

# Modelling in UML-B

November, 2010

Colin Snook University of Southampton

# What is UML-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
- ▶ Constrain/Action Language inherited from Event-B (with optional extras)

... but has some similarities with UML

- ▶ Class Diagrams
- ▶ State Machine Diagrams

Translator generates Event-B

Errors indicated on diagram

# What are the benefits?

## Visualisation

- ▶ Helps understanding
- ▶ communication

## Faster modelling (allows you to experiment)

- ▶ One drawing node = several lines of B
- ▶ Extra information inferred from position of elements  
(e.g. if contained in a class or statemachine)

## Provides structuring constructs

- ▶ Class
- ▶ Hierarchical state-machines

# Getting Started

## Install UML-B using the Rodin update site

- ▶ Help – Install, select the main Rodin update site, wait for it to retrieve the categories, select UML-B Modelling Environment under Modelling Extensions.

## UML-B Perspective

### UML-B New Project Wizard

- ▶ Opens a project diagram for you
- ▶ Add machines and contexts
- ▶ Double click on a machine to open a class diagram
- ▶ or on a context to open a context diagram



Look for this icon

### UML-B Menu

- ▶ Enable automatic translation
- ▶ Disabled by default (Recommended for larger models)

### UML-B toolbar button

- ▶ Save and translate

# Class-oriented problems

In Event-B models, often find a pattern

- ▶ Set  $I$  (or could be constant or variable)
- ▶ Variables  $v \in I \rightarrow T$
- ▶ Events  $e(i,..)$  when  $i \in I, \dots$  then  $v(i) := x$

$I$  is a set of instances of a class

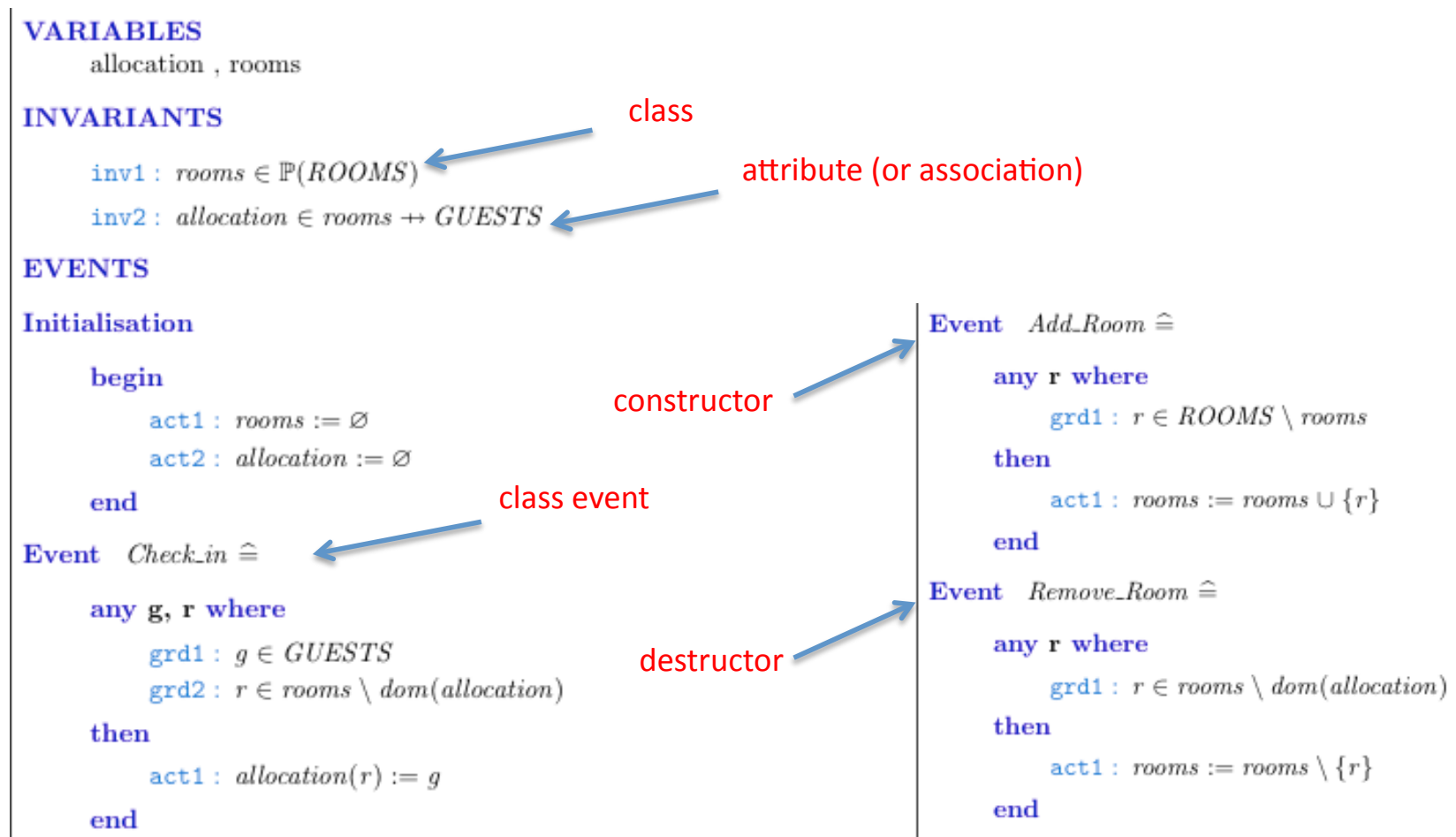
$v$  is a set of values, one for each instance (a class attribute)

$e$  is a 'family' of identical events to assign values to  $v$  (a class event)

I.e. trying to represent class-oriented problems

because Event-B has no lifting mechanism... (c.f. Z promotion, ADT's)

# An Event-B model of a class-oriented problem



# State-oriented problems

In Event-B models, often find a pattern

- ▶ Set  $S = \{S1, S2, S3...\}$
- ▶ Variables  $s \in S$
- ▶ Events  $e1$  when  $s=S1, \dots$  then  $s := S2$   
 $e2$  when  $s=S2, \dots$  then  $s := S3 \dots$

$s$  controls the sequence of events  
an explicit state variable or 'program counter'

I.e. trying to represent process  
because Event-B has no event sequence mechanism...

# UML-B Class (and Context) Diagrams

## Example - modelling with UML-B

In a university degree programme, students are registered on degree courses.  
Students must be enrolled to be registered in a course.  
Courses can be removed from the degree programme.

