Integrating Concurrency Theory with State

Steve Schneider
University of Surrey, UK

Joint work with Helen Treharne

Challenge

- How can we model and reason about systems with complex and detailed state information and complex concurrent or control behaviour?

- How far can we get by integrating formal methods?

State- and event-based formalisms

- Process algebra/concurrency theory:
  - –Good for complex interaction between processes
  - –Focus on patterns of communication and control
  - –Abstract state as much as possible
  - –State tends to be mostly orthogonal

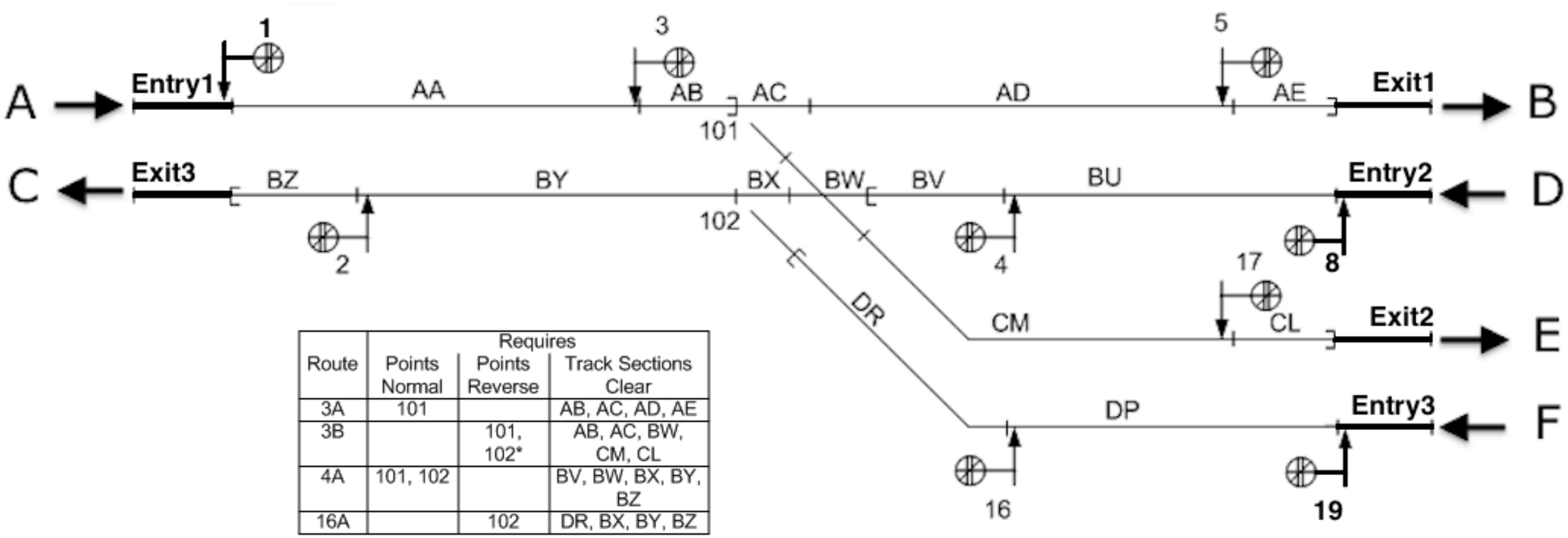
- **Challenge**: handling rich state information

State- and event-based formalisms

- State-based formalisms (Z, B, VDM,…)
  - Suited to data-rich systems and structured state
  - State transitions: specify how and when state is updated
  - Weak or cumbersome notions of flow of control and interaction

- **Challenge**: want to consider higher level behavioural properties (e.g. traces, LTL)

Motivating Example: railways



Route	Requires: Points Normal	Requires: Points Reverse	Requires: Track Sections Clear
3A	101		AB, AC, AD, AE
3B		101, 102*	AB, AC, BW, CM, CL
4A	101, 102		BV, BW, BX, BY, BZ
16A		102	DR, BX, BY, BZ

* flank protection

State:

train locations;

points positions;

signals; locks on routes;

control tables; topology

Trains:

driving rules;

Road map

Application

MDE

Railways

Protocols

fUML

Verification

frameworks

ProB tool

Theorem-proving

Theorems

Formalisms

Common semantic framework

+time

B

CSP

Event-B

Structure of Talk

- B and Event-B for state
- Combination with CSP
- Example: modelling for railway safety
- Incorporating time

Structure of Talk

- **B and Event-B for state**
- Combination with CSP
- Example: modelling for railway safety
- Incorporating time

Classical B machines and control

```
Machine  M1 =                                  n: 0
Variables      n
Invariant      n ∈ {0,1}
Initialisation   n:=0
Operations
  up=
     pre n=0 then n:=n+1 end
 down =
     pre n = 1 then n:=n-1 end
```

* Operations always enabled, but the machine will break (diverge) if they are called outside their preconditions

Classical B machines and control

```
Machine  M1 =
Variables      n
Invariant      n ∈ {0,1}
Initialisation   n:=0
Operations
 up=
    pre n=0 then n:=n+1 end
down =
    pre n = 1 then n:=n-1 end
```

n: 1

* Operations always enabled, but the machine will break (diverge) if they are called outside their preconditions

* up can occur as the first event, setting n to 1.

Classical B machines and control

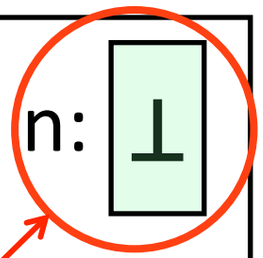
```
Machine  M1 =
Variables      n
Invariant      n ∈ {0,1}
Initialisation   n:=0
Operations
 up=
    pre n=0 then n:=n+1 end
down =
    pre n = 1 then n:=n-1 end
```

n: 0

* Operations always enabled, but the machine will break (diverge) if they are called outside their preconditions
* up can occur as the first event
* down can occur as the second event

Classical B machines and control

Machine M1 =
Variables n
Invariant n ∈ {0,1}
Initialisation n:=0
Operations
up=
pre n=0 then n:=n+1 end
down =
pre n = 1 then n:=n-1 end



* Operations always enabled, but the machine will break (diverge) if they are called outside their preconditions
* up can occur as the first event
* down can occur as the second event
* a second occurrence of down will diverge the machine

Classical B machines and control

Machine M1 =
Variables n
Invariant n ∈ {0,1}
Initialisation n:=0
Operations
up=
pre n=0 then n:=n+1 end
down =
pre n = 1 then n:=n-1 end

n: 0

* Operations always enabled, but the machine will break (diverge) if they are called outside their preconditions

- preconditions do not determine control flow
- they constrain what control flow is necessary to ensure correct behaviour

Classical B machines and control

Machine M1 =
Variables n
Invariant n ∈ {0,1}
Initialisation n:=0
Operations
up=
pre n=0 then n:=n+1 end
down =
pre n = 1 then n:=n-1 end

n: 0

* Operations always enabled, but the machine will break (diverge) if they are called outside their preconditions

$$P = up \rightarrow down \rightarrow P$$

Variant: Event-B for modelling

Machine M1 =
Variables n
Invariant n ∈ {0,1}
Initialisation n:=0
Event up =
when n=0 then n:=n+1 end
Event down =
when n = 1 then n:=n-1 end

n: 0

* Events are enabled when their guard is true, and **blocked** when their guard is false
* up can occur as the first event
* down cannot occur, and will be blocked

* control variables in the guards used to manage control flow
* very cumbersome in practice, and scales badly

B machine semantics

- **Proof obligations** for a consistent machine
  - –Initialisation establishes the invariant
  - –Operations preserve the invariant: they are guaranteed to establish it when called within their preconditions
  - –Machine Invariant captures safety properties on states

- wp semantics used:
  - – for Invariant I and operation PRE P THEN S END:

$$I \wedge P \Rightarrow wp(S, I)$$

Structure of Talk

- B and Event-B for state
- **Combination with CSP**
- Example: modelling for railway safety
- Incorporating time

CSP(-lite)

$$
\begin{aligned}
P \quad ::= \quad & STOP \mid SKIP \mid P;Q \mid \\
& a \rightarrow P \mid c?x \rightarrow P \mid c!v \rightarrow P \mid \\
& P \sqcap Q \mid P \,\Box\, Q \mid \\
& P \parallel Q \mid P \mid\mid\mid Q \mid P \setminus A \\
& X \mid \mu X \,.\, P
\end{aligned}
$$

CSP semantics: sets of observations

- **Traces**: finite sequences of (visible) events:
  traces(P) = { tr | tr can be observed of P }

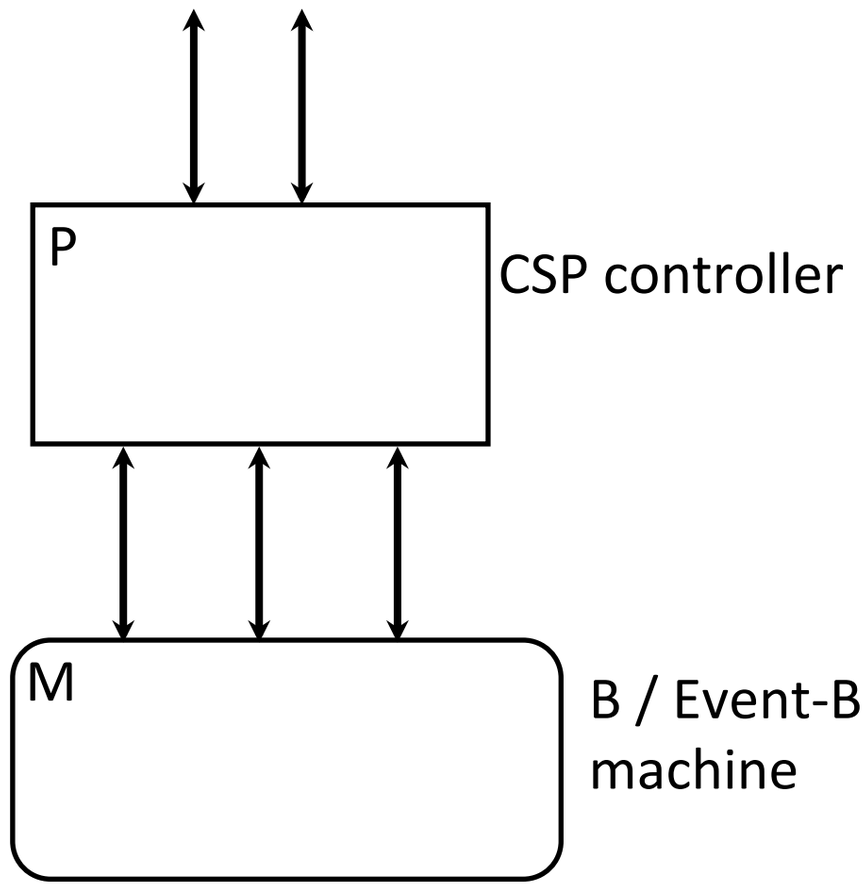
- **Failures**: traces together with refusal sets:
  failures(P) = {(tr,X) | (tr,X) can be observed of P}

- **Divergences**: traces during which the process may diverge:
  divergences(P) = {tr | P may diverge when performing tr}

- **Infinite traces**: infinite sequences of (visible) events:
  infinites(P) = { u | u can be observed of P }

Combining CSP and B

- Some operations in the machine correspond to events in the controller
- The interface is the alphabet of M
- Divergence can occur if an operation of M is called outside its precondition (Classical B)
- Consistency: when P ensures that M does not diverge.



