



The screenshot displays the CoreASM IDE with the following components:

- Code Editor:** Shows the `DiningPhilosophers.casm` file with the following code:

```
25 function controlled hungry : Agents -> BOOLEAN
26
27 // main program of every philosopher
28 rule Philosopher = {
29   if hungry(self) and (not eating(self)) then
30     if CanPickBothChopsticks then
31       StartEating
32     else
33       print name(self) + " is hungry but can't eat."
34   //...
35   hungry(self) := flip }
```
- Breakpoints:** A breakpoint is set at line 29.
- Expressions:** Shows the state of the `hungry` variable for each philosopher:

Name	Value
"Last Selected Agents"	[Sina, Albert]
"hungry"	hungry
hungry(Albert)	false
hungry(Fredrich)	true
hungry(Juan)	false
hungry(Sina)	false
hungry(Herbert)	true
- Variables:** Shows the state of the `rightChop` variables:

Name	Value
Step	18
Last Selected Agents	[Sina, Albert]
rightChop(Sina)	c3
rightChop(Herbert)	c1
rightChop(Fredrich)	c2
rightChop(Albert)	c5
rightChop(Juan)	c4
- Console:** Displays the output of the program:

```
CoreASM DiningPhilosophers
TABLE: c1 Herbert c2 Fredrich c3 Sina c4 J ^
Juan starts eating.
Fredrich starts eating.
Herbert is hungry but can't eat.
Sina is hungry but can't eat.
Fredrich stops eating.
Juan stops eating.
Herbert starts eating.
Herbert stoos eating.
```

Alexander Raschke (Ulm), Egon Börger (Pisa)
Short Introduction to CoreASM

General Information

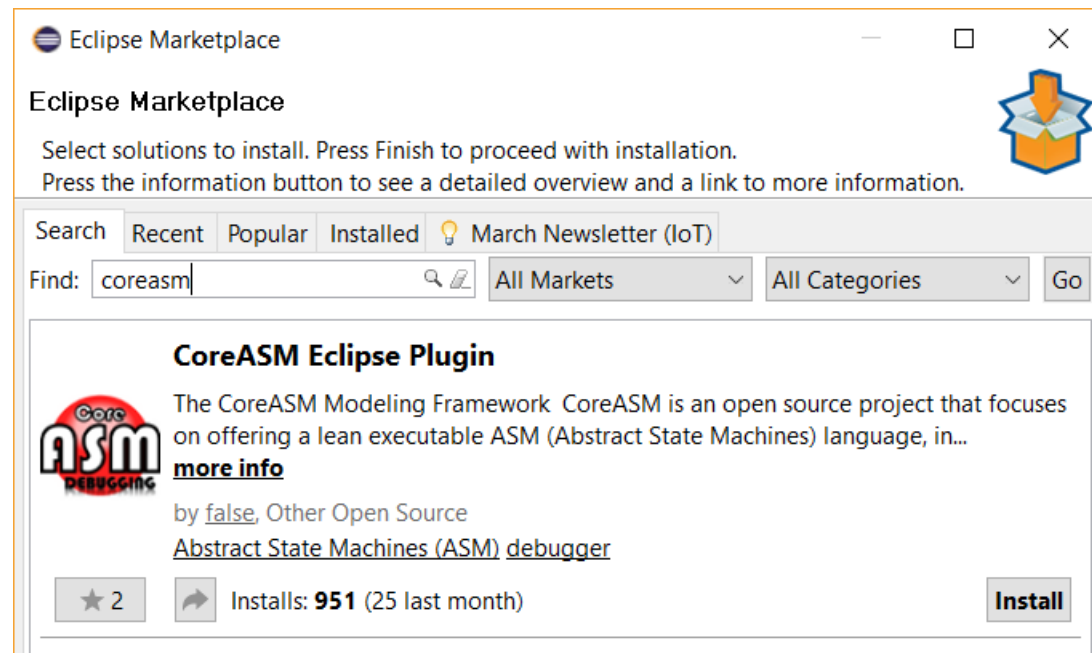
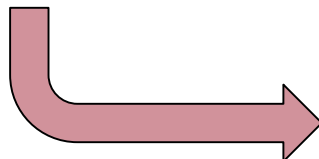
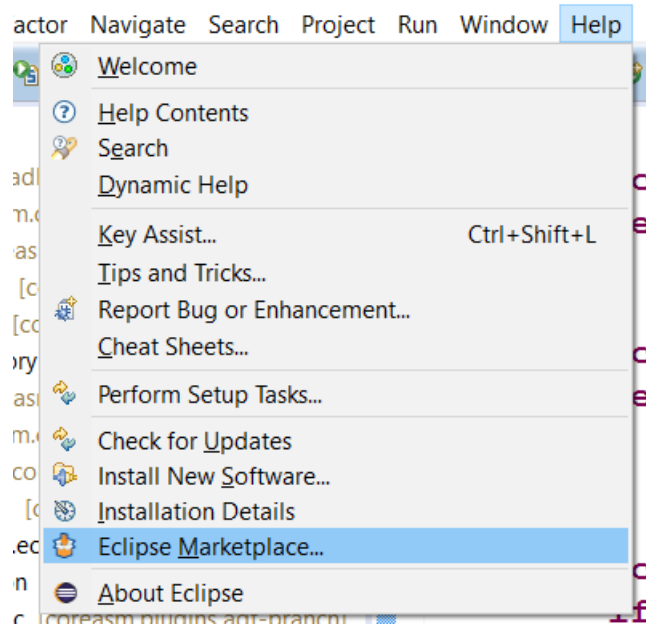
- Developed by Farahbod Roozbeh et al. from 2005 to 2010 as PhD-thesis at Simon Fraser University, Vancouver, Canada.
- Open-source tool (Java 1.7)
- Source code (and documentation) available at <https://github.com/coreasm>
- General Idea:
 - minimal core and everything else is implemented as a plugin (even ConditionalRule (if-then-else) is a plugin!)
 - Easy to extend and very flexible possibilities
 - Set of plugins already shipped with installation (see user manual at https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.engine/rsc/doc/user_manual/CoreASM-UserManual.pdf)

CoreASM overview

CoreASM	
Carma command line interface	eclipse plugin(s) <ul style="list-style-type: none">- project integration- editor- debugger
CoreASM engine <ul style="list-style-type: none">- Parser, Interpreter, Abstract Storage, Schedulers- Basic plugins: Block, Conditional, Forall, ...- Standard plugins: TurboASM, Signature (types), ...- Additional plugins: Time, Scheduling, Math, ...	