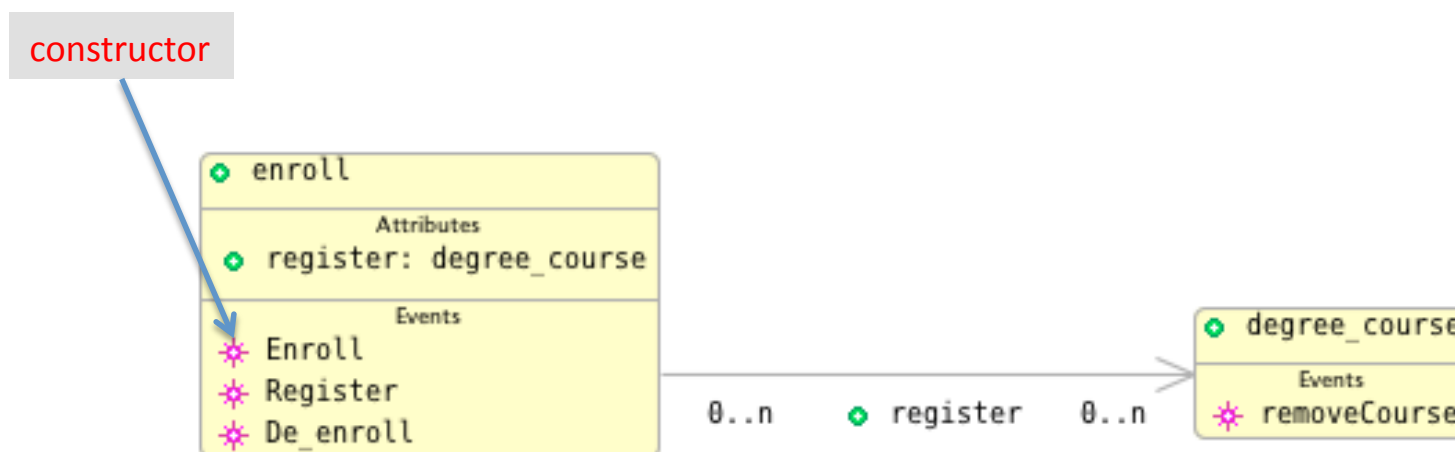
# Example - modelling with UML-B

In a university degree programme, students are registered on degree courses.  
 Students must be enrolled to be registered in a course.  
 Courses can be removed from the degree programme.



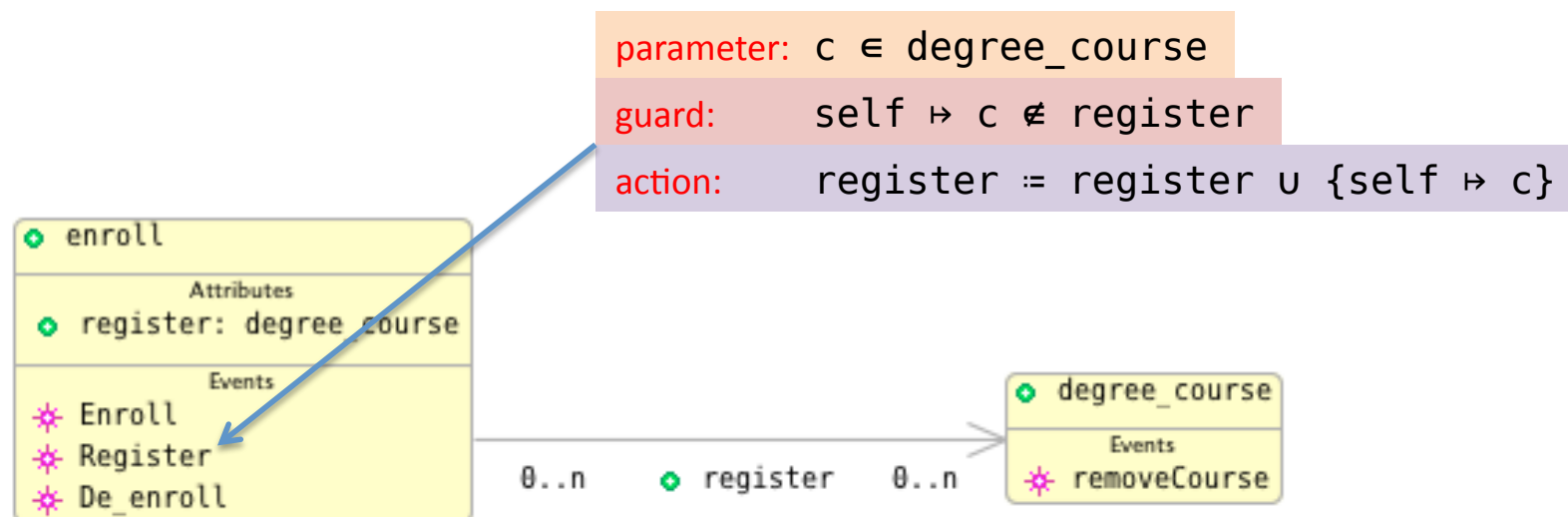
# Example - modelling with UML-B

In a university degree programme, students are registered on degree courses.  
 Students must be enrolled to be registered in a course.  
 Courses can be removed from the degree programme.



# Example - modelling with UML-B

In a university degree programme, students are registered on degree courses.  
 Students must be enrolled to be registered in a course.  
 Courses can be removed from the degree programme.



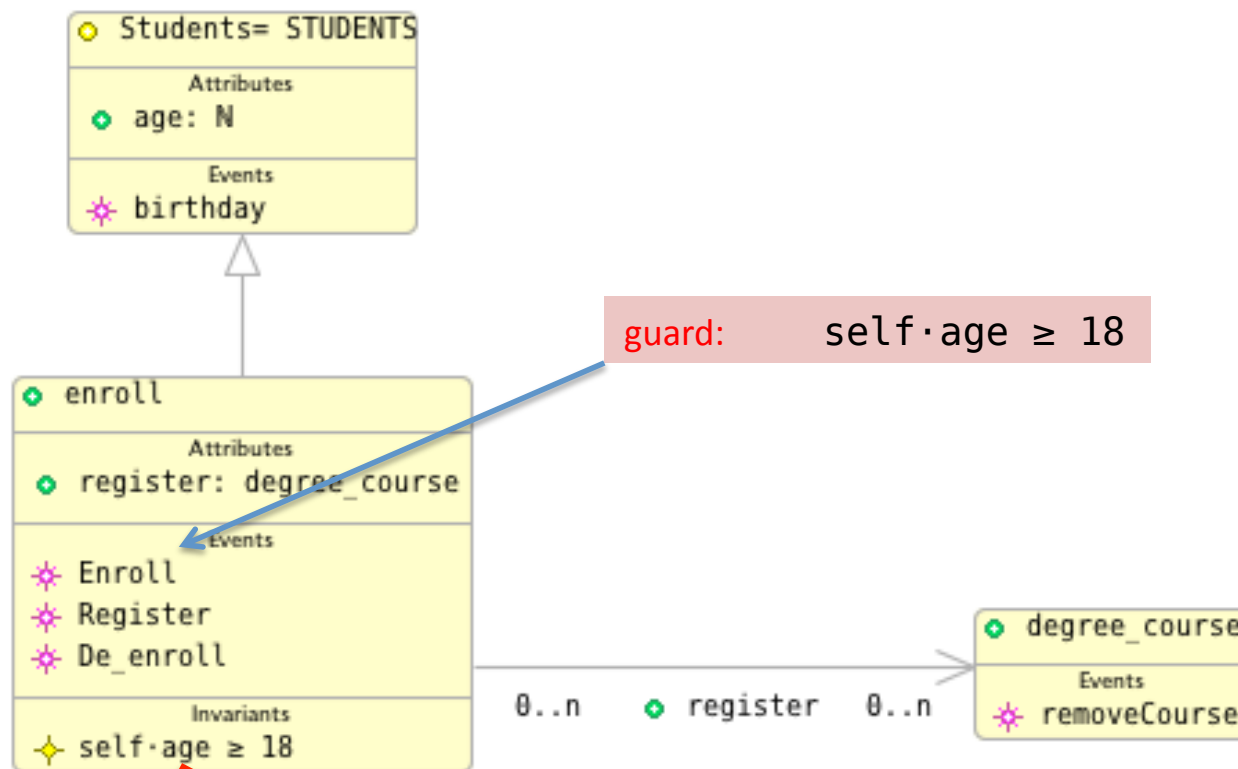
# Example - modelling with UML-B

In a university degree programme, students are registered on degree courses.  
 Students must be enrolled to be registered in a course.  
 Courses can be removed from the degree programme.