Semantic approach

- The B machine is given a CSP semantics.
- Morgan's wp semantics (1990) for action systems is applicable here, to give CSP semantics for B machines

$$\overline{wp}(S, P) \quad = \quad \neg wp(S, \neg P)$$

$$\begin{aligned}
traces(M) &= \{\langle a_1, a_2, \ldots, a_n \rangle \mid \overline{wp}(init; a_1; a_2; \ldots; a_n \ , \ true\} \\
failures(M) &= \{\langle (a_1, a_2, \ldots, a_n, X) \rangle \mid \overline{wp}(init; a_1; a_2; \ldots; a_n \ , \ \bigwedge_{a \in X} \neg g_a\} \\
divergences(M) &= \{\langle a_1, a_2, \ldots, a_n \rangle \mid \overline{wp}(init; a_1; a_2; \ldots; a_n \ , \ false\}
\end{aligned}$$

CSP||B semantic foundation

- Once M has a CSP semantics then it can be treated as a CSP process, and combined using the CSP operators.
- Thus a controlled component P||M can be given a CSP semantics.
- Want access to tool support available for B and for CSP, and lift the results to the combination.

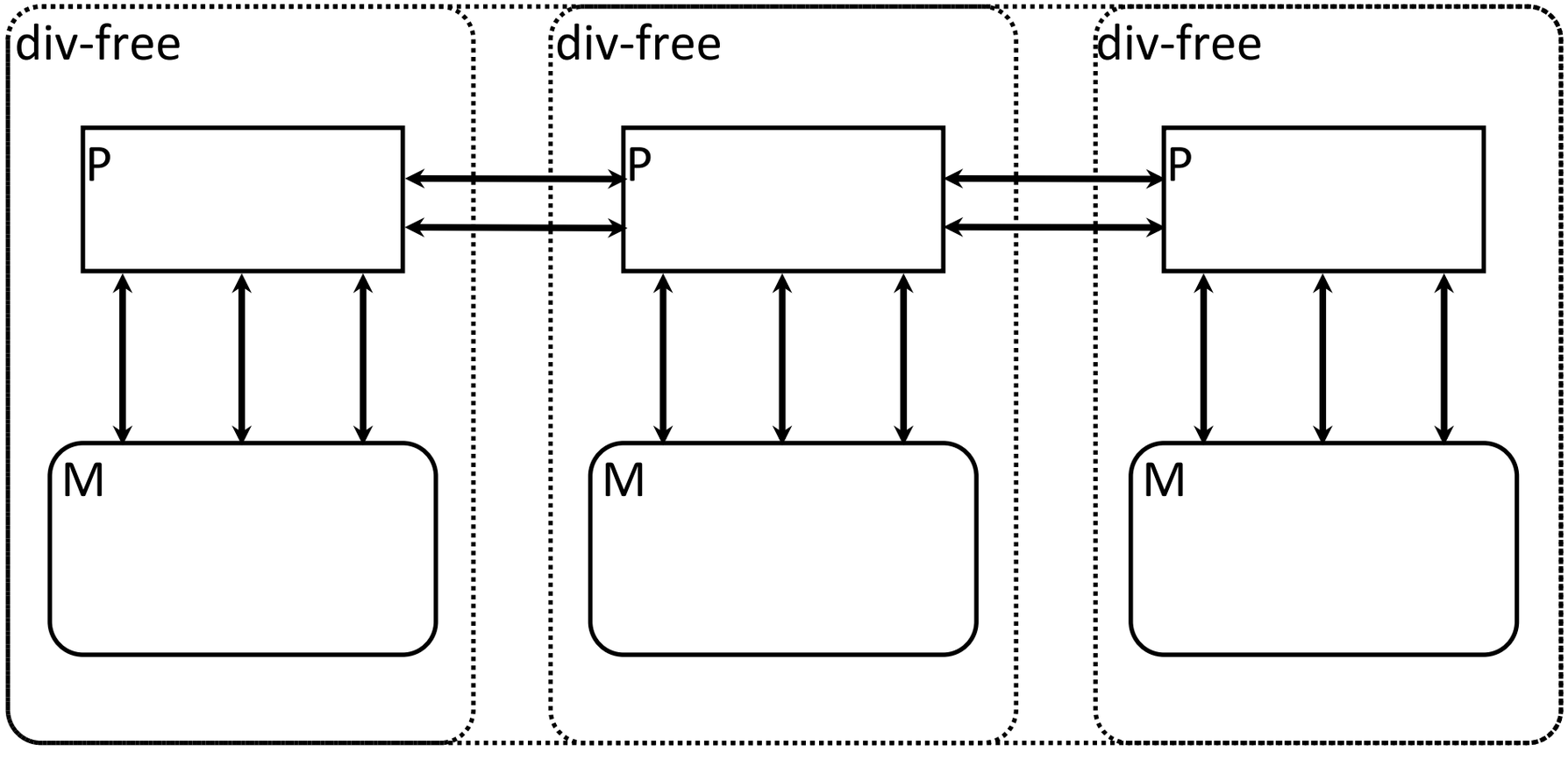
CSP||B and CSP||Event-B

- Various results, theorems and proof rules have been obtained for (combinations of) controlled components:
  - deadlock-freedom (conditions for just checking CSP)
  - divergence-freedom (control loop invariants)
  - refinement is compositional (particular proof rules available to maintain liveness of machines)
  - model-checking with ProB (invariant checking, LTL, CTL ...)

Composition results

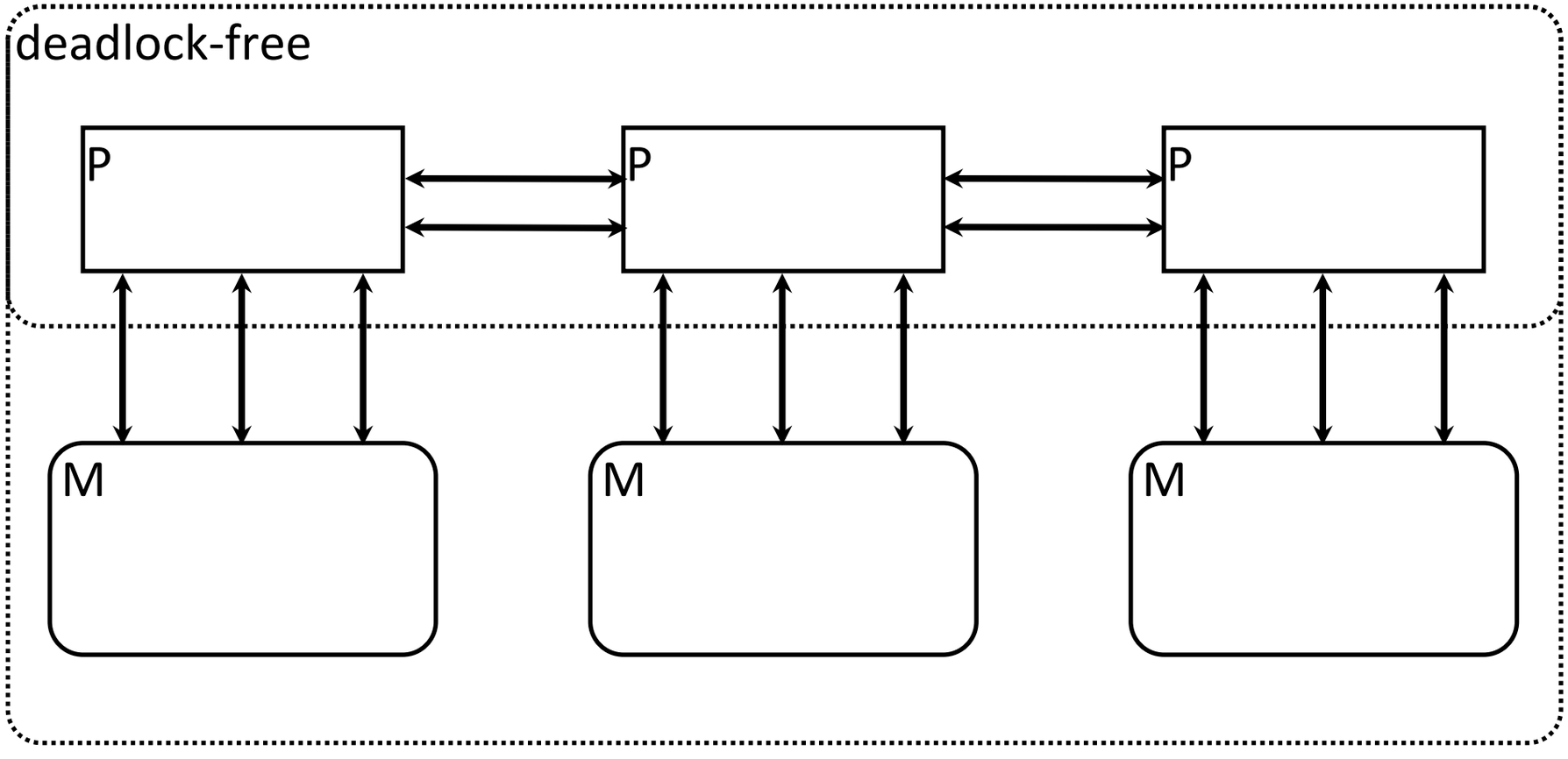
- If each CSP||B controlled component is divergence-free... *(B theorems available)*

  ...then so is their combination



Composition results

- If the combination of CSP controllers is deadlock-free... (…and the combination is divergence-free…)
  …then so is their CSP||B combination



Structure of Talk

- B and Event-B for state
- Combination with CSP
- **Example: modelling for railway safety**
- Incorporating time

Application to Railway modelling

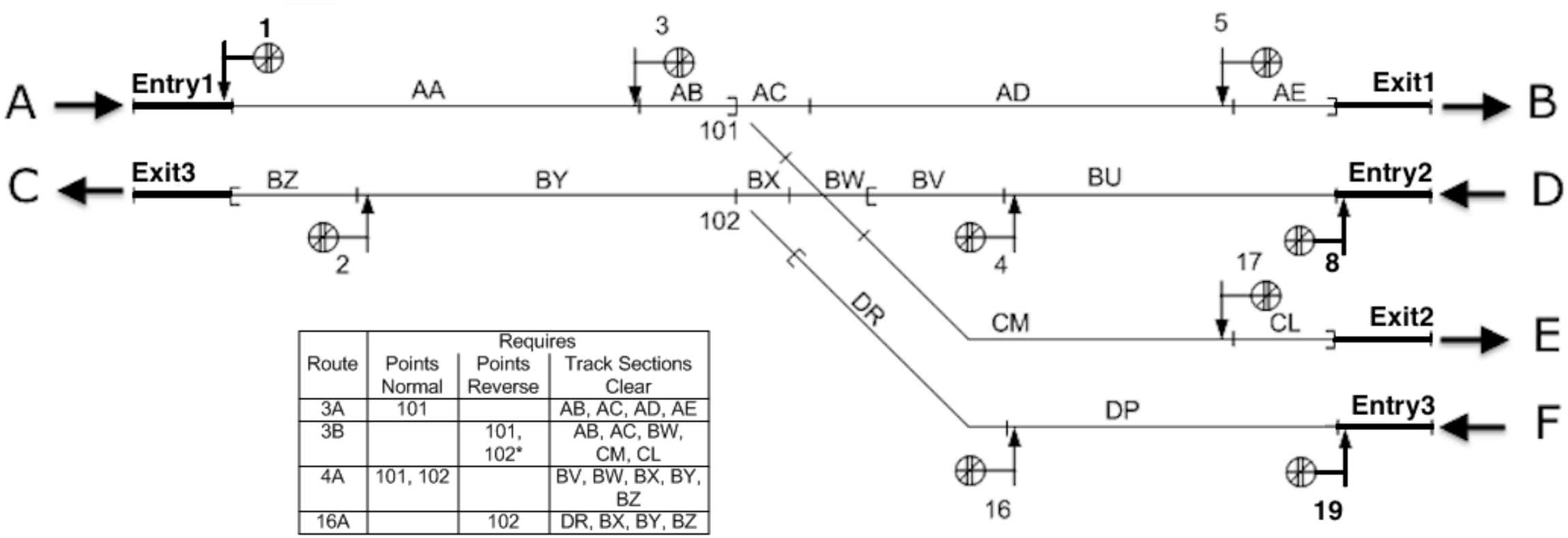
Steve Schneider
Helen Treharne
Matthew Trumble