Installation

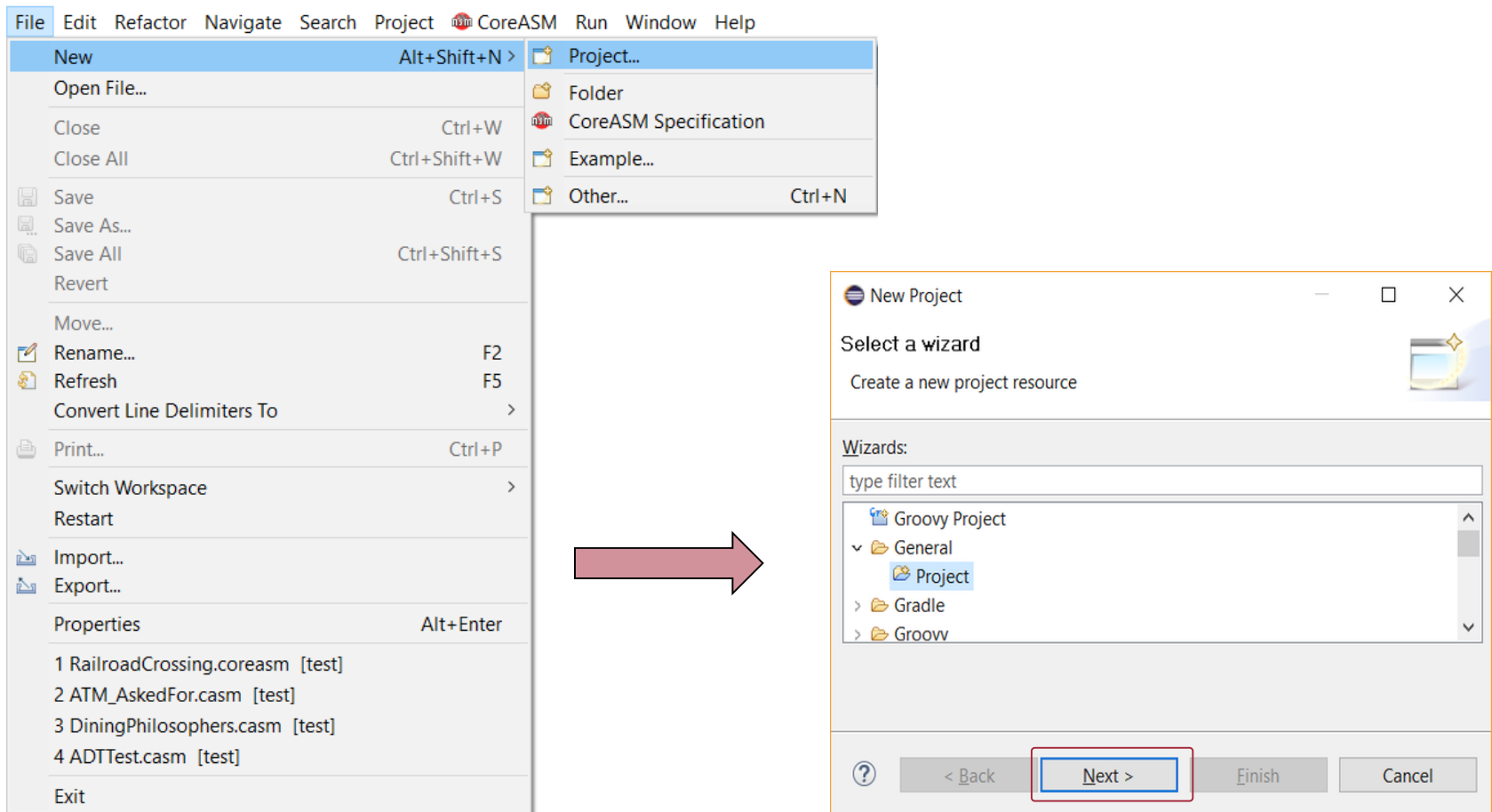
- Easiest way to get CoreASM is via eclipse marketplace:
<https://marketplace.eclipse.org/content/coreasm-eclipse-plugin>



A screenshot of the Eclipse Marketplace window. The window title is 'Eclipse Marketplace'. The main text says: 'Select solutions to install. Press Finish to proceed with installation. Press the information button to see a detailed overview and a link to more information.' Below this is a search bar with 'coreasm' entered. The search results show the 'CoreASM Eclipse Plugin' by 'false'. The description reads: 'The CoreASM Modeling Framework CoreASM is an open source project that focuses on offering a lean executable ASM (Abstract State Machines) language, in... more info'. Below the description, it says 'by false, Other Open Source Abstract State Machines (ASM) debugger'. At the bottom, there is a star rating of 2, a share icon, and the text 'Installs: 951 (25 last month)'. An 'Install' button is visible in the bottom right corner.

Getting started

- First, you have to create a new (general) project in eclipse



The image illustrates the steps to create a new project in Eclipse. On the left, the 'File' menu is open, showing the path: File > New > Project... A red arrow points from the 'Project...' option in the menu to the 'New Project' wizard dialog on the right. The dialog shows the 'Wizards' list with 'General > Project' selected. The 'Next >' button at the bottom is highlighted with a red box.

File Edit Refactor Navigate Search Project CoreASM Run Window Help

New Alt+Shift+N > Project...
Open File... Folder
Close Ctrl+W CoreASM Specification
Close All Ctrl+Shift+W Example...
Save Ctrl+S Other... Ctrl+N
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To >
Print... Ctrl+P
Switch Workspace >
Restart
Import...
Export...
Properties Alt+Enter
1 RailroadCrossing.coreasm [test]
2 ATM_AskedFor.casm [test]
3 DiningPhilosophers.casm [test]
4 ADTTest.casm [test]
Exit