# Adding an Invariant

Enrolled students must be 18.



Translation:  $\forall self \cdot ((self \in enroll) \Rightarrow (age(self) \geq 18))$

# UML-B Class Diagrams – Translation rules (part)

UML-B	Event-B
Class (variable instances) Class (fixed instances) Class (variable inst and has super class) Class (fixed inst and has super class)	$\text{Variable} \subseteq \text{Set}$ $\text{Set}$ $\text{Variable} \subseteq \text{SuperClass}$ $\text{Constant} \subseteq \text{SuperClass}$
Attribute (card 0..n - 1..1) Attribute (card 0..n - 0..1) Attribute (card 0..n - 0..n) Etc. (try other cardinalities in UML-B)	$\text{Variable} \in \text{Class} \rightarrow \text{Type}$ $\text{Variable} \in \text{Class} \leftrightarrow \text{Type}$ $\text{Variable} \in \text{Class} \leftarrow \text{Type}$ Etc.
Associations	As Attribute but Type is another class
Class Event	$\text{Event}(\text{self}) \text{ WHEN } \text{self} \in \text{Class} \dots$
Class Constructor	$\text{Event}(\text{self}) \text{ WHEN } \text{self} \in \text{SET} \setminus \text{Class} \dots$
Class Invariant	$\forall \text{self} \cdot ((\text{self} \in \text{Class}) \Rightarrow \text{Class invariant})$

## Example – Event-B produced by UML-B

```

machine m sees m_implicitContext

variables enroll // class instances
           degree_course // class instances
           register // attribute of enroll
           age // attribute of Students

invariants
  @type enroll ∈ P (Students)
  @type degree_course ∈ P (degree_course_SET)
  @type register ∈ enroll ↔ degree_course
  @type age ∈ Students → N
  @Invariant1 ∀self.((self∈enroll)⇒(age(self) ≥ 18))

events
  event INITIALISATION
    then
      @init enroll = ∅
      @init degree_course = ∅
      @init register = ∅
      @init age = Students × {0}
    end

  event Enroll
    any self // constructed instance of class enroll

    where
      @type self ∈ Students \ enroll
      @Guard1 age(self) ≥ 18
    then
      @enroll_constructor enroll = enroll ∪ {self}
    end

  event Register
    any self // contextual instance of class enroll
           c

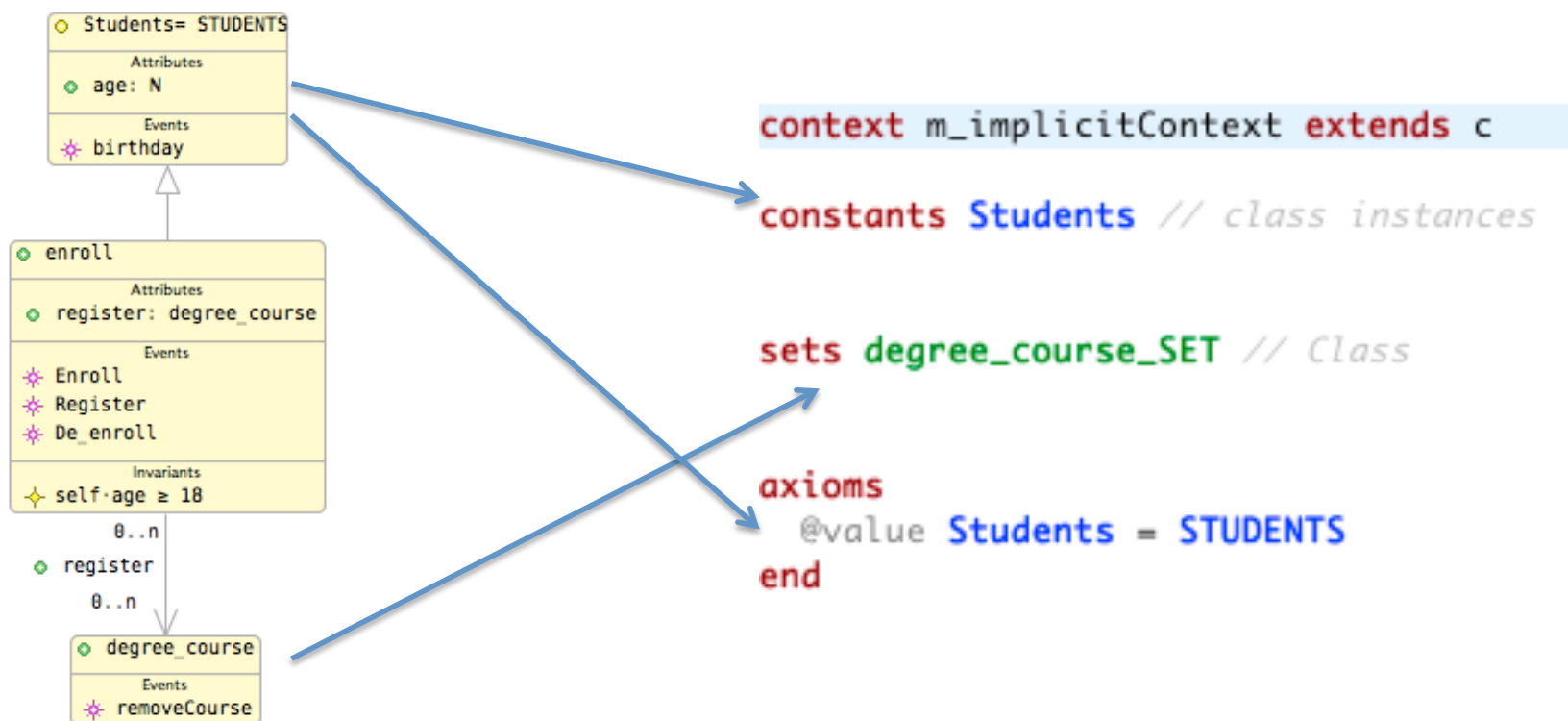
    where
      @type c ∈ degree_course
      @type self ∈ enroll
      @Guard1 self ↦ c ∉ register
    then
      @Action1 register = register ∪ {self ↦ c}
    end

```

# The 'Implicit' Context

Each class diagram creates an *implicit* context

- ▶ Contains the 'basis' of things on the class diagram
- ▶ e.g. a carrier set for the type of class instances