UNIVERSITY OF SURREY

Markus Roggenbach
Faron Moller
Nga Hoang Nguyen
Phil James

Swansea University
Prifysgol Abertawe

Railway example track plan



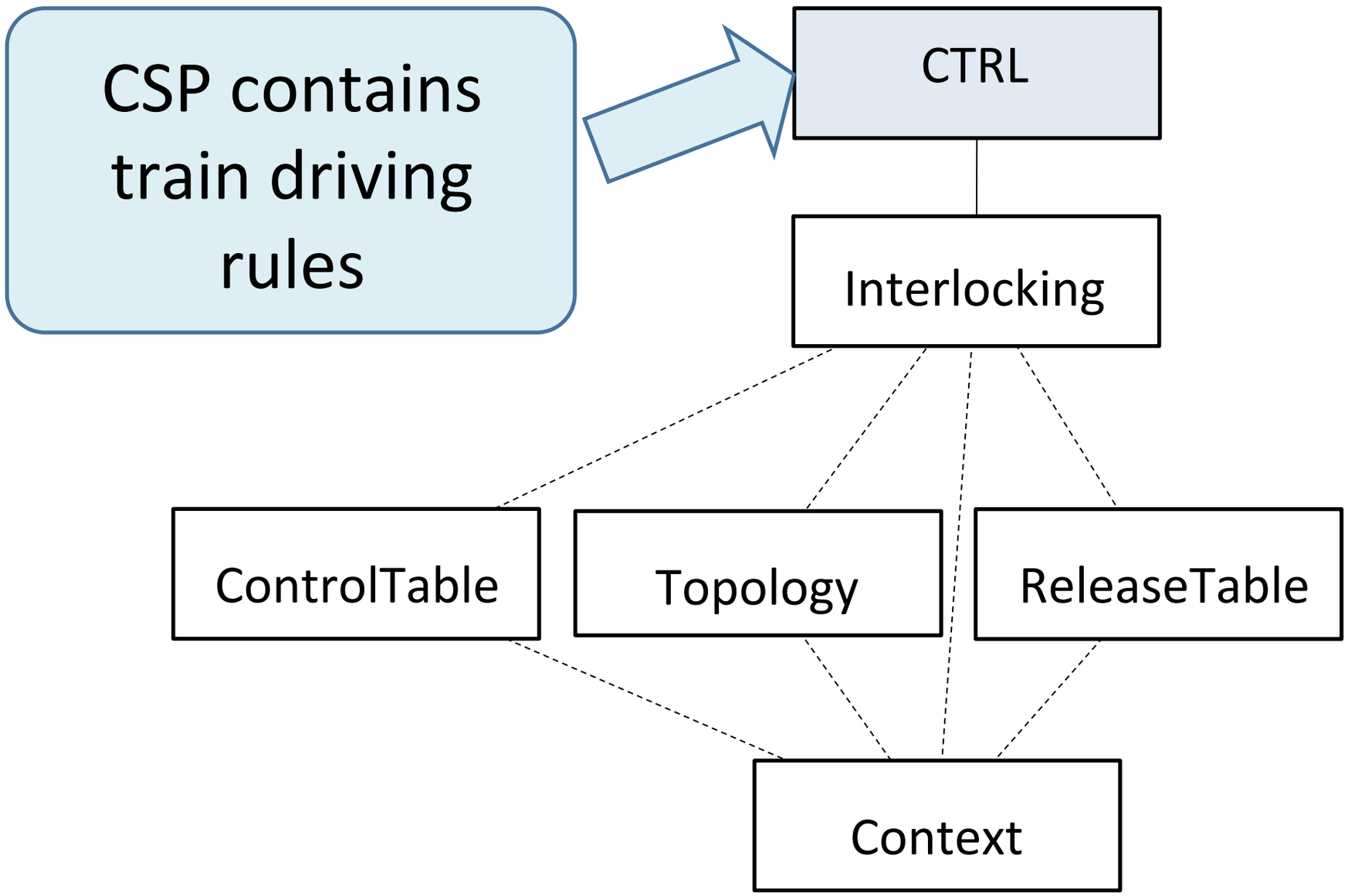
Route	Requires: Points Normal	Requires: Points Reverse	Requires: Track Sections Clear
3A	101		AB, AC, AD, AE
3B		101, 102*	AB, AC, BW, CM, CL
4A	101, 102		BV, BW, BX, BY, BZ
16A		102	DR, BX, BY, BZ

* flank protection

State:
train locations;
points positions;
signals; locks on routes

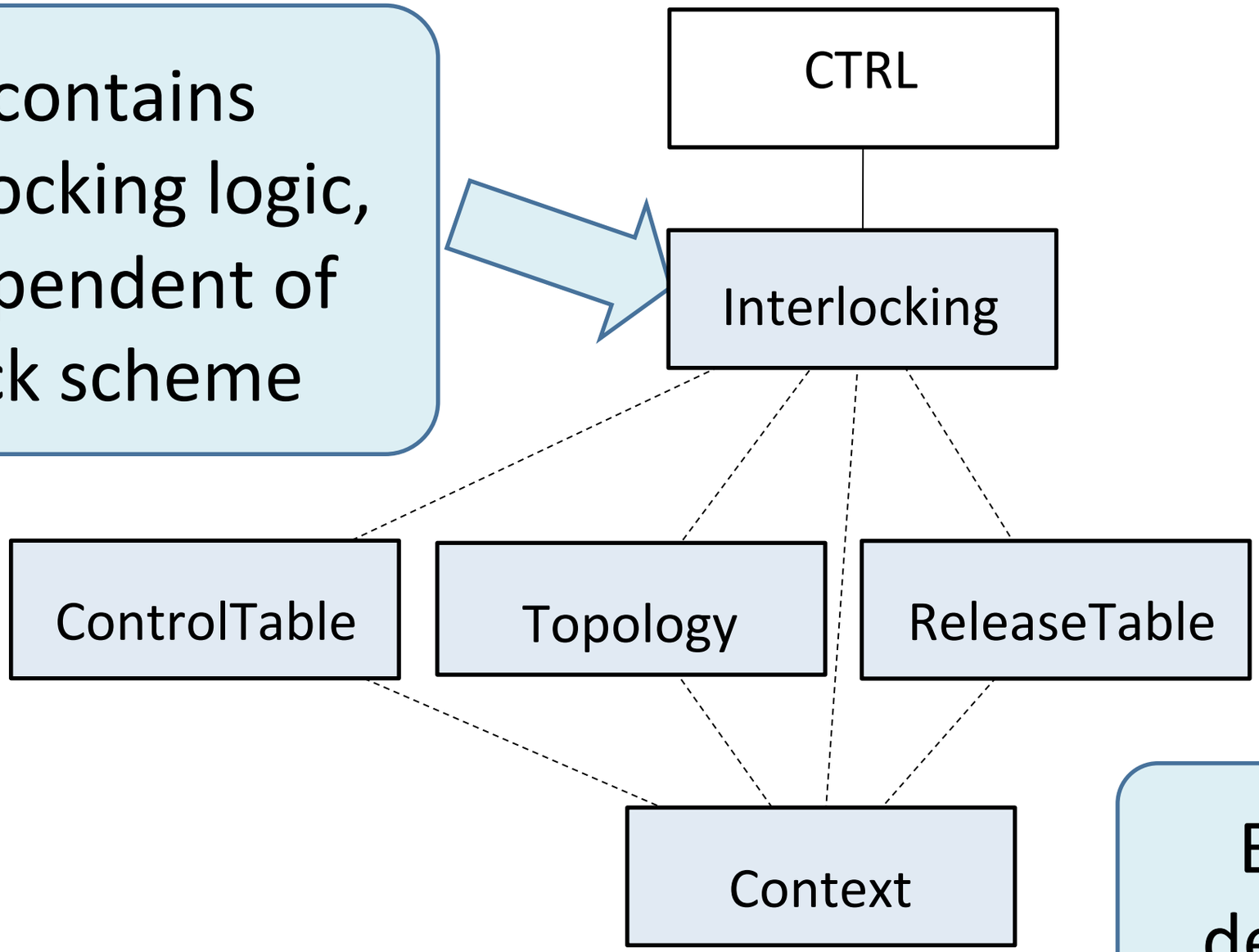
Context:
track topology;
control table;
lock table

CSP||B Architecture



CSP||B Architecture

B contains interlocking logic, independent of track scheme



B machines defining track scheme

Example of CSP model

$$RW\_CTRL =$$
$$\quad \sqcap_{r \in ROUTE} (request!r?b \rightarrow RW\_CTRL)$$
$$\quad \sqcap$$
$$\quad \sqcap_{r \in ROUTE} (release!r?b \rightarrow RW\_CTRL)$$

$$TRAIN\_OFF(t) = enter!t?newp \rightarrow TRAIN\_CTRL(t, newp)$$

$$TRAIN\_CTRL(t, pos) =$$
$$\quad pos \notin EXIT \wedge pos \in SIGNALHOMES \;\&\; nextSignal!t?aspect \rightarrow$$
$$\quad\quad \textbf{if } aspect == green$$
$$\quad\quad \textbf{then}$$
$$\quad\quad\quad move!t.pos?newp \rightarrow TRAIN\_CTRL(t, newp)$$
$$\quad\quad\quad \sqcap$$
$$\quad\quad\quad stay!t.pos \rightarrow TRAIN\_CTRL(t, pos)$$
$$\quad\quad \textbf{else}$$
$$\quad\quad\quad stay!t.pos \rightarrow TRAIN\_CTRL(t, pos)$$
$$\quad \Box$$
$$\quad pos \notin EXIT \wedge pos \notin SIGNALHOMES \;\&$$
$$\quad\quad move!t.pos?newp \rightarrow TRAIN\_CTRL(t, newp)$$
$$\quad\quad \sqcap$$
$$\quad\quad stay!t.pos \rightarrow TRAIN\_CTRL(t, pos)$$
$$\quad \Box \ \ldots$$