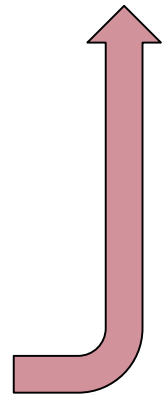
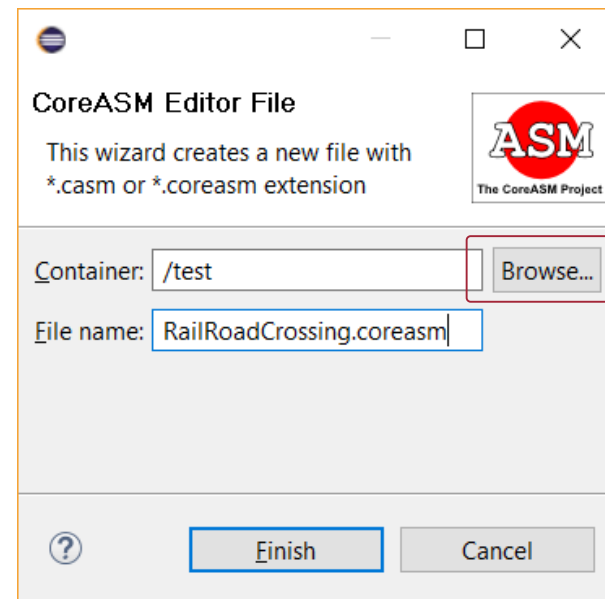
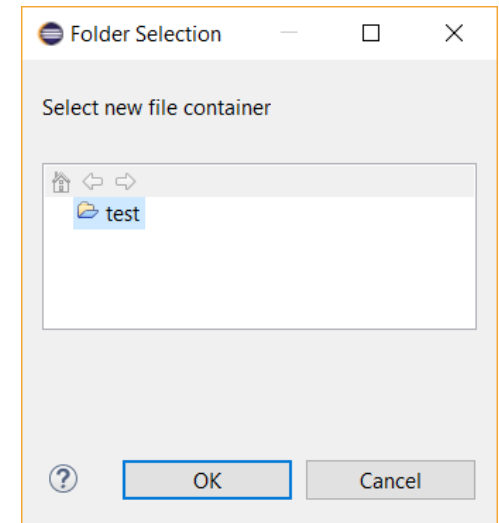
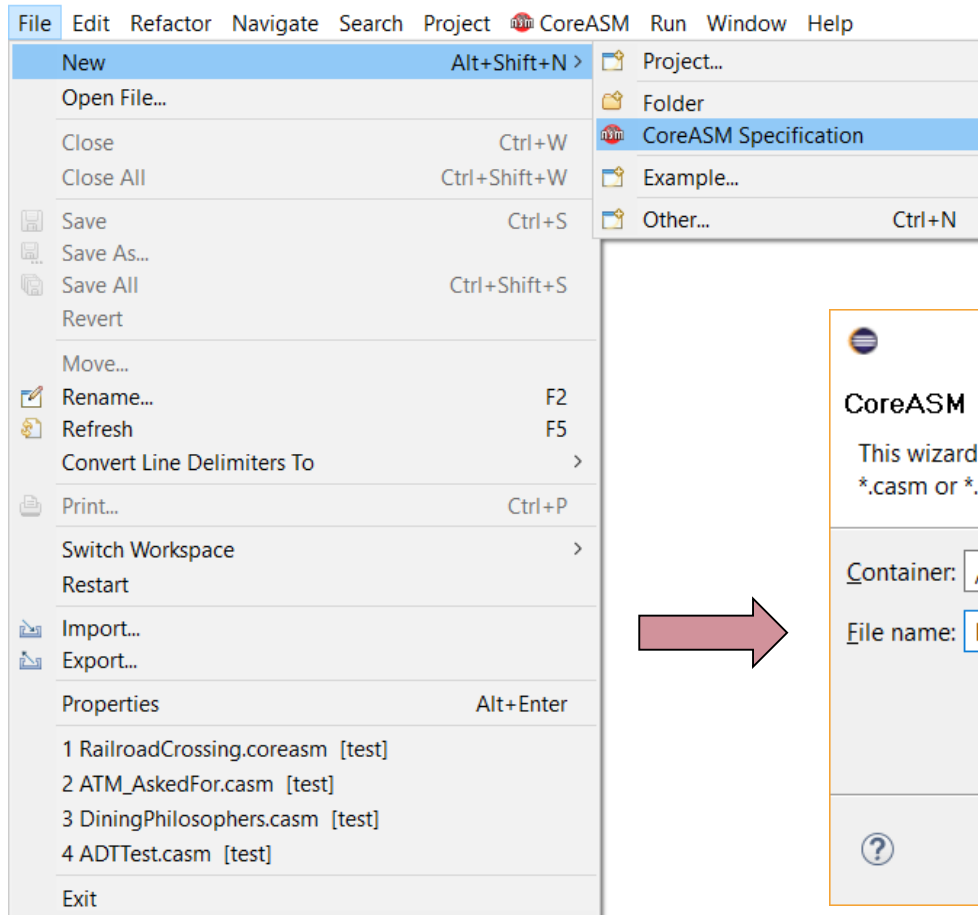
New Project
Select a wizard
Create a new project resource