# Context Diagrams

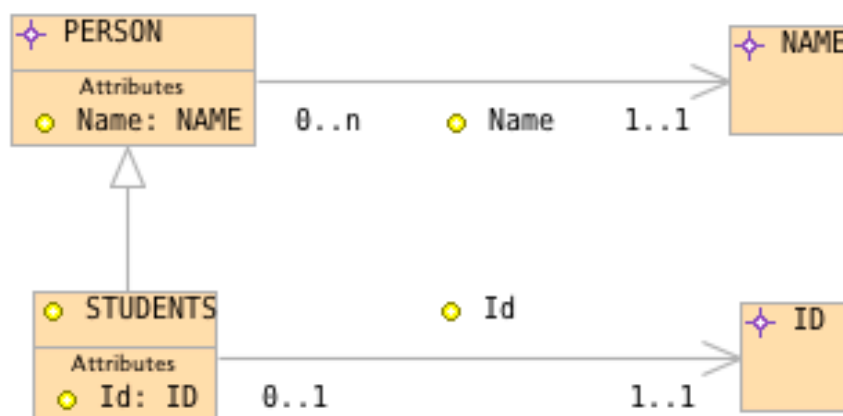
How can we model constants that belong to a class?

in Event-B our machine would **see** a **Context**  
with **sets, constants, axioms**

UML-B takes a similar approach

- ▶ Class Diagram (Machine) sees Context Diagram
- ▶ Similar to a Class Diagram but translates to sets, constants and axioms
- ▶ **ClassType** instead of Class
- ▶ **Constant Attributes/Associations** represent constants
- ▶ **Axioms** instead of Invariants
- ▶ No Events

# A Context Diagram and its translation



**context** c

**constants** STUDENTS // classType instances  
 Id // attribute of STUDENTS  
 Name // attribute of PERSON

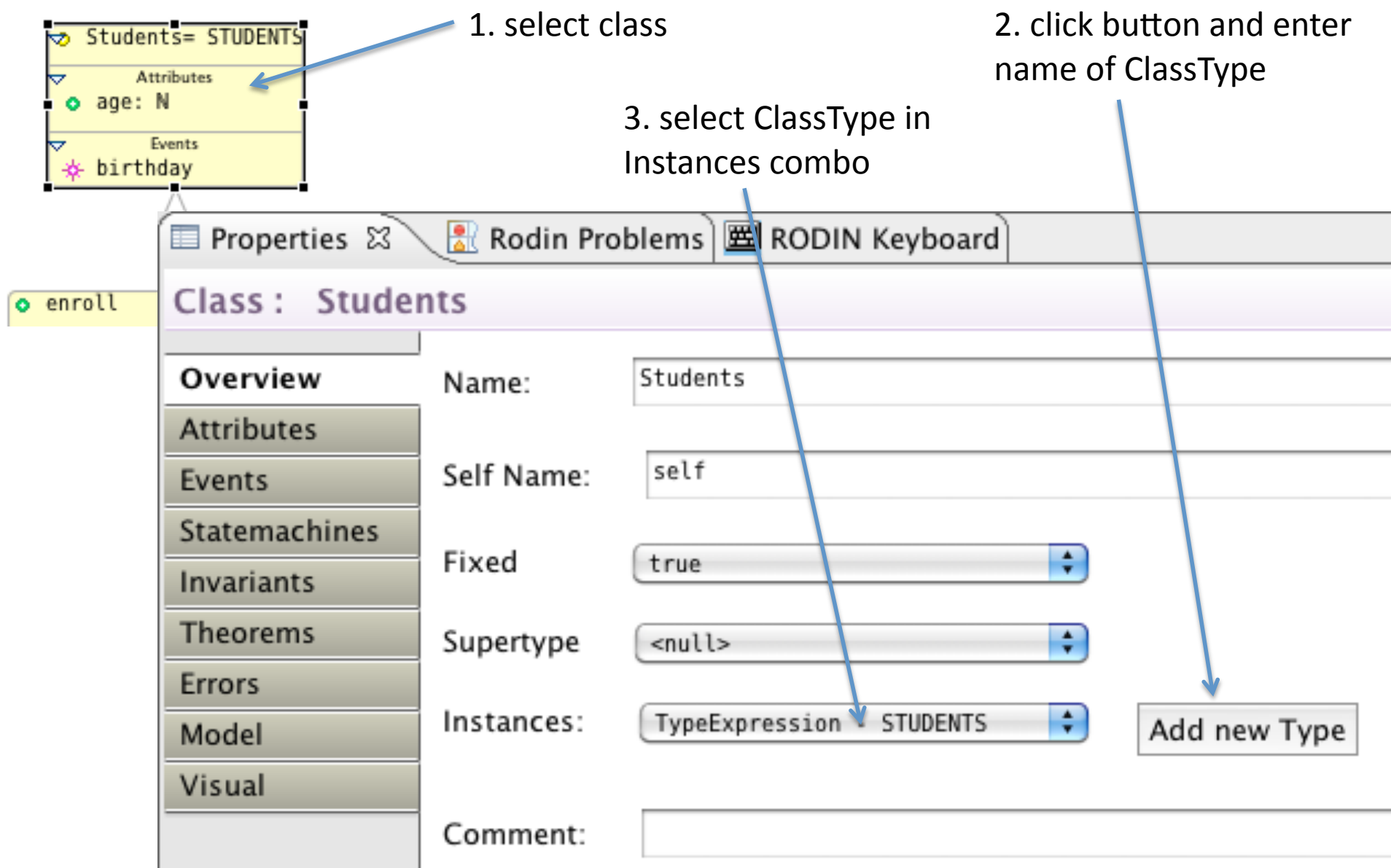
**sets** ID // ClassType  
 PERSON // ClassType  
 NAME // ClassType

**axioms**

@type STUDENTS ∈ P (PERSON)  
 @type Id ∈ STUDENTS → ID  
 @type Name ∈ PERSON → NAME

**end**

# Linking a Class to a ClassType



1. select class

2. click button and enter name of ClassType

3. select ClassType in Instances combo

enroll

**Class: Students**

**Overview**

**Attributes**

**Events**

**Statemachines**

**Invariants**

**Theorems**

**Errors**

**Model**

**Visual**

Name: Students

Self Name: self

Fixed: true

Supertype: <null>

Instances: TypeExpression STUDENTS

Comment:

Add new Type

# Summary of Class Diagrams

Class diagrams for class-oriented modelling  
automatically generates class structures in Event-B

Attribute and association cardinalities

Options for class instances  
variable (constructors and destructors)  
fixed

Automatically generates an 'implicit context'

Context diagrams for class oriented modelling of sets, constants and enumerated types

