_any_nested:    not extended ordinary ›
    WHERE
    o   isin_S0:     S0 = TRUE not theorem ›
    THEN
    o   leave_S02:   S02 ≔ FALSE ›
    o   leave_S01:   S01 ≔ FALSE ›
    o   enter_S03:   S03 ≔ TRUE ›
    END
```

From ANY at top level




```
o   t_any:   not extended ordinary ›
    THEN
    o   leave_S2:   S2 ≔ FALSE ›
    o   leave_S1:   S1 ≔ FALSE ›
    o   enter_S3:   S3 ≔ TRUE ›
    END
```

Forks and Joins




```
o   t1:  not extended ordinary ›
    WHERE
    o   isin_SI:     SI = TRUE not theorem ›
    THEN
    o   leave_SI:    SI ≔ FALSE ›
    o   enter_SMB1:  SMB1 ≔ TRUE ›
    o   enter_SM:    SM ≔ TRUE ›
    o   enter_SMA1:  SMA1 ≔ TRUE ›
    END
```

```
o   t3:  not extended ordinary ›
    WHERE
    o   isin_SMA2:  SMA2 = TRUE not theorem ›
    o   isin_SMB2:  SMB2 = TRUE not theorem ›
    THEN
    o   leave_SMA2: SMA2 ≔ FALSE ›
    o   leave_SM:   SM ≔ FALSE ›
    o   leave_SMB2: SMB2 ≔ FALSE ›
    o   enter_SF:   SF ≔ TRUE ›
    END
```

Example – Factory Machine

Machine

1) A factory machine can be switched on and off.
2) When it is on it can then be started and becomes active.
3) When it is active it can run repeatedly until it is stopped.

Shield

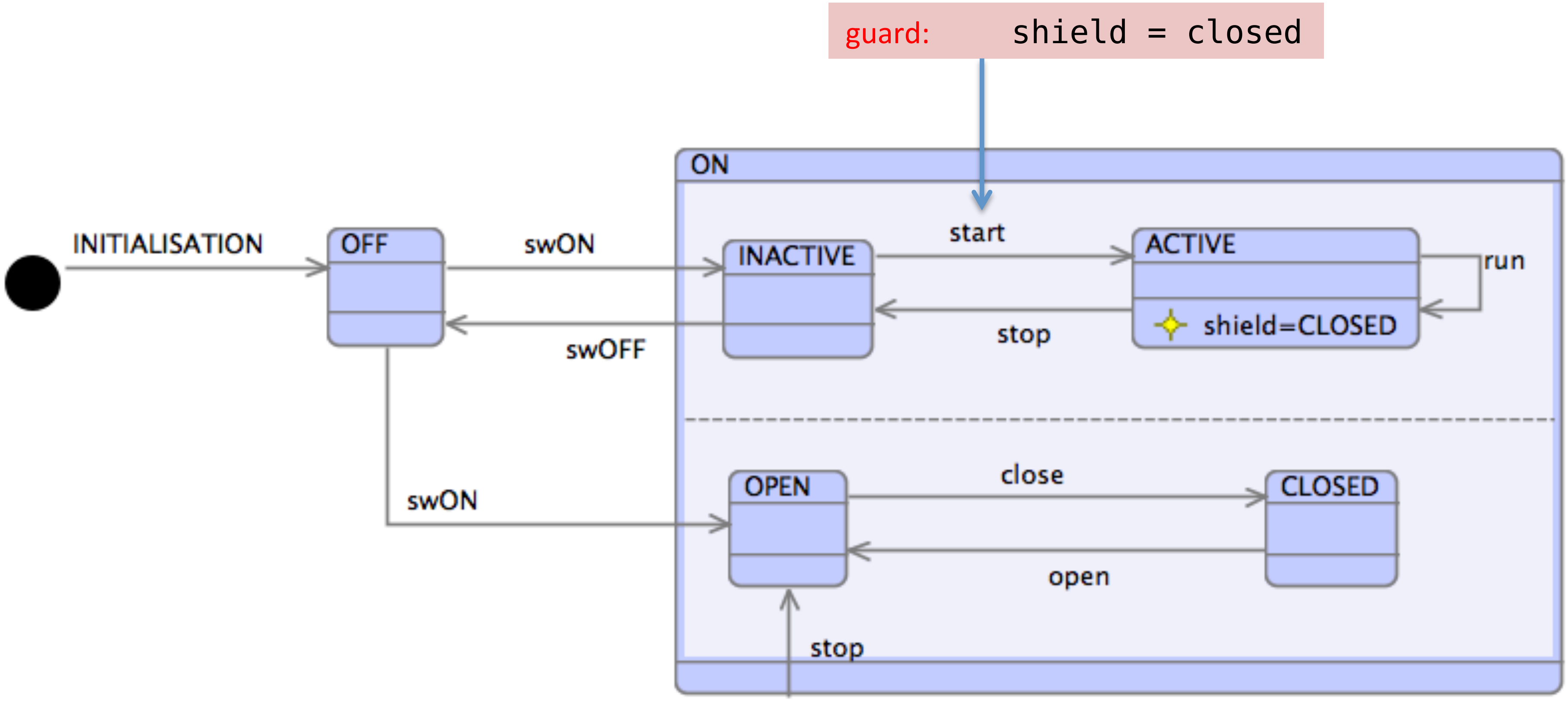
4) A separately controlled safety shield can be opened and closed when the machine is on.