Wizards:
type filter text
Groovy Project
General
Project
Gradle
Groov

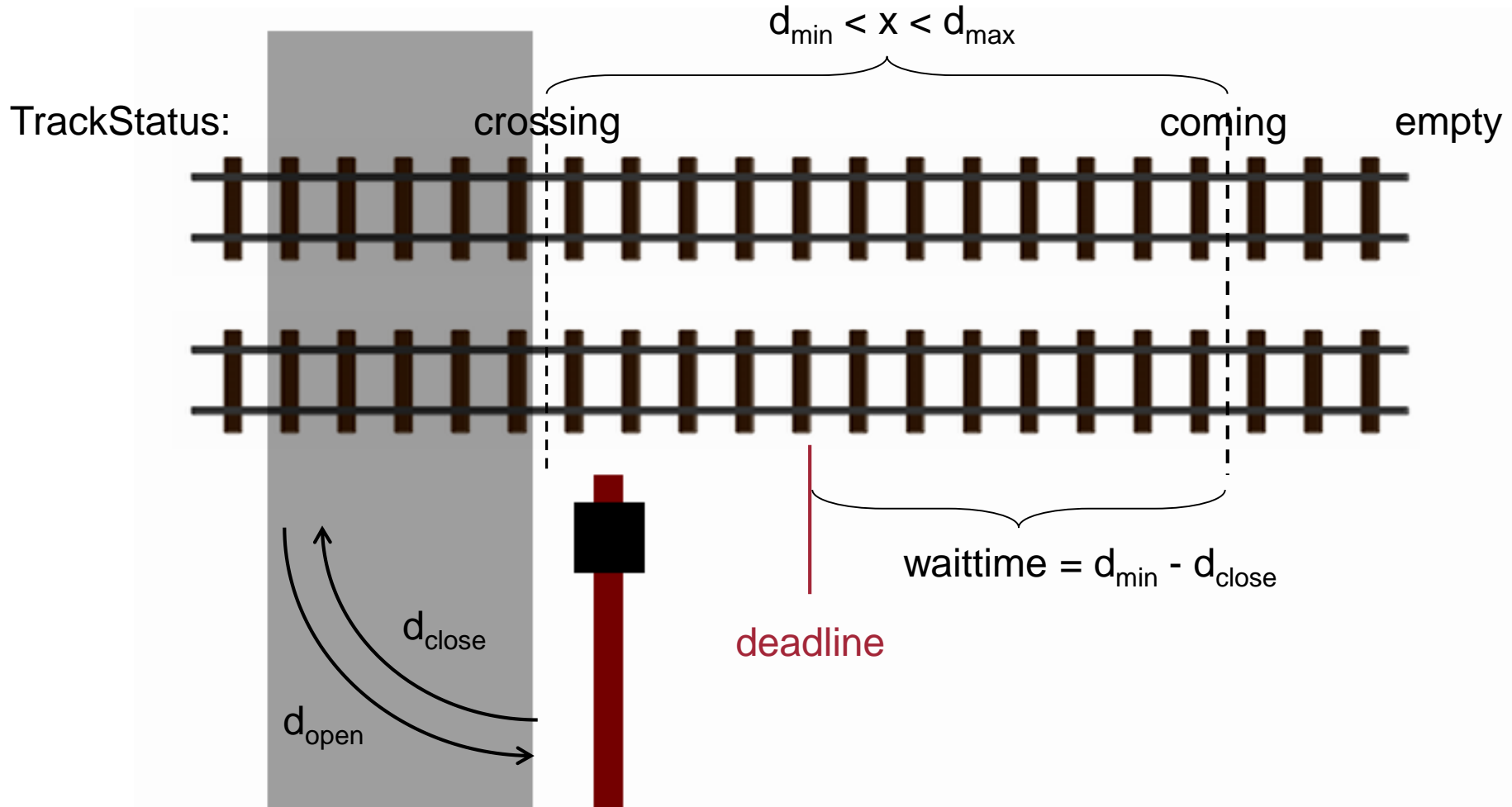
Back Next > Finish Cancel

Getting started

- In this new project container, you can create a CoreASM specification

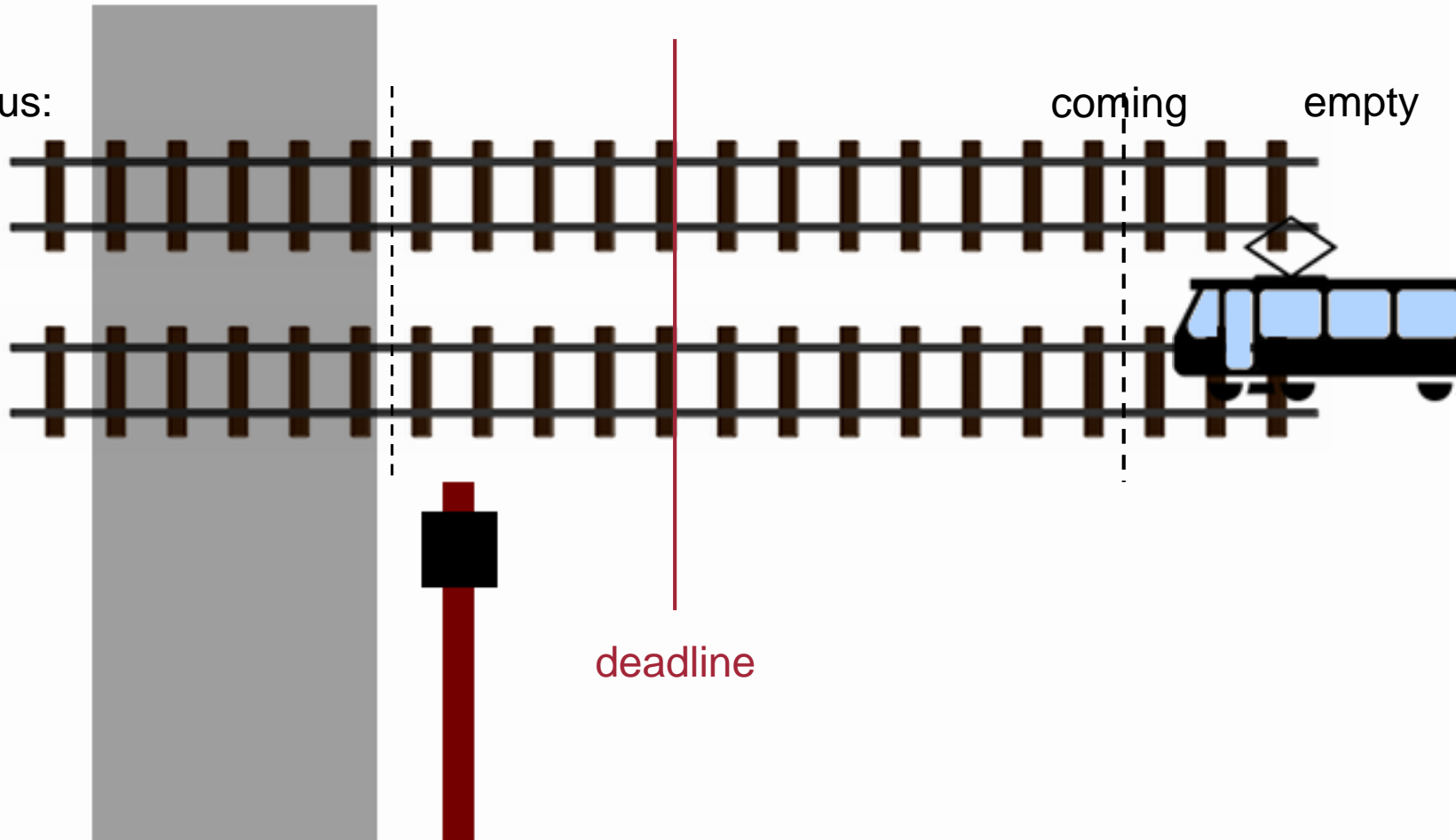


First example: Railroad crossing



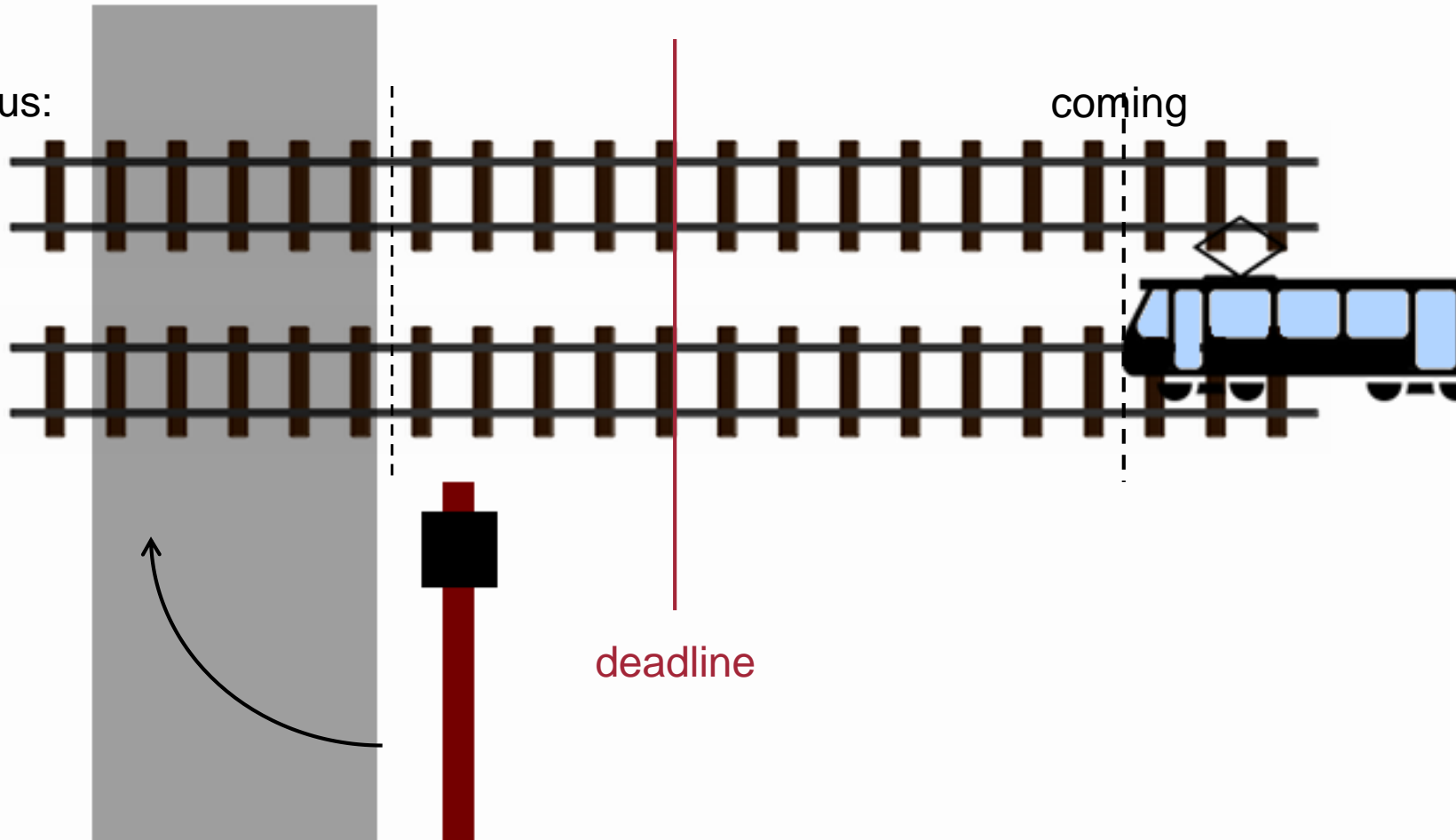
First example: Railroad crossing

TrackStatus:



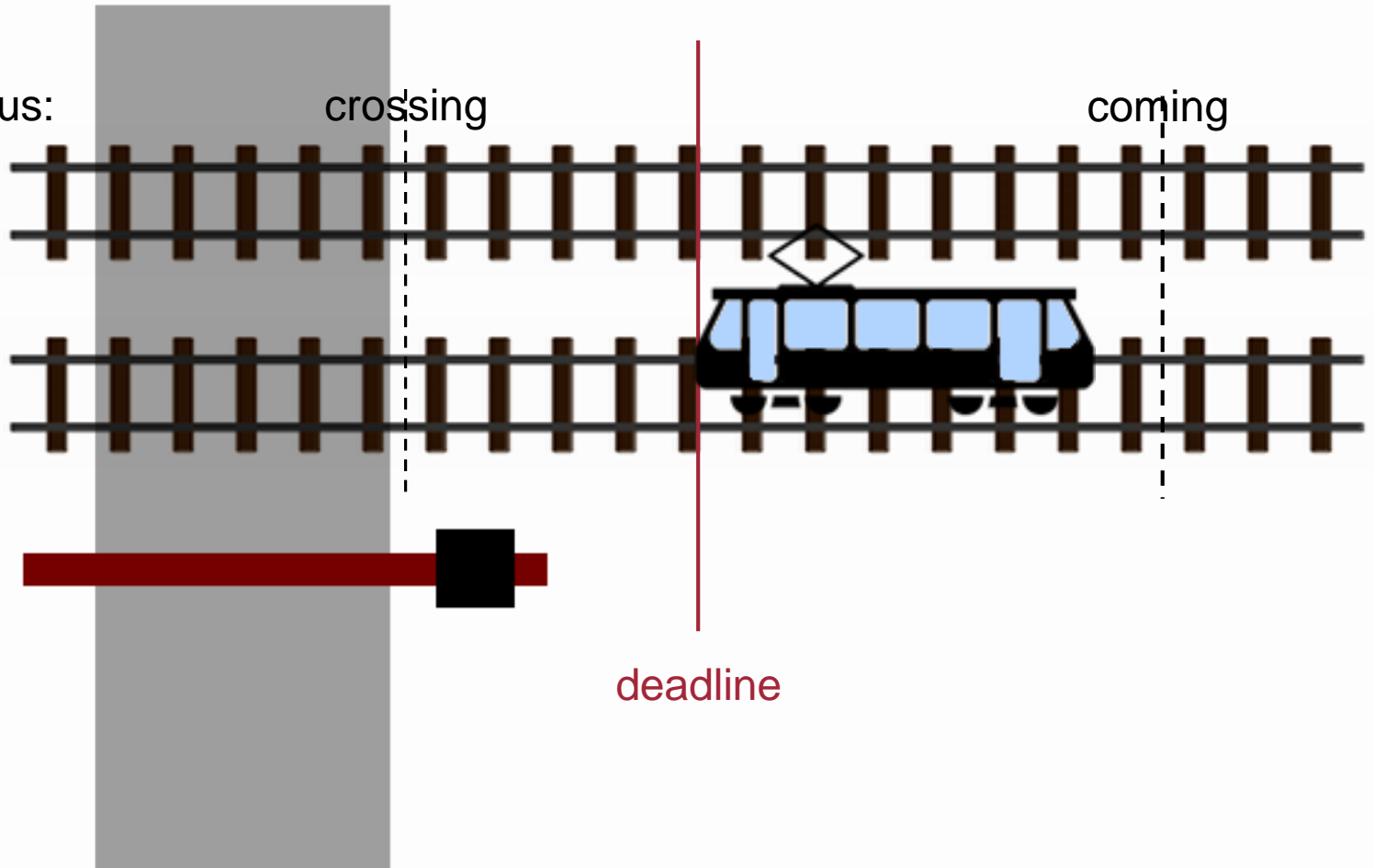
First example: Railroad crossing

TrackStatus:



First example: Railroad crossing

TrackStatus:

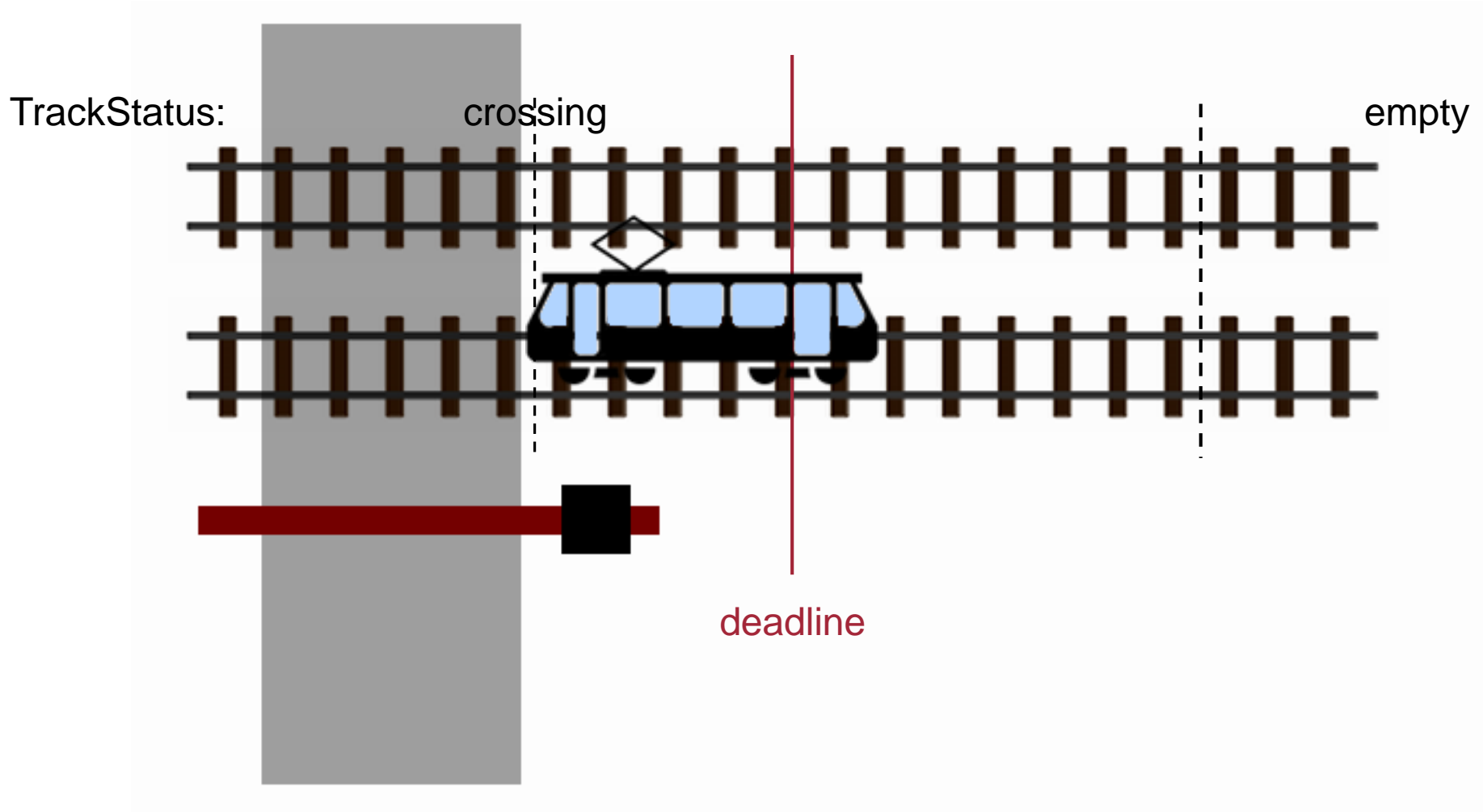


crossing

coming

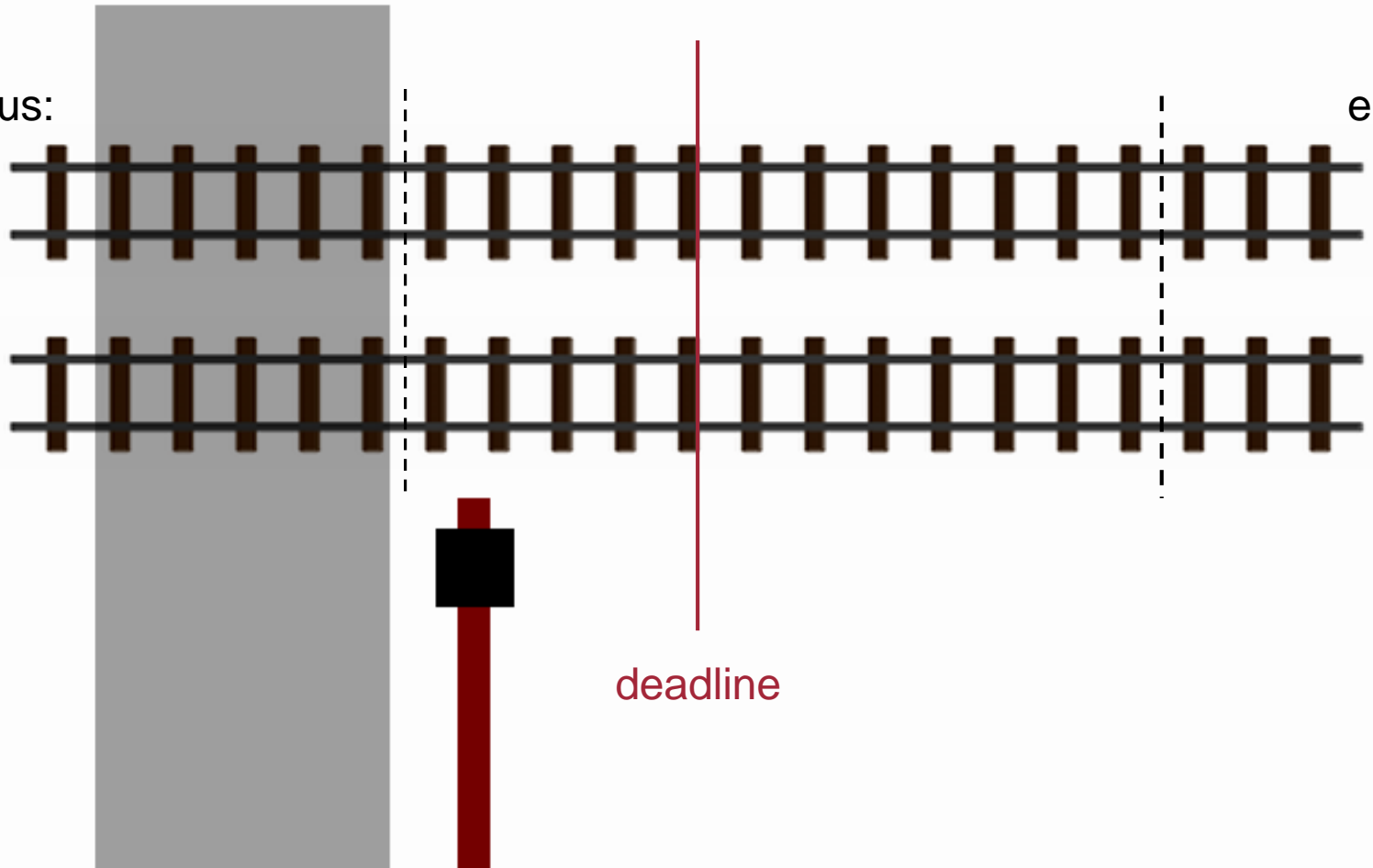
deadline

First example: Railroad crossing



First example: Railroad crossing

TrackStatus:



empty

deadline

From Mathematical Notation to Executable Specifications

TRACKCTL =

forall $x \in TRACK$

SetDeadline(x)

SignalClose(x)

ClearDeadline(x)

SignalOpen

where

SetDeadline(x) = **if** *TrackStatus*(x) = *coming* **and**

Deadline(x) = ∞ **then** *Deadline*(x) := *currtime* + *WaitTime*

SignalClose(x) = **if** *currtime* = *Deadline*(x) **then** *Dir* := *close*

ClearDeadline(x) = **if** *TrackStatus*(x) = *empty* **and**

Deadline(x) < ∞ **then** *Deadline*(x) := ∞

SignalOpen = **if** *Dir* = *close* **and** *SafeToOpen* **then** *Dir* := *open*

SafeToOpen = $\forall x \in TRACK$

TrackStatus(x) = *empty* **or** *currtime* + d_{open} < *Deadline*(x)

GATECTL = SWITCH((*Dir* = *open*, *opened*), (*Dir* = *close*, *closed*))

From Mathematical Notation to Executable Specifications

In the signature we find, besides the monitored system time *currtime*, a monitored function $TrackStatus: TRACK \rightarrow \{empty, coming, inCrossing\}$ and a controlled function $Deadline: TRACK \rightarrow REAL \cup \{\infty\}$ to measure the allowable $WaitTime = d_{min} - d_{close}$ between the appearance of a train and the latest possible moment to start the gate closing (for the gate to be closed in time). A function $Dir \in \{open, close\}$ controlled by the track control signals when to open or close the gate. The actual gate status *opened* or *closed* is the value of the gate control state, which therefore is called *GateStatus*.

The sync RAILCROSSCTL ASM consists of the basic ASMs TRACKCTL and GATECTL controlling the tracks and the gate in the presence of the environment which sets the monitored function *TrackStatus*. For each track the deadline is set upon arrival of a train, the signal to close is sent to the gate control upon deadline expiration, and the deadline is cleared when the track becomes empty. The signal to open is sent to the gate only when it is safe to do so. GATECTL is an instance of the FLIPFLOP on p. 47. We refer to its two control state transitions as *OpenGate* and *CloseGate*.

First example in CoreASM

Every specification must start with this keyword

CoreASM RailRoadCrossing

This name is arbitrary; it is recommended to choose a useful name

```
use StandardPlugins  
use TimePlugin  
use MathPlugin
```

StandardPlugins is a collection of several plugins; it is recommended to use it in every specification

MathPlugin is used to calculate with NUMBERS

TimePlugin provides a monitored function "now" representing the current system time (incremented after every step)

rules are introduced with this keyword

don't miss the "par"/"endpar" keywords!
Intendation is not recognized!

rule TrackControl =

par

forall x **in** Track **do**

par

SetDeadline(x)

SignalClose(x)

ClearDeadline(x)

endpar

SignalOpen

endpar

many rules have a "do" at the end!