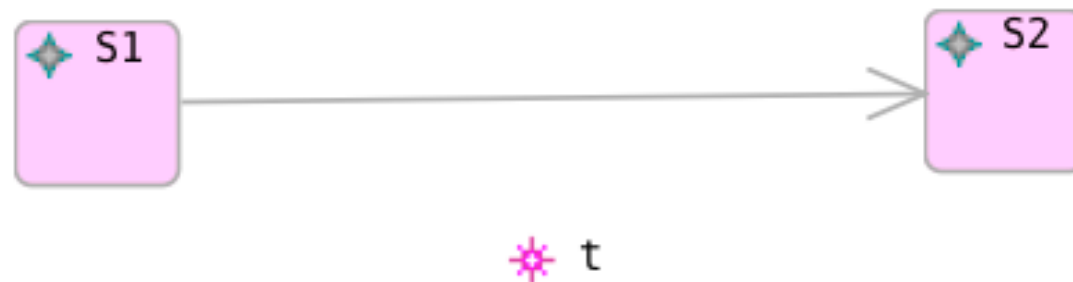
# UML-B State Machine Diagrams

# State Machines

State machines provide a way to model behaviour (transitions)

Constrained by some data (source state)

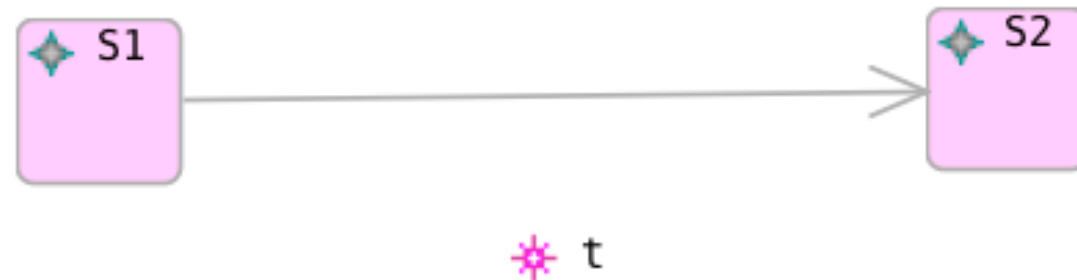
The transition's behaviour is to change the data (to target state)



Transition **t** can only fire when the state is **S1**  
 when **t** fires it changes the state to **S2**

*How could we represent this in Event-B?*

# State Machines to Events



EVENTS

$t \hat{=} \text{WHEN } \langle \text{in } S1 \rangle \text{ THEN } \langle \text{goto } S2 \rangle \text{ END}$

*where,  $\langle \text{in } S1 \rangle$  and  $\langle \text{goto } S2 \rangle$  depend on the data that represents state*

# State machine as a type

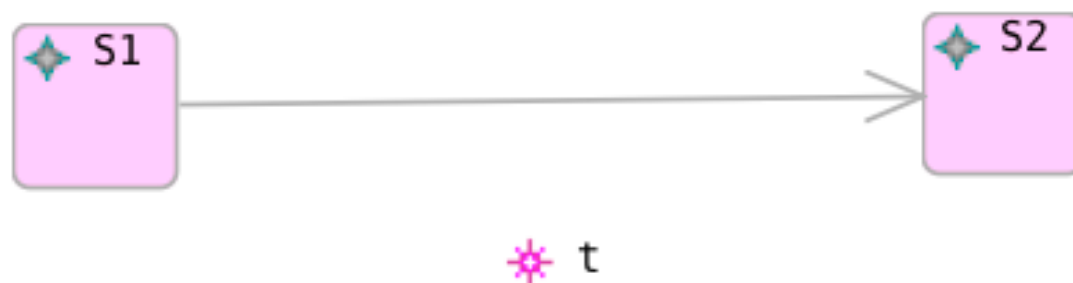
We could treat the whole statemachine as an enumerated type  
 the current state is given by a variable  
 (called **state\_function** translation in UML-B)

## VARIABLES

sm : sm\_STATES

## SETS

sm\_STATES = {S1,S2}



## EVENTS

$t \hat{=} \text{WHEN } sm = S1 \text{ THEN } sm := S2 \text{ END}$

# State machine collection of variables

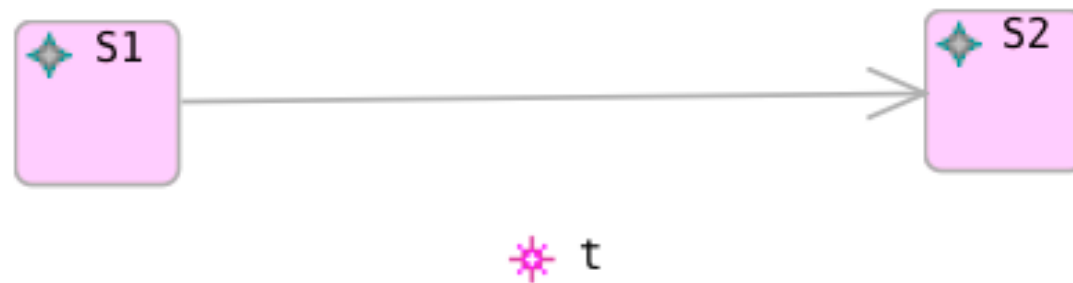
Or we could treat each state as a separate variable  
 (called `state_sets` translation in UML-B)

## VARIABLES

S1 : BOOL

S2 : BOOL

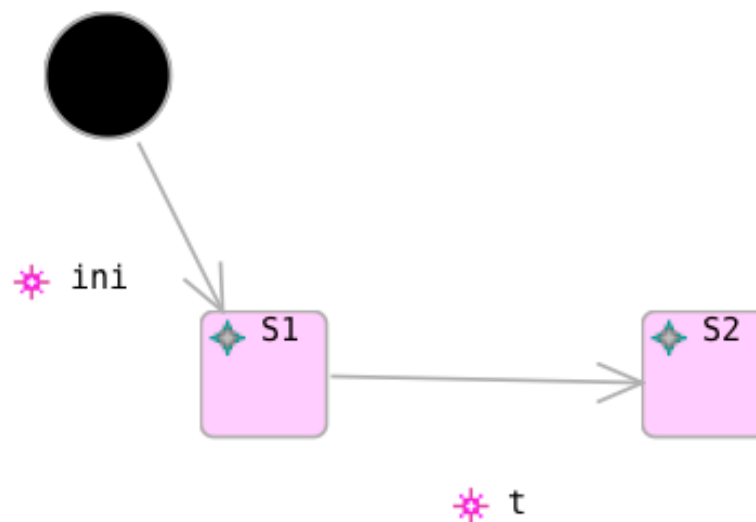
*where, one of S1, S2 is TRUE at any moment*



## EVENTS

$t \hat{=} \text{WHEN } S1 = \text{TRUE}$   
 $\text{THEN } S1 := \text{FALSE}$   
 $\text{S2} := \text{TRUE}$   
 $\text{END}$