$$ALL\_TRAINS = |||_{t \in TRAIN} \; TRAIN\_OFF(t)$$

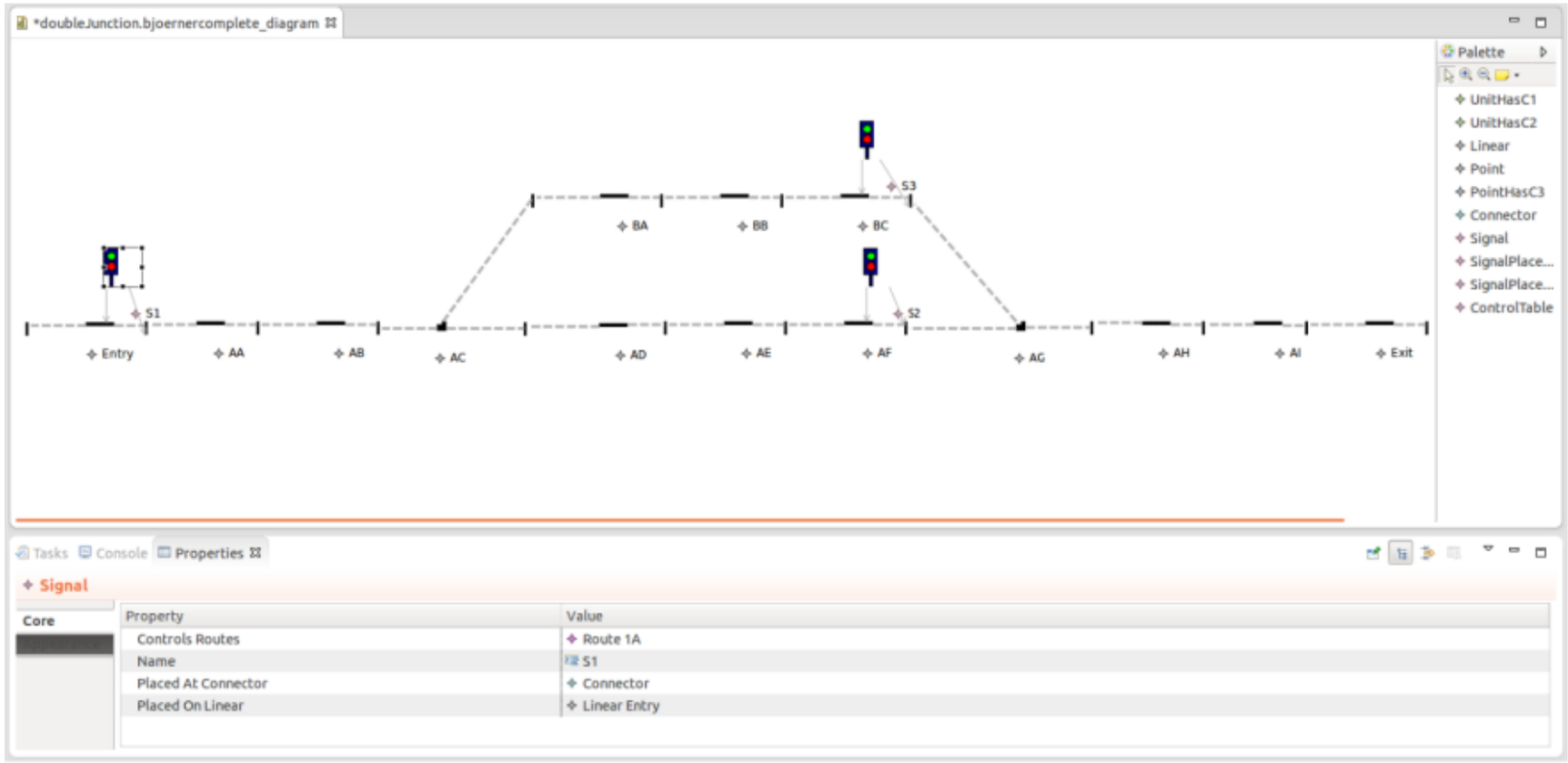
$$move!t.pos?newp \rightarrow \ldots$$

Example of B Interlocking operation

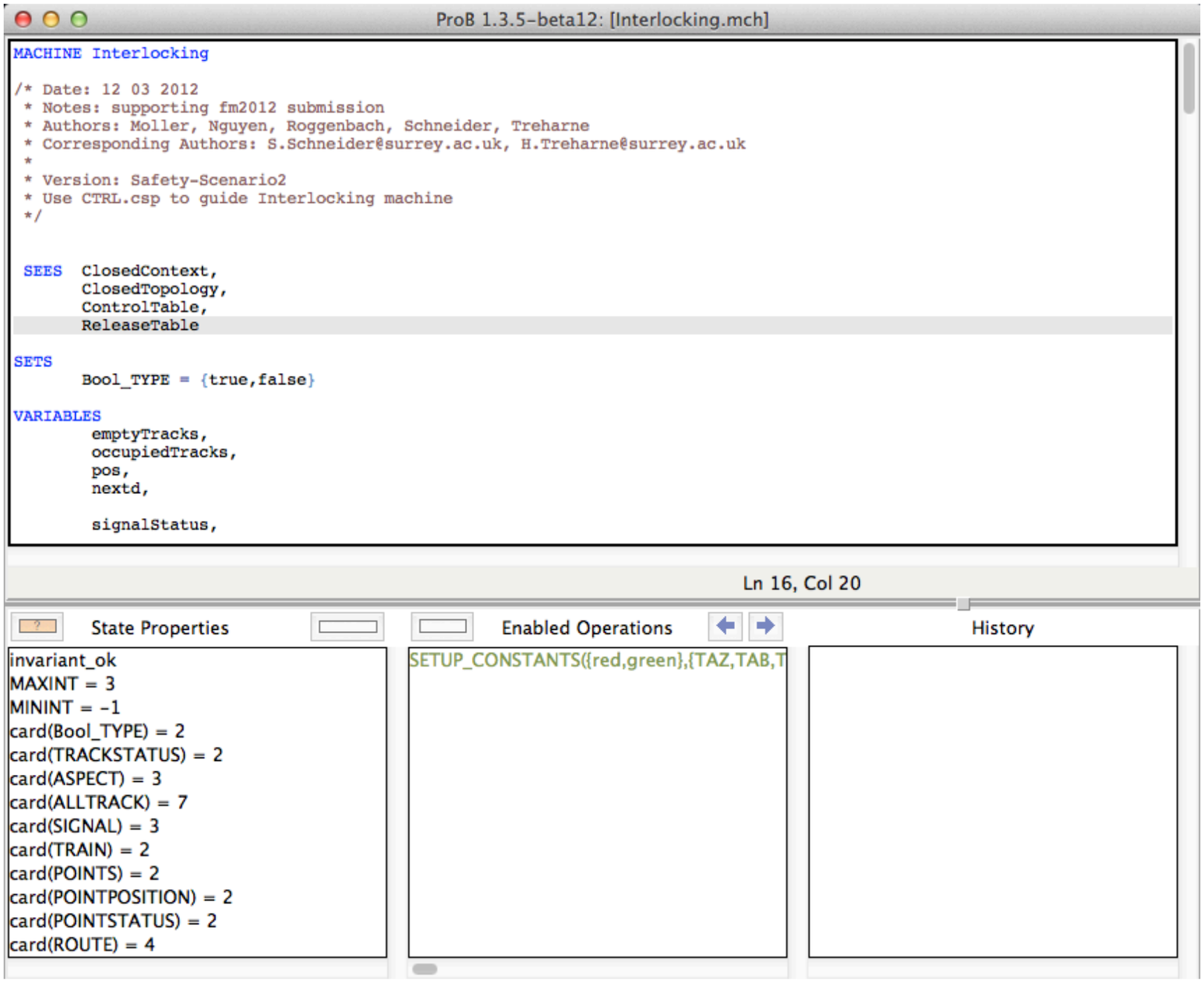
```
currp, newp <-- move(t) =
PRE t : TRAIN & t : dom(pos)
THEN
   IF (pos(t) /: dom(nextd)) THEN
                LET track BE
                  track = nullTrack
                IN
      currp := pos(t) ||
                  pos(t) := track ||
                  newp := track
                END
   ELSE
                LET track BE
                  track = nextd(pos(t))
                IN
                  currp := pos(t) ||
                  pos(t) := track ||
                  newp := track ||
                  currentLocks := currentLocks - releaseTable[{track}] ||
                  IF (pos(t) : ran(homeSignal))
                  THEN
                     signalStatus( homeSignal~(pos(t))) := red
                  END
                END
   END
END;
```

OnTrack: capturing track plans

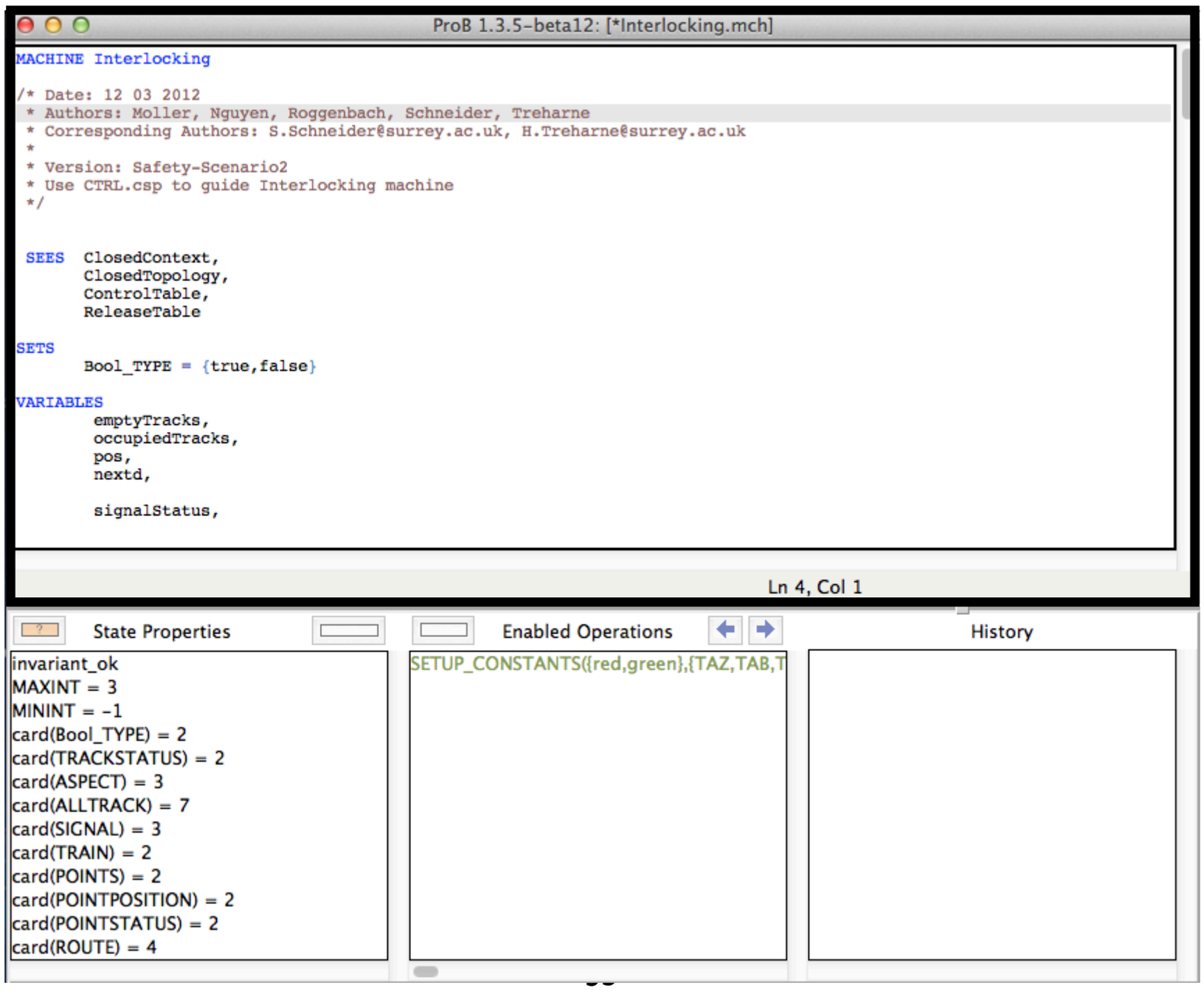
- Model-Driven Engineering: translating railway domain concepts into CSP||Event-B models