5) The shield is opened automatically when the machine is stopped.

Invariant

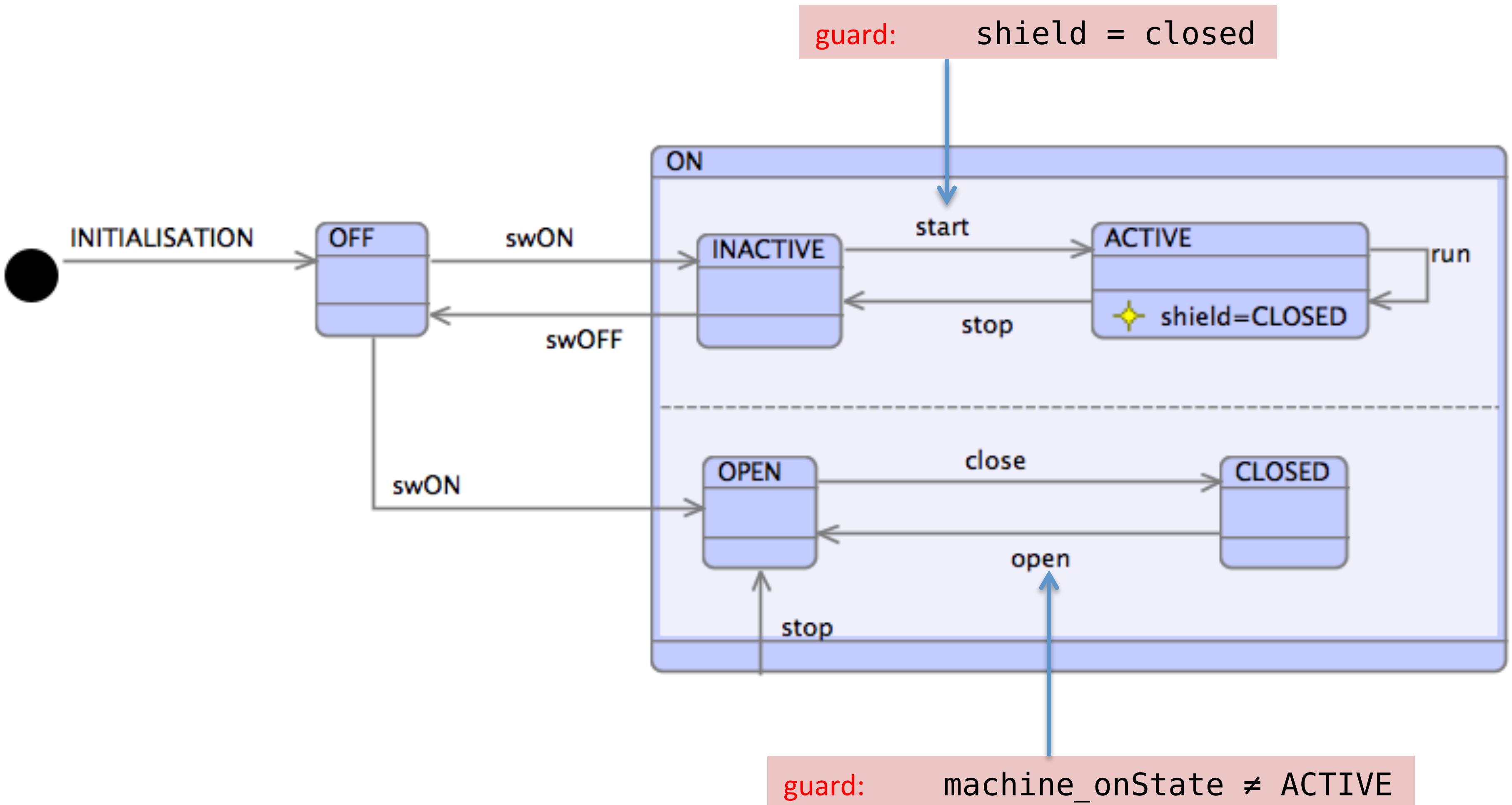
6) Safety Requirement:

The machine should never be in the active state (where runs can occur) with the shield in the open position.