# Initial transition



Statemachine as type

**INITIALISATION**

`sm := S1`

States as variables

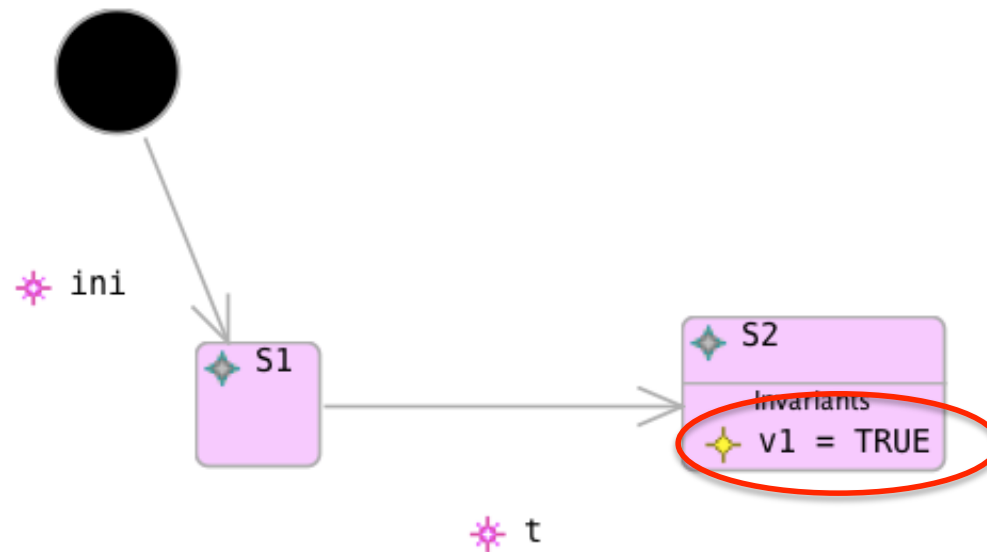
**INITIALISATION**

`S1 := TRUE`

`S2 := FALSE`

# State Invariant

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



Translations:

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

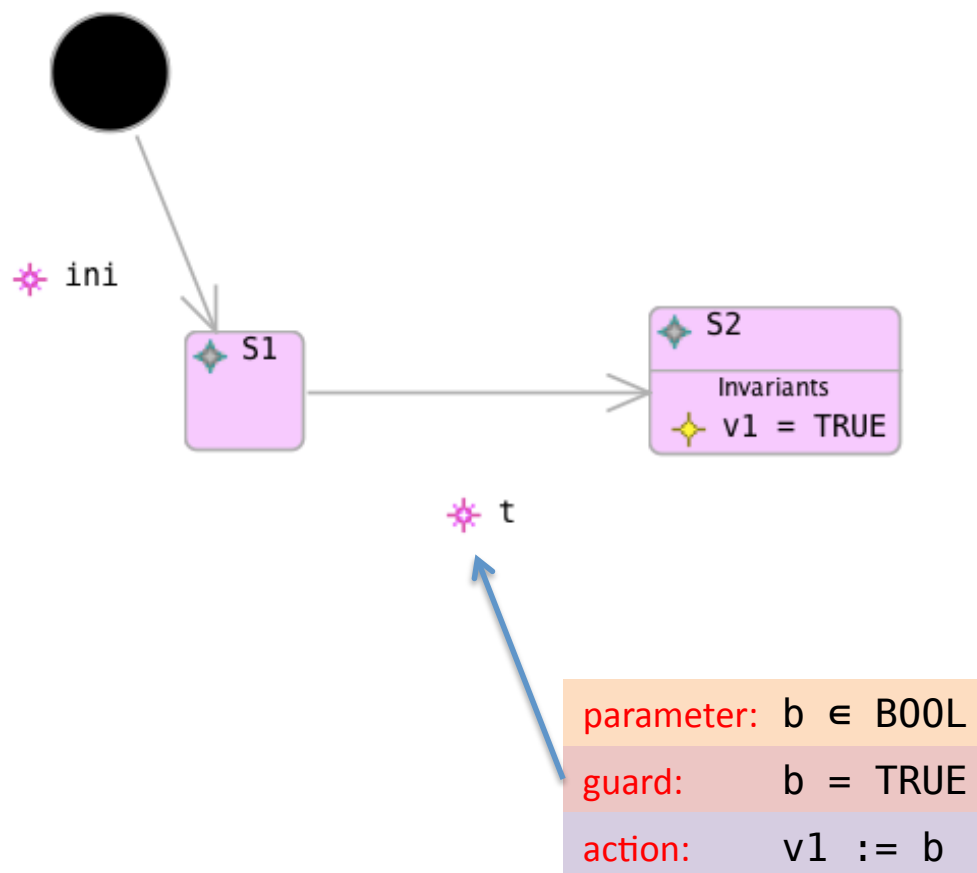
or

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

# Transition parameters, guards and actions

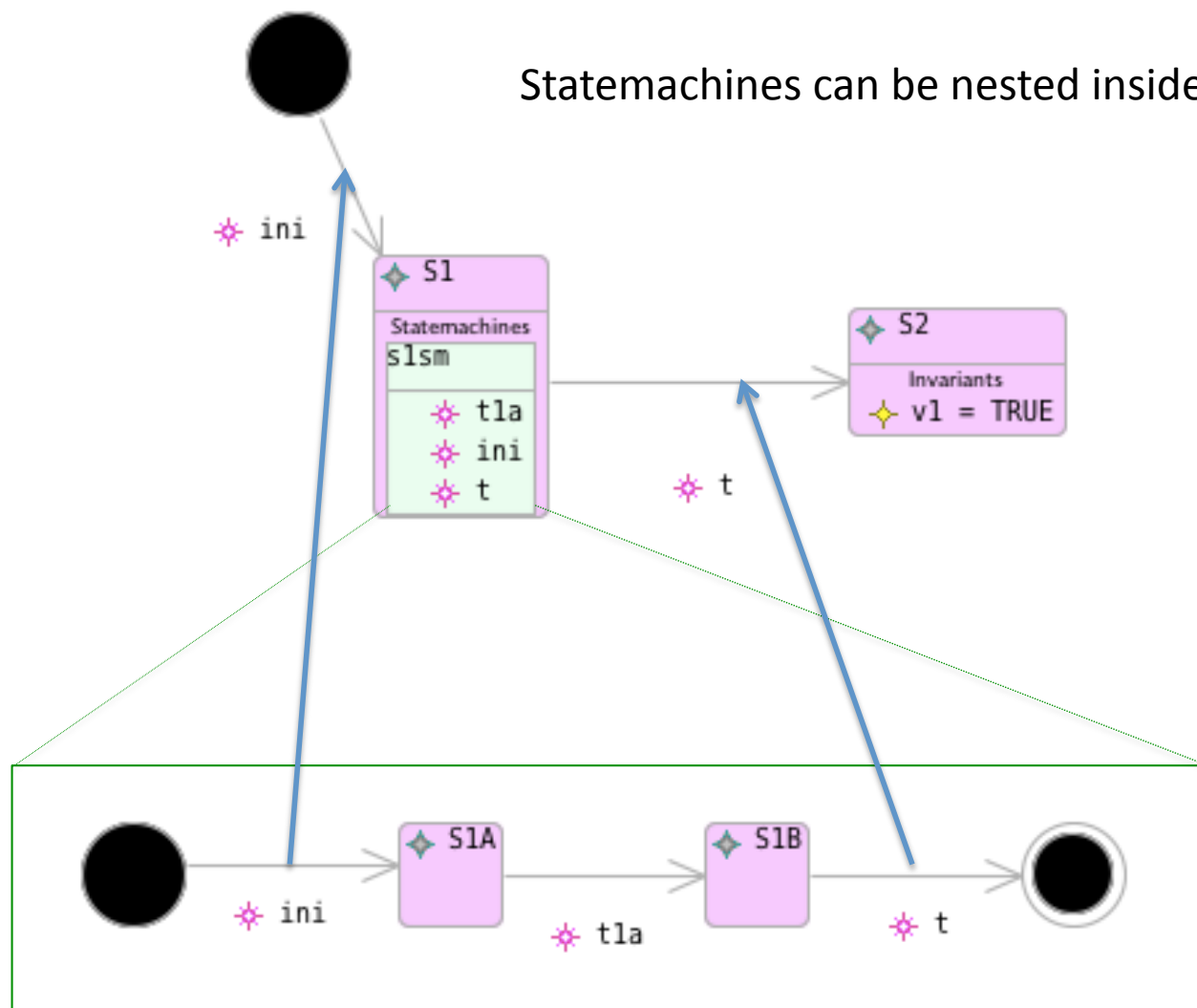
Transitions are events.