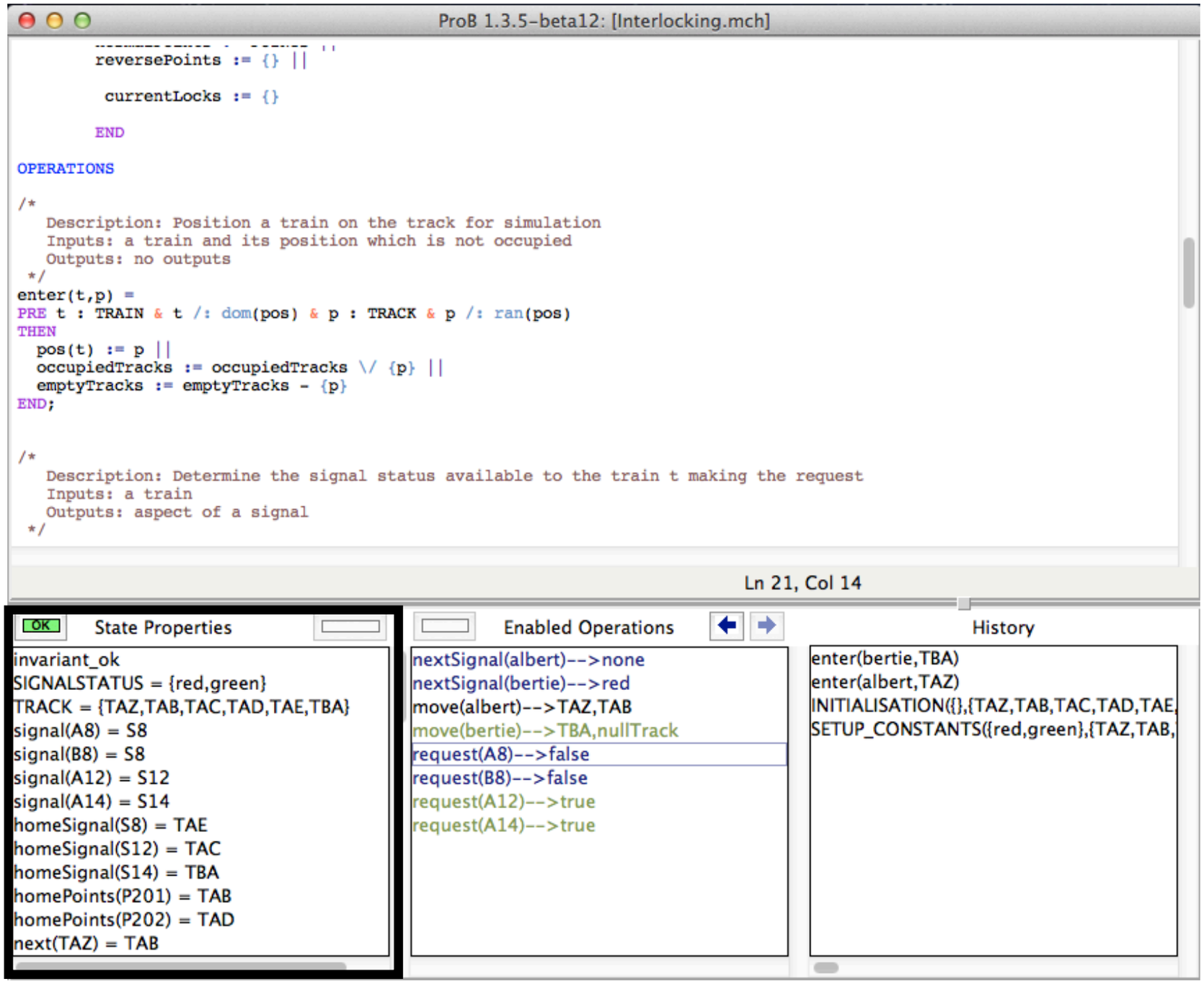
ProB model checker for B (Leuschel) can check CSP||B combinations



The Model panel



State and invariant panel



Enabled operations

ProB 1.3.5-beta12: [Interlocking.mch]

```
        reversePoints := {} ||

         currentLocks := {}

        END

OPERATIONS

/*
   Description: Position a train on the track for simulation
   Inputs: a train and its position which is not occupied
   Outputs: no outputs
 */
enter(t,p) =
PRE t : TRAIN & t /: dom(pos) & p : TRACK & p /: ran(pos)
THEN
  pos(t) := p ||
  occupiedTracks := occupiedTracks \/ {p} ||
  emptyTracks := emptyTracks - {p}
END;


/*
   Description: Determine the signal status available to the train t making the request
   Inputs: a train
   Outputs: aspect of a signal
 */
```

Ln 21, Col 14

OK State Properties

```
invariant_ok
SIGNALSTATUS = {red,green}
TRACK = {TAZ,TAB,TAC,TAD,TAE,TBA}
signal(A8) = S8
signal(B8) = S8
signal(A12) = S12
signal(A14) = S14
homeSignal(S8) = TAE
homeSignal(S12) = TAC
homeSignal(S14) = TBA
homePoints(P201) = TAB
homePoints(P202) = TAD
next(TAZ) = TAB
```

Enabled Operations

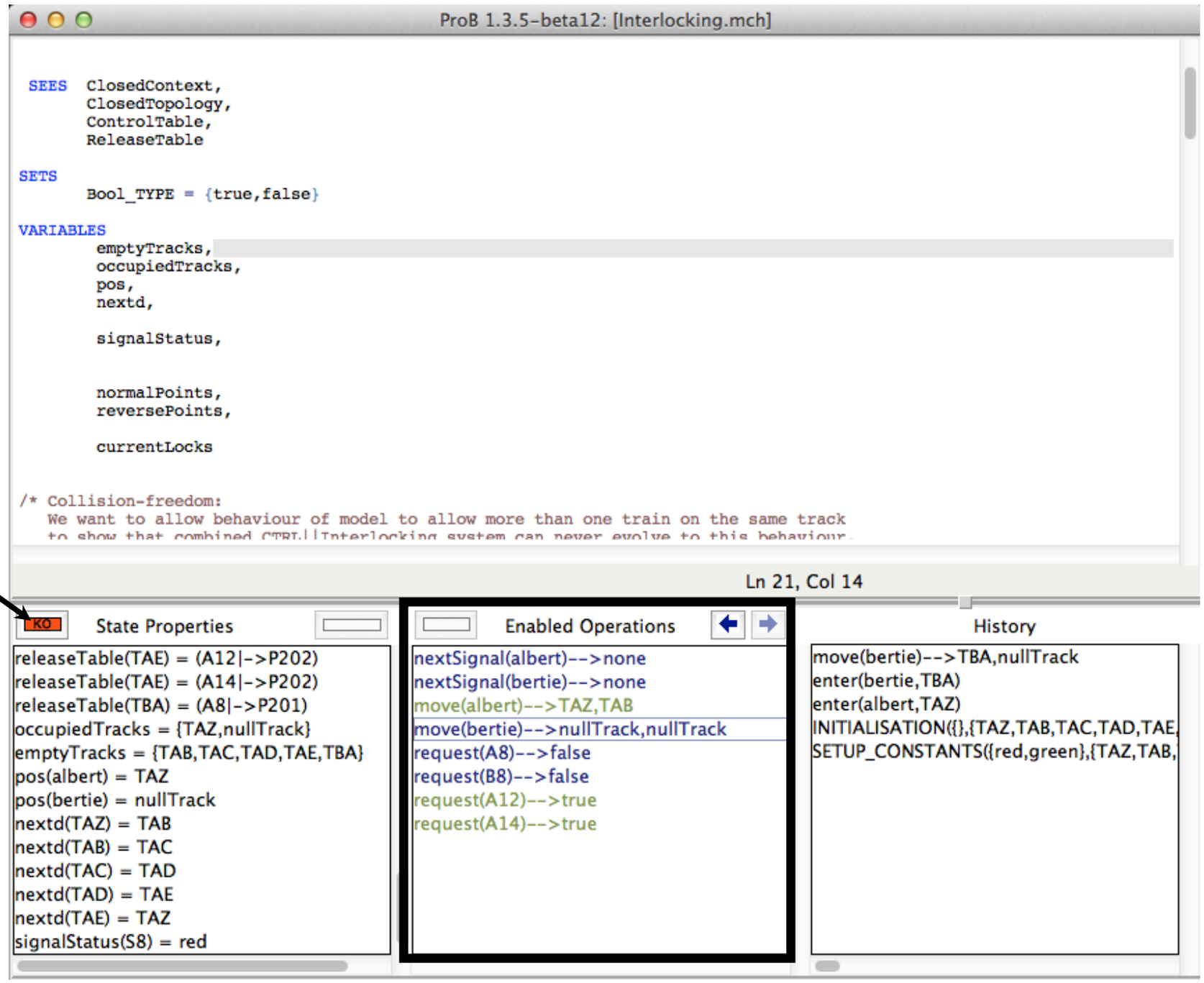
```
nextSignal(albert)-->none
nextSignal(bertie)-->red
move(albert)-->TAZ,TAB
move(bertie)-->TBA,nullTrack
request(A8)-->false
request(B8)-->false
request(A12)-->true
request(A14)-->true
```

History

```
enter(bertie,TBA)
enter(albert,TAZ)
INITIALISATION({},{TAZ,TAB,TAC,TAD,TAE,
SETUP_CONSTANTS({red,green},{TAZ,TAB,
```