Example – Factory Machine

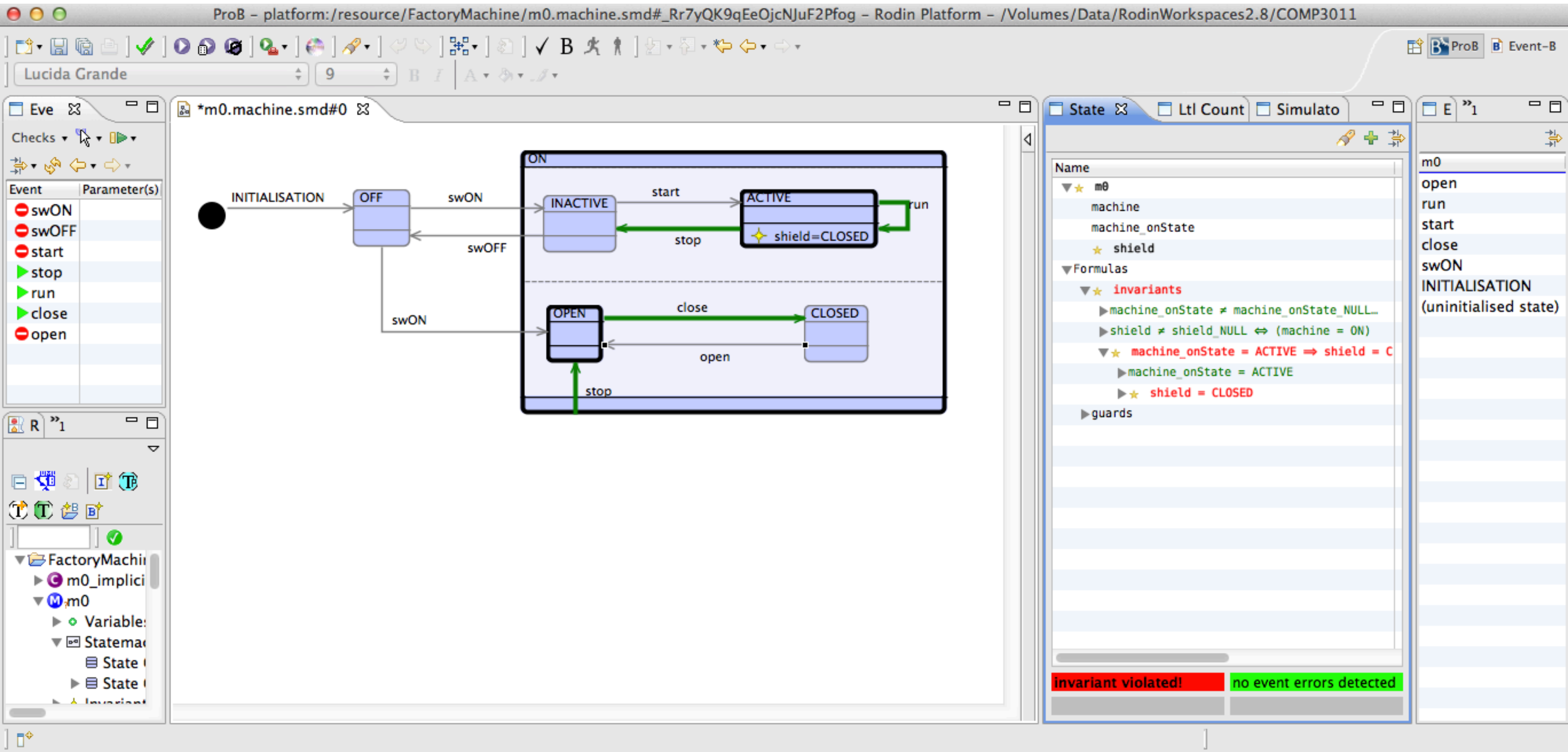


Example – Factory Machine



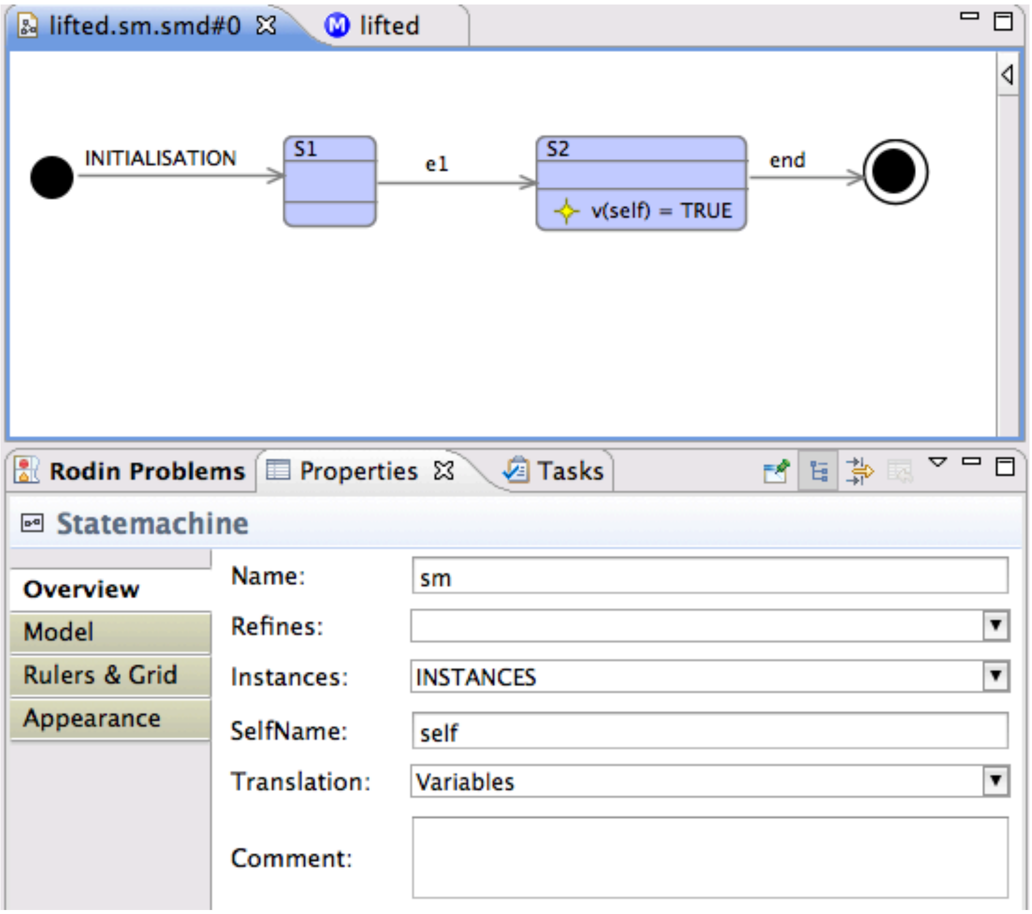
omitted for model checking demo on next page

State-machine Animation showing invariant violation



Lifted (O-O) State-machines

Statemachines can be 'lifted' to a set of instances in an O-O way
- Effectively, each instance has a ''copy'' of the statemachine
- Need to be able to define 'self' name in case of transition synchronisation



Lifted (O-O) State-machines - Enumeration Translation

VARIABLES/INVARIANTS

$sm \in INSTANCES \longrightarrow sm\_STATES$

SETS

$sm\_STATES = \{S1, S2\}$

EVENTS

INITIALISATION $\triangleq$

$sm := INSTANCES \times \{S1\}$

e1(self) $\triangleq$

WHERE $sm(self) = S1$

THEN $sm(self) := S2$

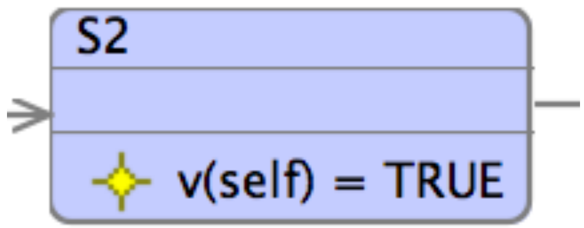
END

Lifted (O-O) State-machines - Variables Translation

```
VARIABLES/INVARIANTS
   S1   ⊆  (INSTANCES)
   S2   ⊆  (INSTANCES)
   partition((S1 ∪ S2), S1, S2)

EVENTS
   INITIALISATION ≙
        S1 :=  INSTANCES
        S2 :=  ∅

   e1(self) ≙
        WHERE  self ∈ S1
        THEN   S1 := S1 \ {self}
               S2 := S2 ∪ {self}
        END
```

Lifted (O-O) State-machines – Invariants (Variables Translation)

Something that must be true for the instance whenever an instance of the class is in that state.