"element of" is translated to "in"

no semicolons at the end of lines

TRACKCTL =

forall $x \in TRACK$

SetDeadline(x)

SignalClose(x)

ClearDeadline(x)

SignalOpen

where

SetDeadline(x) = if TrackStatus

"where" for local definitions is not supported, hence all definitions are global

$$\begin{aligned}
 \text{SetDeadline}(x) &= \text{if } \text{TrackStatus}(x) = \text{coming} \text{ and} \\
 &\quad \text{Deadline}(x) = \infty \text{ then } \text{Deadline}(x) := \text{currtime} + \text{WaitTime} \\
 \text{SignalClose}(x) &= \text{if } \text{currtime} = \text{Deadline}(x) \text{ then } \text{Dir} := \text{close} \\
 \text{ClearDeadline}(x) &= \text{if } \text{TrackStatus}(x) = \text{empty} \text{ and} \\
 &\quad \text{Deadline}(x) < \infty \text{ then } \text{Deadline}(x) := \infty \\
 \text{SignalOpen} &= \text{if } \text{Dir} = \text{close} \text{ and } \text{SafeToOpen} \text{ then } \text{Dir} := \text{open} \\
 \text{SafeToOpen} &= \forall x \in \text{TRACK} \\
 &\quad \text{TrackStatus}(x) = \text{empty} \text{ or } \text{currtime} + d_{\text{open}} < \text{Deadline}(x)
 \end{aligned}$$

logical operators are
"and", "or", "not"

```
rule SetDeadline(x) =
  if trackStatus(x) = coming and deadline(x) = infinity then
    deadline(x) := now + waitTime
```

defined in
MathPlugin

```
rule SignalClose(x) =
  if now >= deadline(x) and now <= deadline(x) + span then
    dir := close
```

defined in
TimePlugin

UpdateRule is "!="

```
rule ClearDeadline(x) =
  if trackStatus(x) = empty and deadline(x) < infinity then
    deadline(x) := infinity
```

comparison is "="

```
rule SignalOpen =
  if dir = close and safeToOpen then
    dir := open
```

"safeToOpen" is a **derived**
function, not a rule

```
derived safeToOpen =
  forall t in Track holds
    trackStatus(t) = empty or (now + dopen) < deadline(t)
```

this **forall** is an expression
and not a rule; that's why we
used "**holds**" instead of "**do**"

GATECTL = SWITCH(*(Dir = open, opened)*, *(Dir = close, closed)*)¹²

The rule "Switch" is not predefined in CoreASM. Thus, we simulate its behaviour.

```
rule GateControl =  
  par  
    if dir = open and gateState = closed then gateState := opened  
    if dir = close and gateState = opened then gateState := closed  
  endpar
```

The conditional rule is "if ... then ... else ...". If you skip the "else"-part, it is treated as if you write "... else skip".

Important part: Initialization

- You cannot run the specification, yet, because you have not defined an init rule:

With the "init"-keyword you specify one "start"-rule that is executed at each step

```
init InitRule
```

```
rule InitRule = par
```

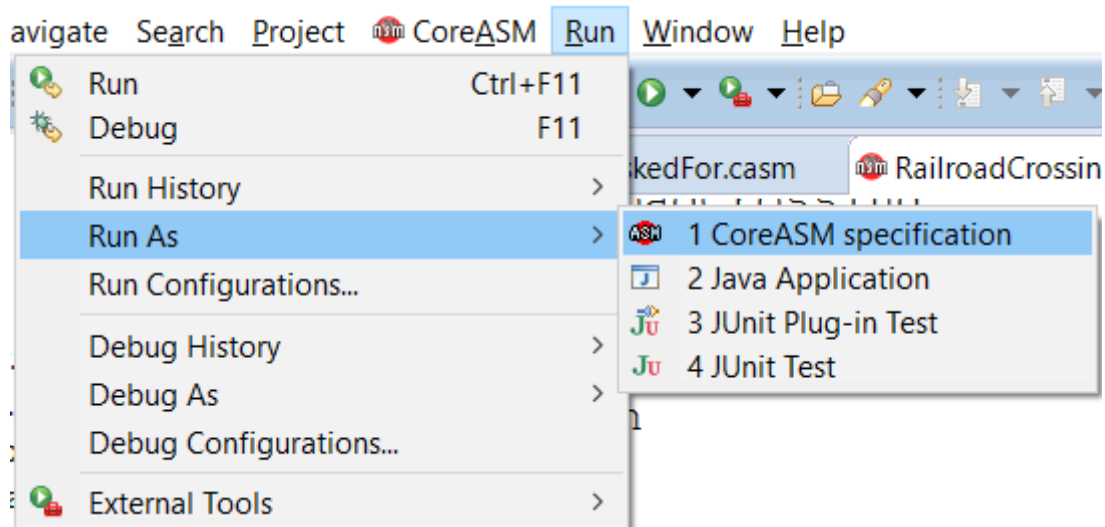
```
  TrackControl
```

```
  GateControl
```

```
endpar
```

You can write the "par"-keyword also at the end of a line

Run the specification



Error message:

```
Cannot perform a 'forall' over undef. Forall domain must be an
enumerable element. (check C:\...\RailroadCrossing.coreasm:100,15)
in TrackControl()
called from InitRule()
```

Change the "Run configuration"

avigate Search Project CoreASM **Run**

- Run Ctrl+F11
- Debug F11
- Run History >
- Run As >
- Run Configurations...**
- Debug History >
- Debug As >
- Debug Configurations...
- External Tools >

Run configurations

coreasm

Name: RailroadCrossing.coreasm

Specification Common

Source

Project test Browse...

Specification RailroadCrossing.coreasm Browse...

Termination condition

- Upon errors Upon failed updates
- When a step returns an empty set of updates
- When a step returns the same set of updates as the previous one
- When there is no agent with a defined program.
- After this many steps have been performed: 10

Verbosity

Log messages with at least the following severity level: Fatal

- Dump updates after each step
- Dump entire state after each step
- Dump final report at termination
- Mark the end of each step
- Print the selected set of agents after each step.

Filter matched 18 of 18 items

Revert Apply

Run Close

Change the "Run configuration"

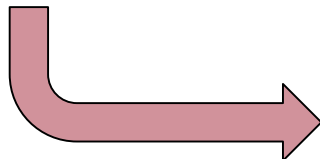
Termination condition

- Upon errors Upon failed updates
- When a step returns an empty set of updates
- When a step returns the same set of updates as the previous one
- When there is no agent with a defined program.
- After this many steps have been performed:

Verbosity

Log messages with at least the following severity level:

- Dump updates after each step
- Dump entire state after each step
- Dump final report at termination
- Mark the end of each step
- Print the selected set of agents after each step.



- - - end of step 1

A closer look at the error message

```
Cannot perform a 'forall' over undef. Forall domain must be an
enumerable element. (check C:\...\RailroadCrossing.coreasm:100,15)
  in TrackControl()
  called from InitRule()
```

- So far, no types or controlled function definitions at all
- every identifier used in the specification is initialized with "**undef**"
- possible types in CoreASM (with plugins)
 - boolean
 - universe (a special built-in kind of set)
 - enumeration
 - number
 - string
 - collections (set, bag, list, queue, stack, map)

Remove "undef"s by defining/initializing all identifiers

```
universe Track = {track1, track2}
```

definition of universes as sets

```
enum TrackStatus = {empty, coming, crossing}
```

```
enum GateState = {opened, closed}
```

```
enum Direction = {open, close}
```

enumerations do not define sets, but possible values of a location

```
rule InitRule = par
```

```
  // initialize all constants
```

```
  dmin := 5000
```

```
  dmax := 10000
```

```
  dopen := 2000
```

```
  dclose := 2000
```

```
  waitTime := 3000
```

```
  dcrossing := 3000
```

```
  span := 500
```

```
  gateState := opened
```

```
  startTime := now
```

line comments with "//"
block comments with "/*..*/"

```
  // initialize the track sensors
```

```
  forall t in Track do
```

```
  par
```

```
    trackStatus(t) := empty
```

```
    deadline(t) := infinity
```

```
    passingTime(t) := 0
```

```
  endpar
```

```
  TrackControl
```

```
  GateControl
```

```
endpar // of the InitRule
```