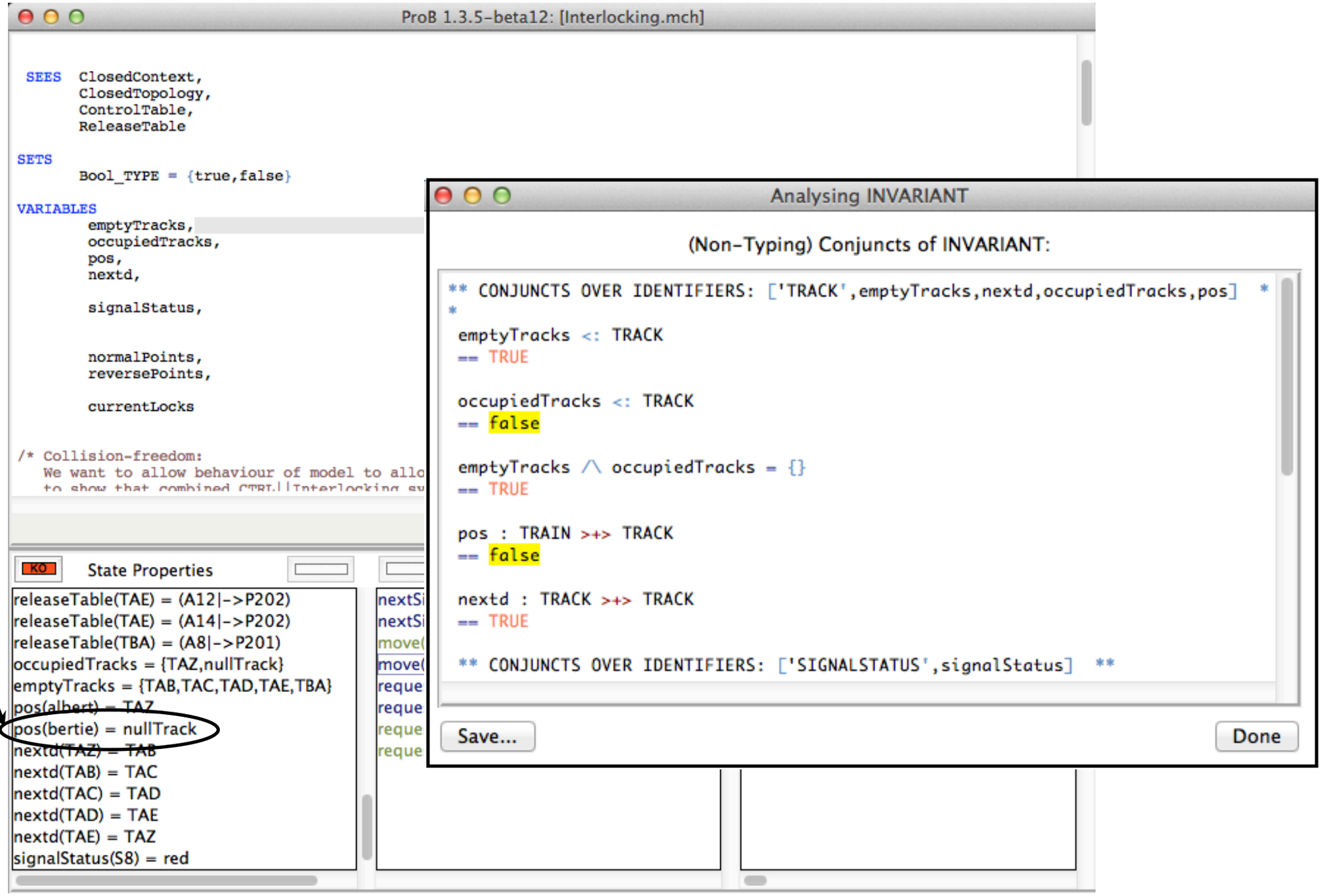
If train moves through a red light…

invariant violation

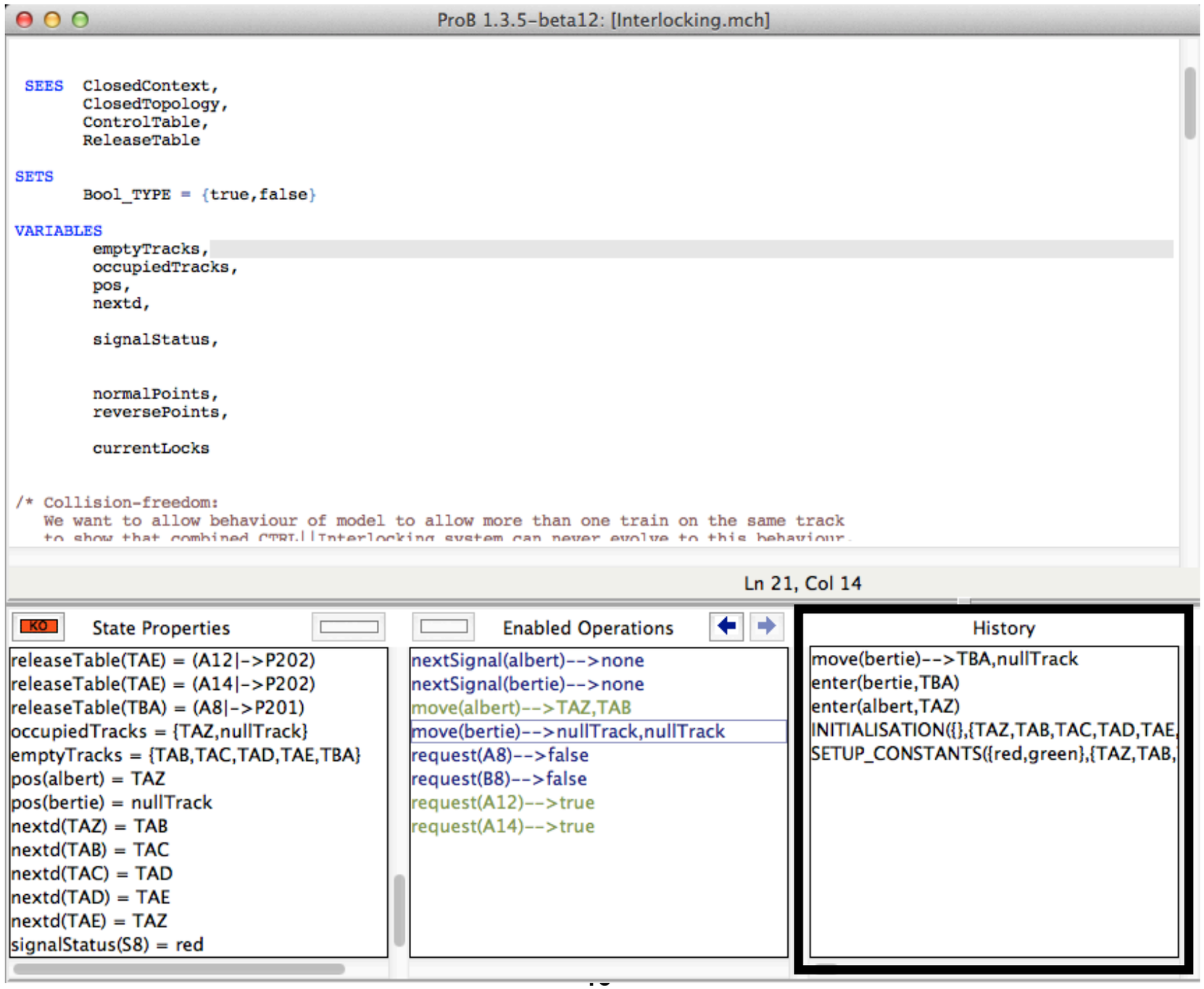


Analysing the invariant

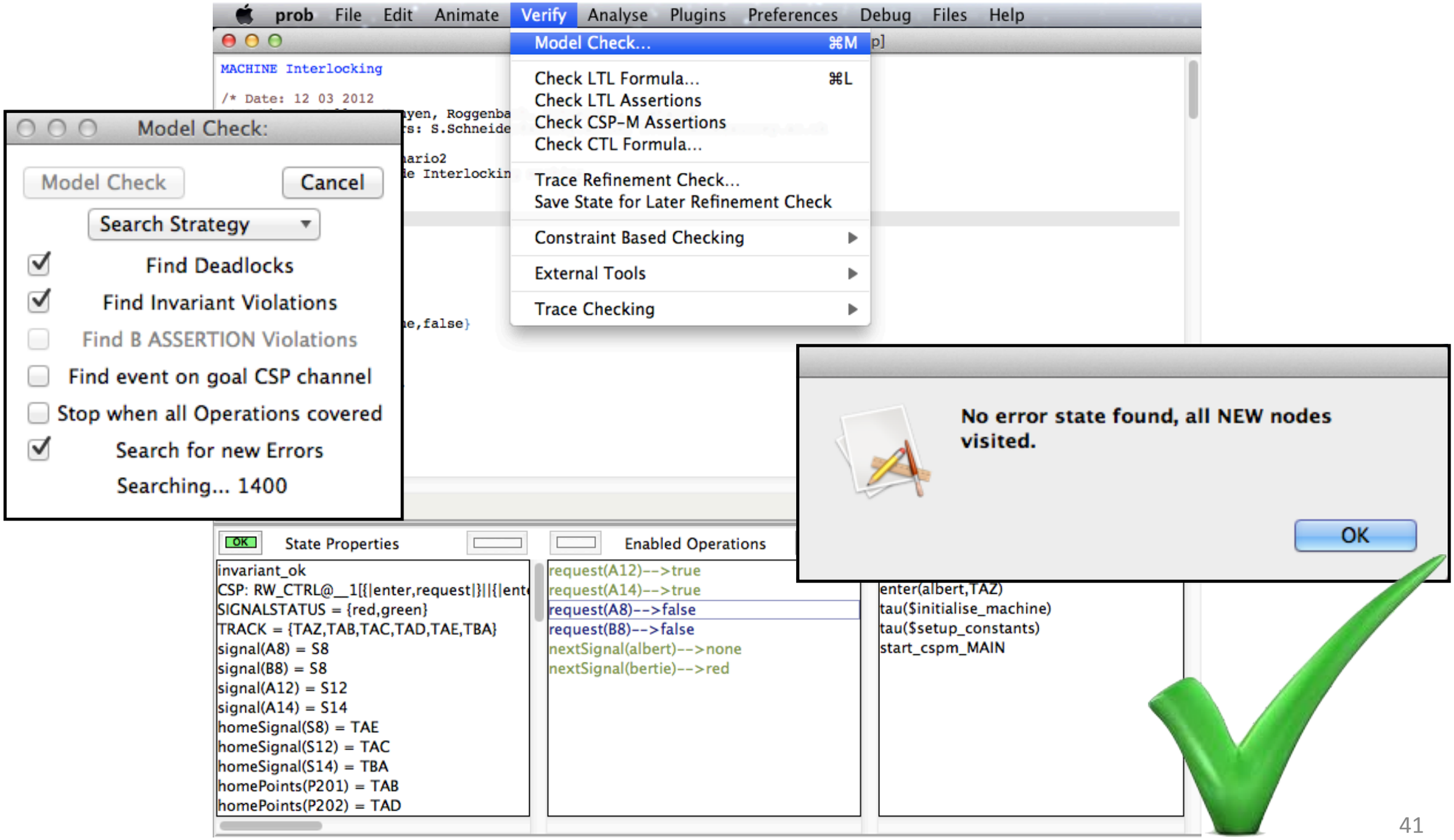
invariant violation



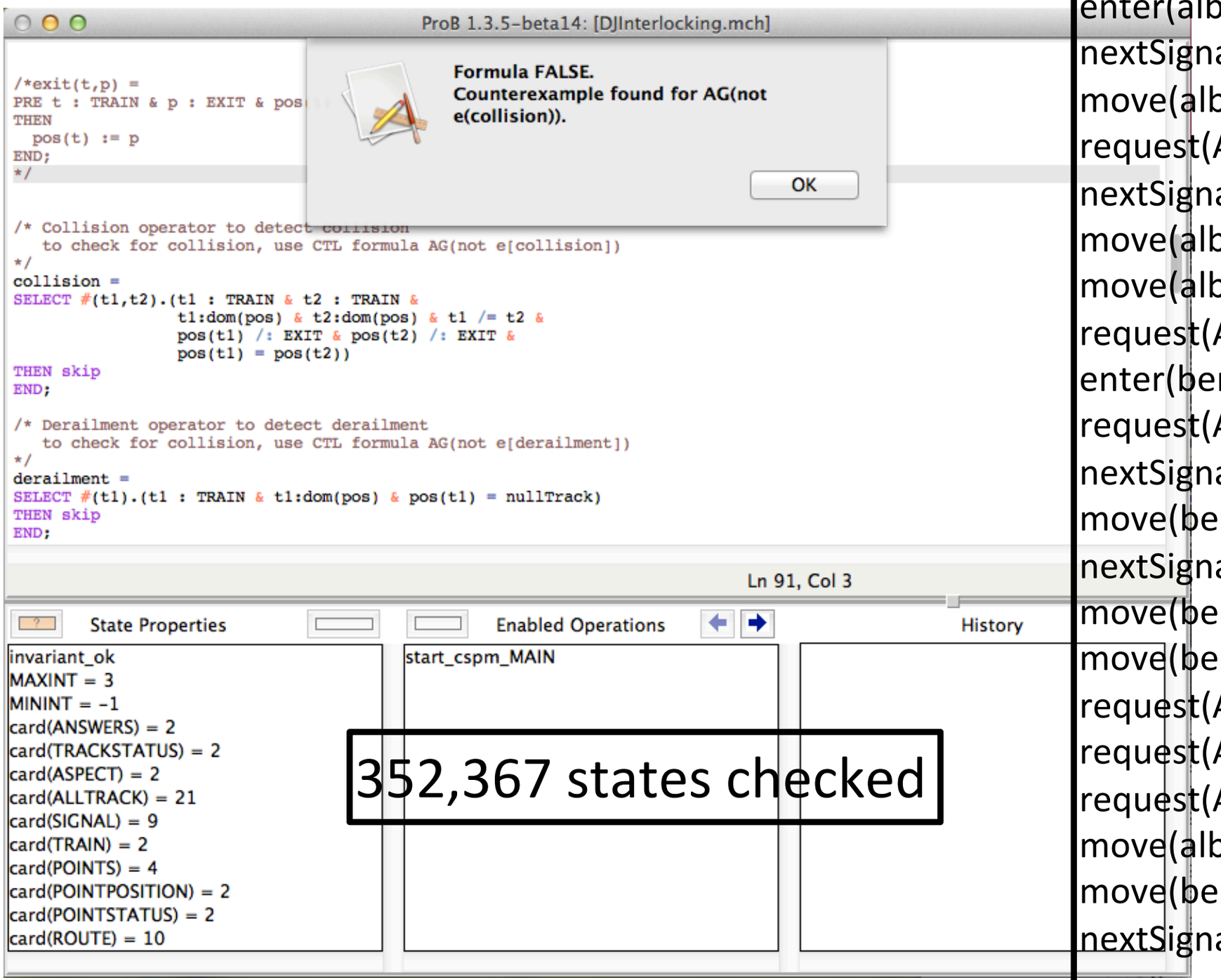
History panel: operations performed so far



Checking safety conditions



Scenario - Faulty Tracks in "ClearTable"



request(A1)-->yes
enter(albert,Entry1)
nextSignal(albert)-->green
move(albert)-->Entry1,AA
request(A3)-->yes
nextSignal(albert)-->green
move(albert)-->AA,AB
move(albert)-->AB,AC
request(A1)-->yes
enter(bertie,Entry1)
request(A3)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry1,AA
nextSignal(bertie)-->green
move(bertie)-->AA,AB
move(bertie)-->AB,AC
request(A16)-->yes
request(A5)-->yes
request(A2)-->yes
move(albert)-->AC,AD
move(bertie)-->AC,AD
nextSignal(albert)-->green

Where we're at

- Engineers understand/agree with the models (train driving rules; track interlocking rules)
- The combined semantics also provides a foundation for compositional results on complex track plans:
  - Finitisation: limiting the number of trains to include in a safety analysis
  - Coverings: breaking down a complex track plan into manageable fragments that can be independently checked