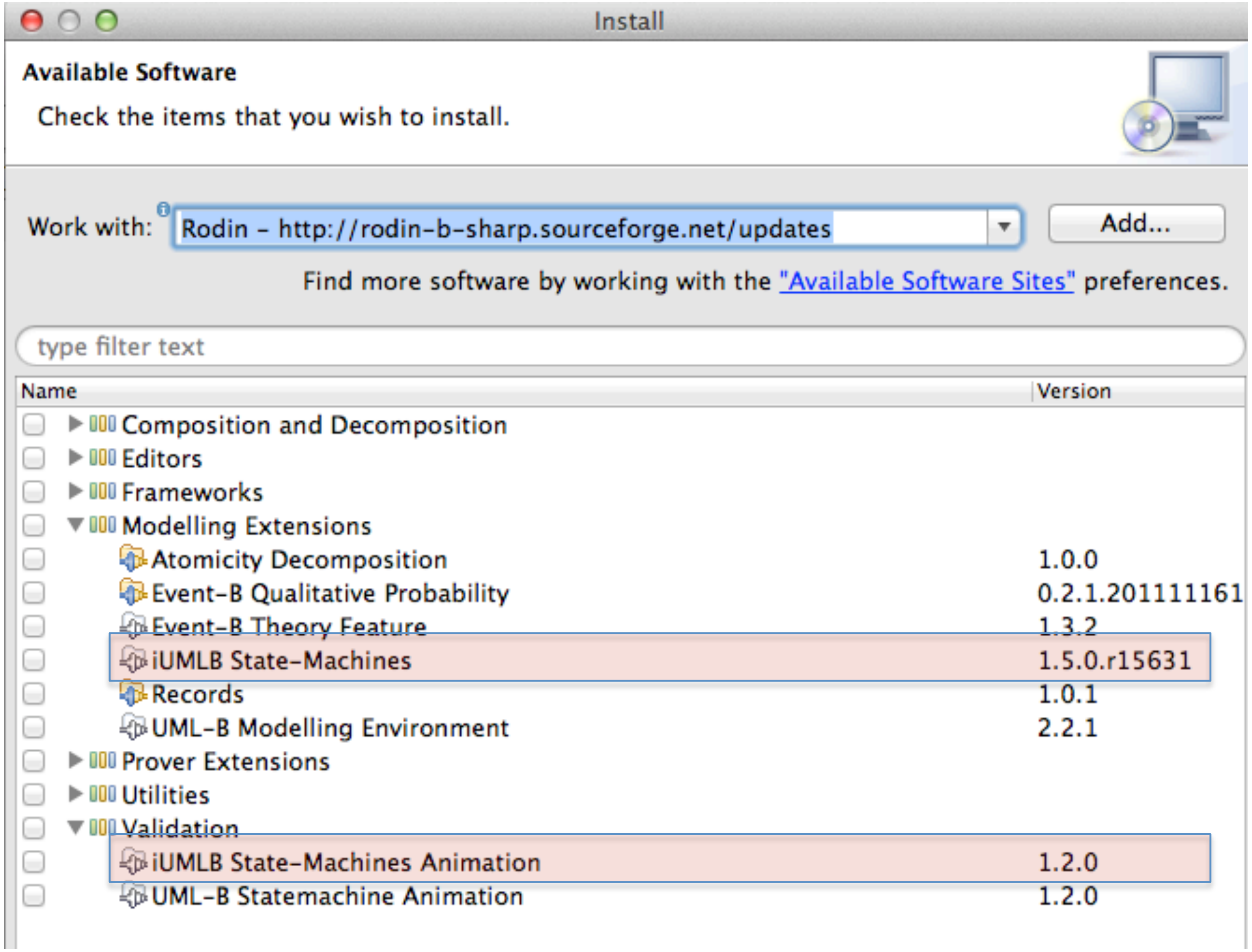
Translation:

$$\forall self \cdot \ (sm(self) = S2) \Rightarrow (v(self) = TRUE))$$

Installation

IMPORTANT: An extra update site has to be added to make some dependencies accessible. Use the ADD button below to add the following update site before installing iUML-B :-

http://download.eclipse.org/modeling/gmp/gmf-tooling/updates/releases/



Summary

Statemachines for modelling behaviour

- Nested statemachines in states
- Invariants in states
- Transitions elaborate events …
- … to control the sequence of event firing (a control flow)

Choice of 2 translations

Can be ‘lifted’ to instances

Can be animated and model checked

- (front-end for Pro-B)