Still errors due to parallelism

- Initialization is performed in parallel → update with initial state not yet available when "TrackControl" or "GateControl" are executed.
- Typical pattern to ensure initialization executed only once and at the beginning:

predefined set of agents (+ init-rule)

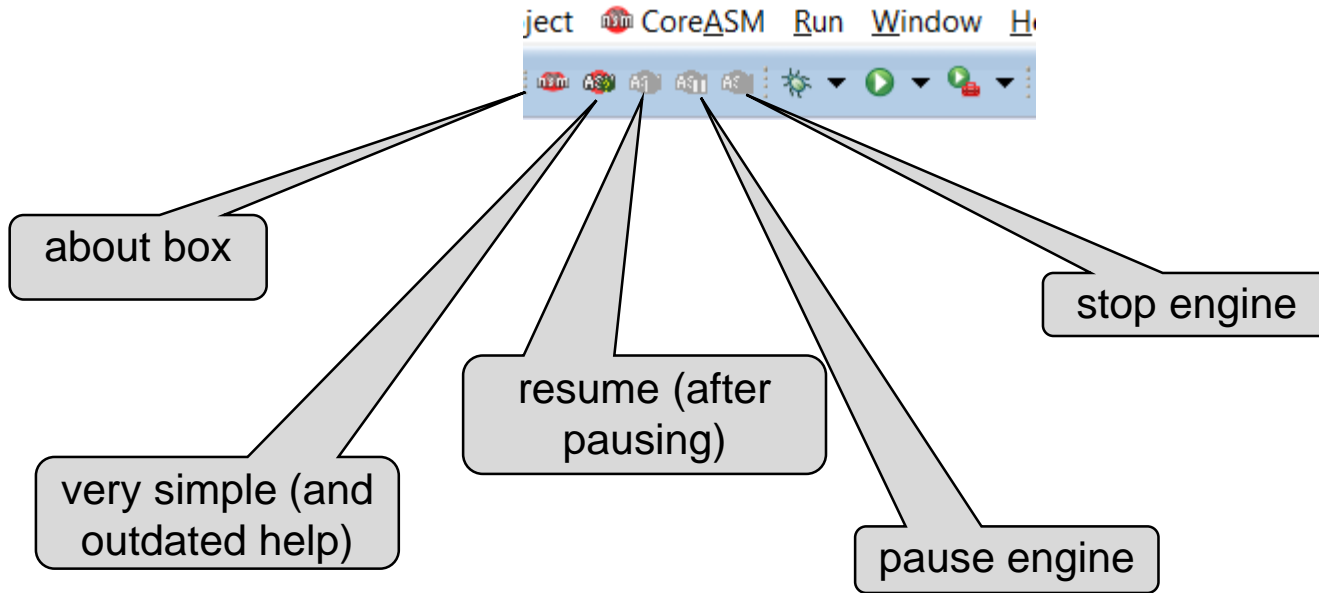
```
universe Agents = {trackController, gateController}
universe Track = {track1, track2}
```

```
rule InitRule = par
  // initialize all constants
  ...
  // initialize the track sensors
  ...
  program(trackController) := @TrackControl
  program(gateController) := @GateControl
  program(self) := undef
endpar
```

assignment of programs (= rules) to agents. **program** is a predefined function on the set of **Agents**. @<rulename> refers to the rule itself (not only the name)

The program of the initialization-agent is removed (set to undef). "**self**" is a predefined keyword to specify the currently executing agent

Running, but... how to stop?



Running, but...

- ... no output is produced, except (if you made the changes in the run configuration):

```
#--- end of step 1
#--- end of step 2
#--- end of step 3
...
```

Don't forget to add "observer" to the set of Agents and to assign this rule as program for this agent!

- Solution: provide an observer

```
rule ObserverProgram =
```

```
  seqblock
```

```
    print "Time: " + ((now - startTime) / 1000) + " seconds"
```

```
    forall t in Track do
```

```
      print "Track " + t + " is " + trackStatus(t)
```

```
      print "Gate is " + gateState
```

```
      print ""
```

```
    endseqblock
```

The TurboASM rule "**seqblock**" executes the rules inside sequentially.

"**print**" uses Java "toString"-method to generate string for arbitrary types for console output, followed by a newline

Running, but...

- ... always the same output is produced
- Solution: provide an environment simulating the movement of a train:

```

rule EnvironmentProgram =
  choose t in Track do par
    if trackStatus(t) = empty then
      if random < 0.001 then par
        trackStatus(t) := coming
        passingTime(t) := now + dmin
      endpar
    if trackStatus(t) = coming then
      if passingTime(t) < now then par
        trackStatus(t) := crossing
        passingTime(t) := now + dcrossing
      endpar
    if trackStatus(t) = crossing then
      if passingTime(t) < now then
        trackStatus(t) := empty
      endpar
  endpar
  
```

"choose" chooses an arbitrary element of the given set

"random" provides a number x with $0 \leq x < 1$

Again, don't forget to add "environment" to the set of Agents and to assign this rule as program for this agent!

Best practices

option Signature.NoUndefinedId strict

don't allow undefined identifiers

define function types

function gateState: -> GateState **initially** opened

function trackStatus: Track -> TrackStatus

function deadline: Track -> **NUMBER**

function dir : -> Direction **initially** open

function passingTime: Track -> **NUMBER**

function startTime: -> **NUMBER**

define initial values (where possible) at the declaration

derived dmin = 5000

derived dmax = 10000

derived dopen = 2000

derived dclose = 2000

derived waitTime = dmin - dclose

derived dcrossing = 3000

derived span = 500

Define constants as derived functions

From mathematical notation to executable specification

Roundup:

- Initialization
- definition of agents, disabling of init agent
- type definitions optional
- output with “print”
- modeling of environment

Some more features making the development easier:

- syntax highlighting
- code completion
- quick fixes
- debugging
- update set comparison

Further documentation

- Please read the **User manual** to get familiar with the language:
https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.engine/rsc/doc/user_manual/CoreASM-UserManual.pdf
- Please read the **Debugging manual** to get familiar with the debugging possibilities:
https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.eclipse/rsc/doc/CoreASM_Eclipse_Debugger_Manual.pdf
- Please read the **Design documentation**, if you want to know details about how CoreASM works or if you want to contribute to this project:
<https://github.com/CoreASM/coreasm.core/raw/master/org.coreasm.engine/rsc/doc/CoreASM-DesignDocumentation.pdf>

Where to get it

eclipse marketplace (<http://marketplace.eclipse.org/search/site/coreasm>)

eclipse update site

(<http://webcoreasm.informatik.uni-ulm.de/coreasm-repository/>)

Sources on github

(<https://github.com/coreasm>)

If you have any question, don't hesitate to contact us via coreasm@uni-ulm.de

Michael Stegmaier



Alexander Raschke

Copyright Notice

It is permitted to (re-) use these slides under the CC-BY-NC-SA licence

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

i.e. in particular under the condition that

- the two original authors are mentioned
- modified slides are made available under the same licence
- the (re-) use is not commercial