Structure of Talk

- B and Event-B for state
- Combination with CSP
- Example: modelling for railway safety
- **Incorporating time**

Introducing time

railway designers are also interested in network capacity

Challenge: Introducing time

- We want to introduce time to CSP||Event-B
- Natural and obvious approach: use Timed CSP as the control language and manage all the time in there
  - leave the B models as untimed, embedded in the timed CSP semantic framework
- … but not quite so simple...

Timed CSP

- Timed language:

$$
\begin{aligned}
P \quad ::= \quad & STOP \mid SKIP \mid WAIT\ t \mid P;Q \mid \\
& a \rightarrow P \mid c?x \rightarrow P \mid c!v \rightarrow P \mid \\
& P \sqcap Q \mid P \,\Box\, Q \mid P \overset{t}{\triangleright} Q \mid \\
& P \parallel Q \mid P \,|||\, Q \mid P \setminus A \\
& X \mid \mu X \,.\, P
\end{aligned}
$$

- (recursions time-guarded)

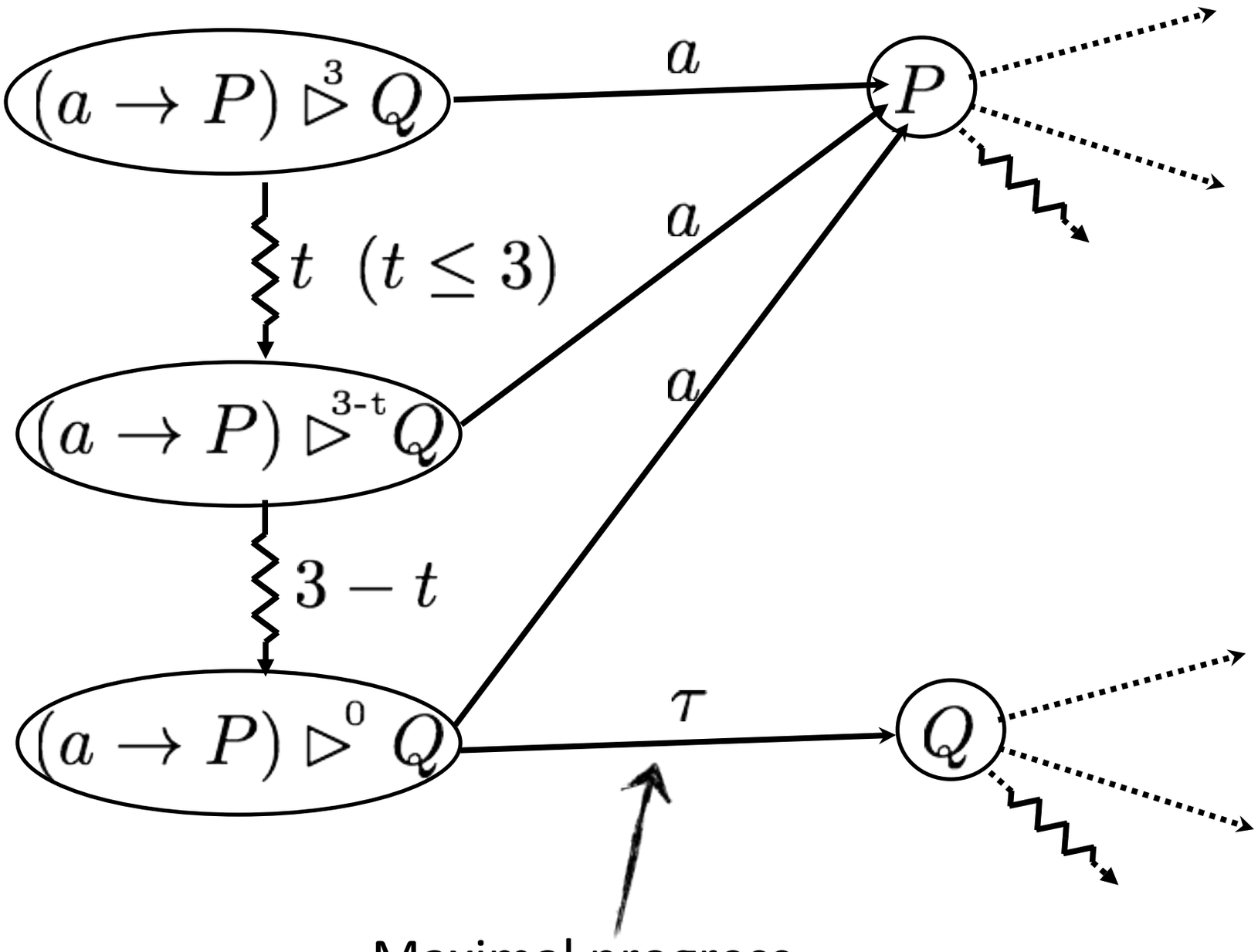
Timed CSP transitions

$$(a \to P) \xrightarrow{a} P$$

$$(a \to P) \overset{2}{\rightsquigarrow} (a \to P)$$

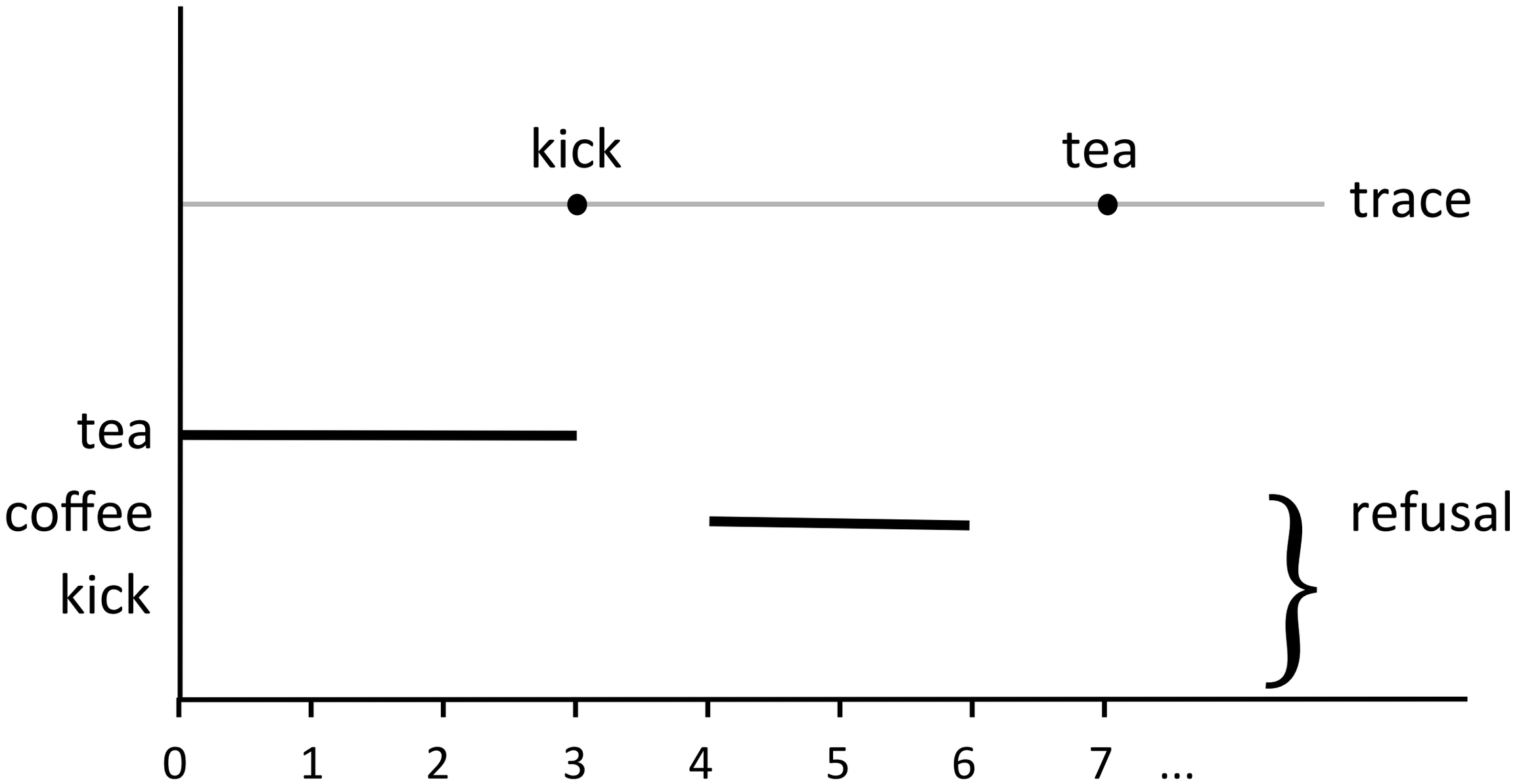
$$(P \overset{7}{\triangleright} Q) \overset{2}{\rightsquigarrow} (P' \overset{5}{\triangleright} Q)$$