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



# Nested Statemachines

Statemachines can be nested inside states



Entry and Exit transitions must align with parent state using **Elaborates** property

## Example

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

A separately controlled safety shield can be opened and closed.  
[The shield is opened automatically when the machine is stopped]

Safety Requirement:

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

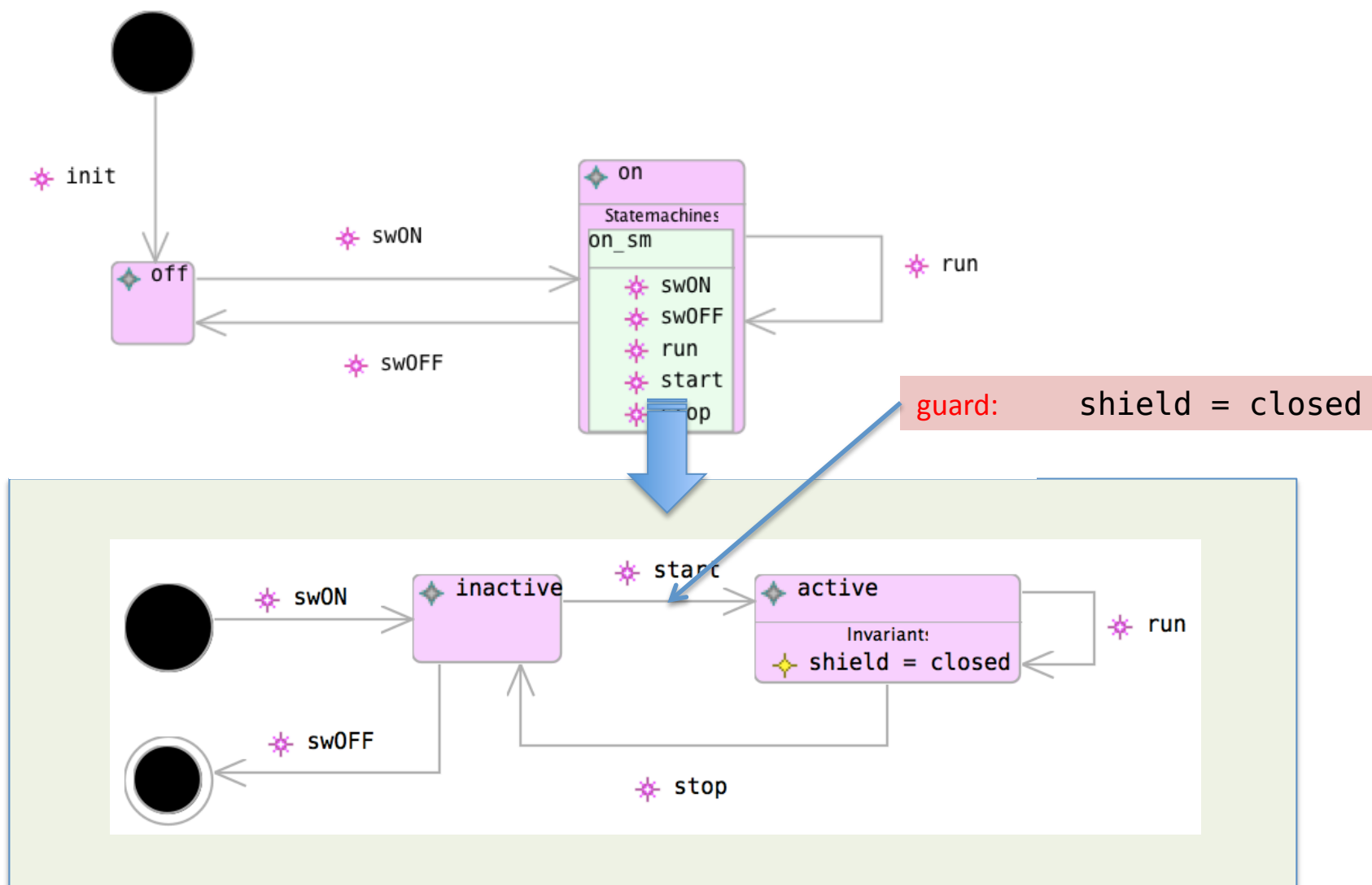
Model the machine and shield as separate statemachines.

Add an invariant to model the safety requirement.

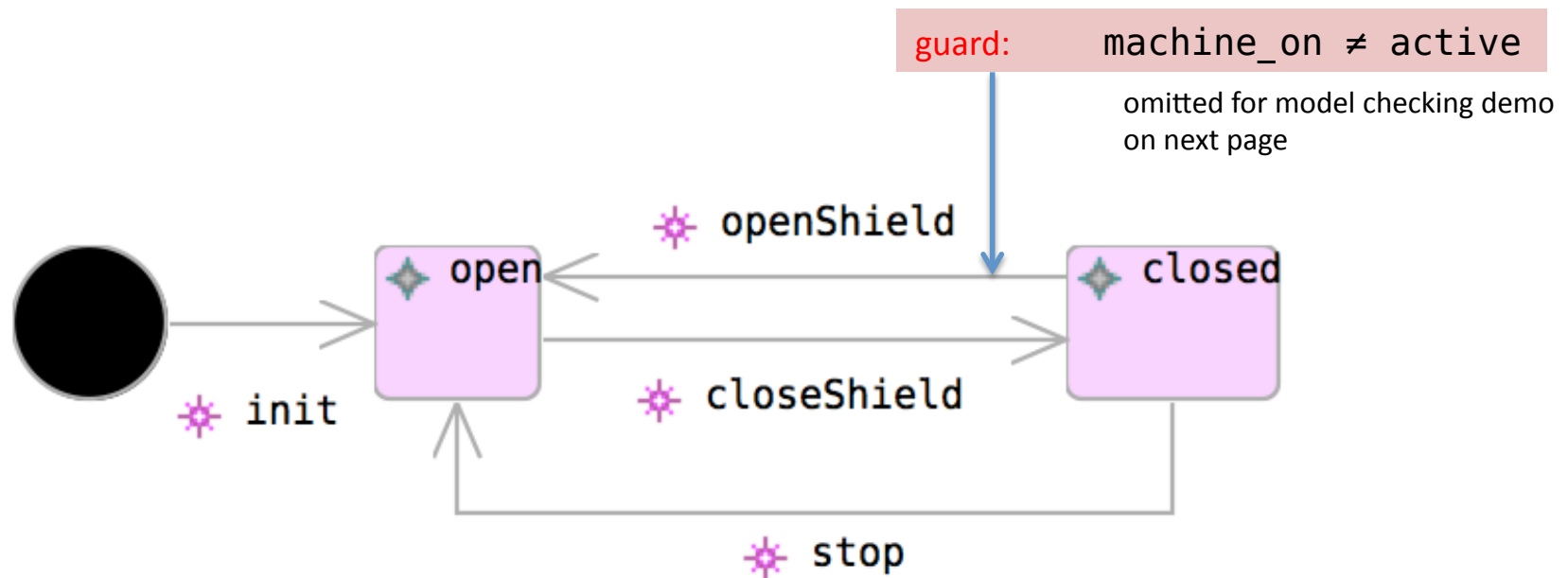
Determine the transitions guards needed to represent the interlocks between the machine and the shield controller.

Use the Pro-B model checker/Animator to ensure that the safety invariant is never violated

# Example – Factory Machine



# Example – Factory Machine Safety Shield



# Statemachine Animation showing invariant violation

C\_Statemachines.m1.machine.stateDiag.anim\_diag

C\_Statemachines.m1.machine.on.machine\_on.stateDiag.anim\_diag

C\_Statemachines.m1.guard.stateDiag.anim\_diag

State

Name	Value	Pre...alue
▼ m1		
guard	open	closed
machine	on	on
machine_on	active	active

History

Operations

- openGuard
- start
- closeGuard
- swON
- INITIALISATION(FALSE,TRUE,TRUE,FALSE,(root))

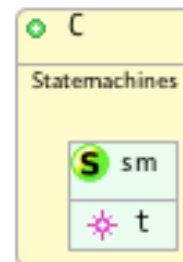
invariant violated!

no event errors detect

# Statemachines in Classes

Statemachines can be added to classes.

Effectively, each class instance has a “copy” of the statemachine



# State machines in Classes – Option 1

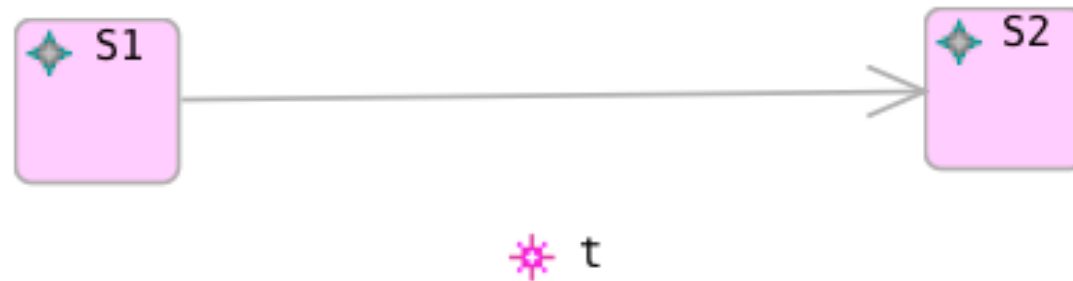
## State machine as a type (state\_function)

### VARIABLES

$sm \in C \rightarrow sm\_STATES$

### SETS

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



### EVENTS

```

t (self) ≙ WHERE  sm(self) = S1
              THEN  sm(self) := S2
              END
  
```

# State machines in Classes – Option 2

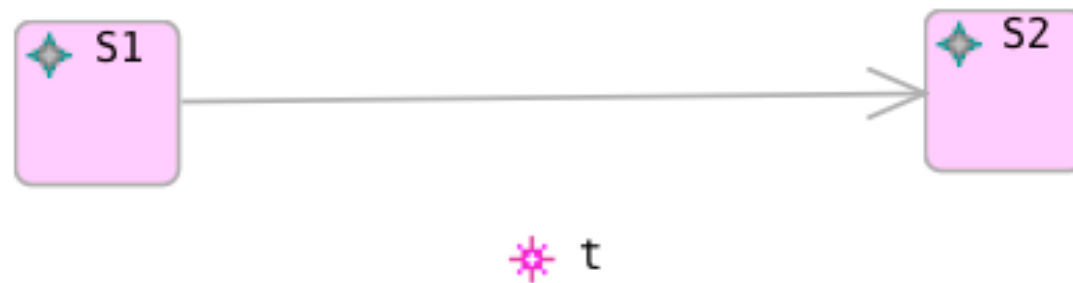
## States as variables (state\_sets)

### VARIABLES

$S1 \in \mathcal{P}(C)$

$S2 \in \mathcal{P}(C)$

where  $S1$  and  $S2$  (and....) are *disjoint*



### EVENTS

$t(\text{self}) \triangleq$  **WHERE**  $\text{self} \in S1$   
**THEN**  $S1 := S1 \setminus \{\text{self}\}$   
 $S2 := S2 \cup \{\text{self}\}$   
**END**

## Initial transition (state\_sets)

For variable instance classes, initial transition is treated as a constructor

```

ini ≐
STATUS
  ordinary
ANY
  self // constructed instance of class C
WHERE
  self.type : self ∈ C_SET \ C
THEN
  C_constructor : C := C ∪ {self}
  sm_enterState_S1 : S1 := S1 ∪ {self}
END

```

## Final transition (state\_sets)

For variable instance classes, final transition is treated as a destructor

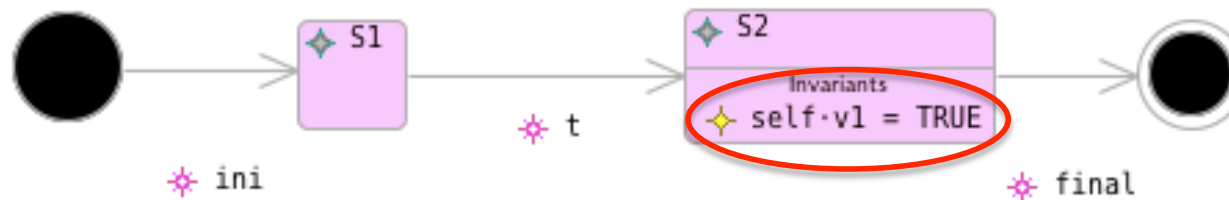
```

final ≐
STATUS
  ordinary
ANY
  self // contextual instance of class C
WHERE
  self.type : self ∈ C
  sm_isin_S2 : self ∈ S2
THEN
  sm_leaveState_S2 : S2 = S2 \ {self}
  C_destructor : C = C \ {self}
END

```

# State Invariant (state\_sets)

Something that must be true for all instances of the class that are in that state.



Translation:

$$\forall \text{self} \cdot ((\text{self} \in C) \Rightarrow ((\text{self} \in S2) \Rightarrow (v1(\text{self}) = \text{TRUE})))$$

# Summary

## Statemachines for modelling behaviour

- ▶ nested – statemachines in states
- ▶ invariants in states
- ▶ transitions are events (with parameters, guards, actions)

## Choice of 2 translations

### Can be lifted to classes

- ▶ initial/final transition as constructor/destructor (variable instance classes)

### State-machines can be animated and model checked

- ▶ (front-end for Pro-B)