Example fragment of a timed transition system



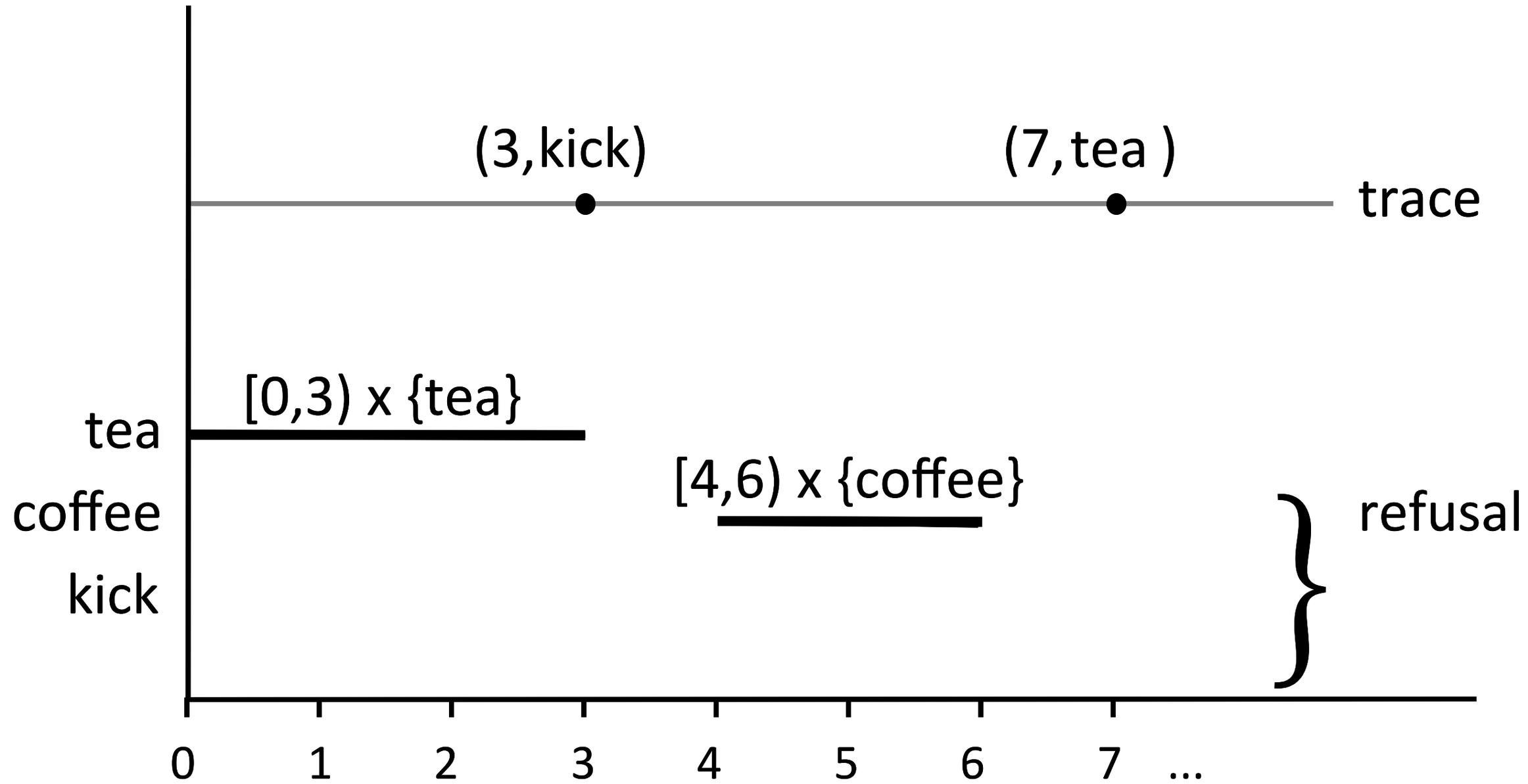
Observations for denotational semantics: Timed Failures

$$(coffee \rightarrow STOP) \,\square\, (kick \rightarrow tea \rightarrow STOP)$$



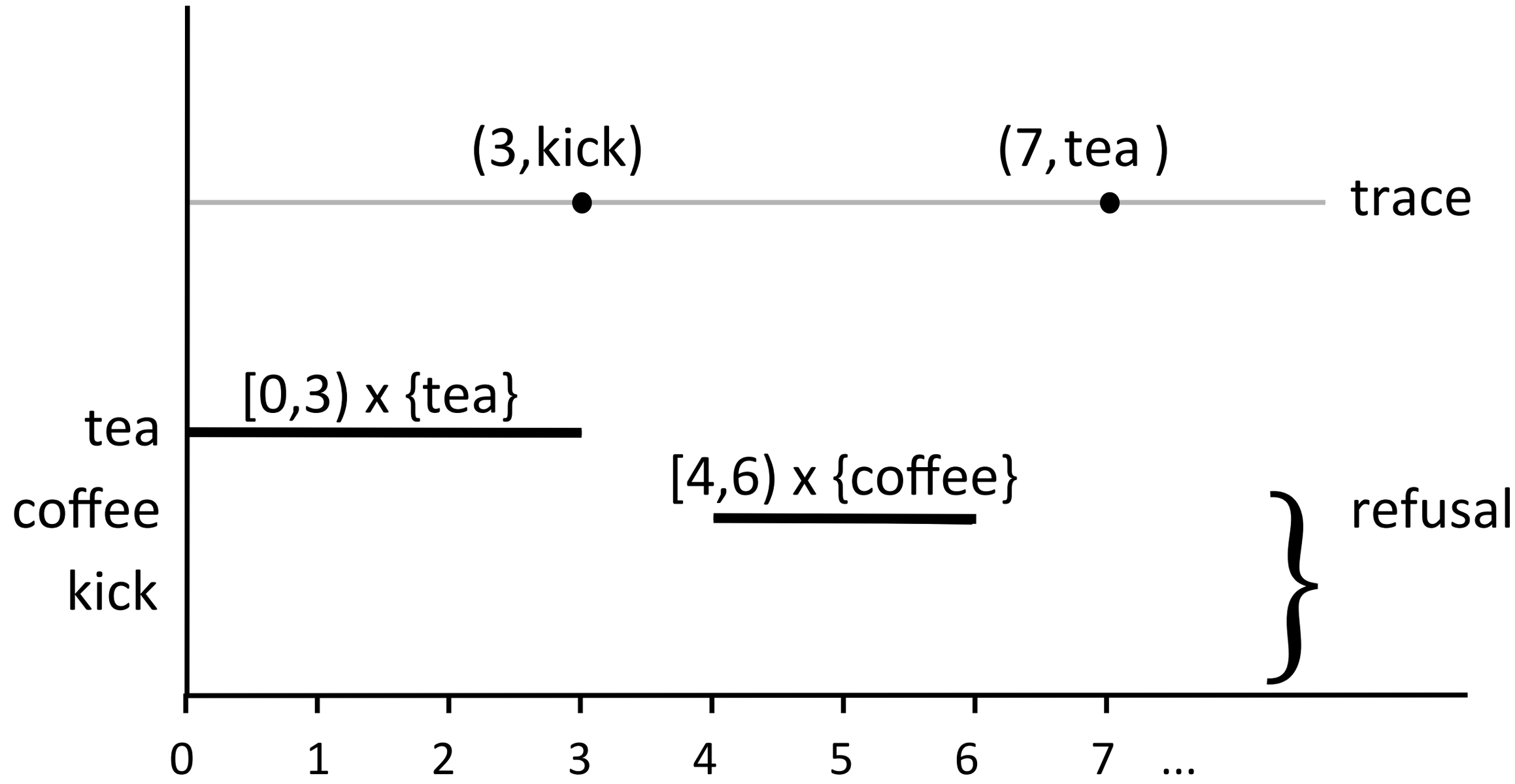
Observations for denotational semantics: Timed Failures

$$(coffee \rightarrow STOP) \,\square\, (kick \rightarrow tea \rightarrow STOP)$$



Observations for denotational semantics: Timed Failures

$$\langle\ [0,3) \times \{tea\}\ ,\ (3, kick)\ ,\ [4,6) \times \{coffee\}\ ,\ (7, tea)\ ,\ \emptyset\ \rangle$$



Timed CSP and B/Event-B: issues

- "leave the B models as untimed, embedded in the timed CSP semantic framework"

- 1. wp semantics doesn't give timed CSP semantics (even for "untimed" machines)
  - no refusals during the trace
- 2. machines react and change state instantly
  - they are not "time guarded"
- 3. need to include divergence (abort) in machines (for B, but not for Event-B)

1. Timed refusal testing

$$P1 \quad = \quad (coffee \rightarrow STOP) \,\Box\, (kick \rightarrow tea \rightarrow STOP)$$
$$\sqcap\, (tea \rightarrow STOP) \,\Box\, (kick \rightarrow coffee \rightarrow STOP)$$

$$P2 \quad = \quad (coffee \rightarrow STOP) \,\Box\, (kick \rightarrow coffee \rightarrow STOP)$$
$$\sqcap\, (tea \rightarrow STOP) \,\Box\, (kick \rightarrow tea \rightarrow STOP)$$

1. Timed refusal testing

$$P1 \quad = \quad (coffee \rightarrow STOP) \,\Box\, (kick \rightarrow tea \rightarrow STOP)$$
$$\sqcap\, (tea \rightarrow STOP) \,\Box\, (kick \rightarrow coffee \rightarrow STOP)$$

$$P2 \quad = \quad (coffee \rightarrow STOP) \,\Box\, (kick \rightarrow coffee \rightarrow STOP)$$
$$\sqcap\, (tea \rightarrow STOP) \,\Box\, (kick \rightarrow tea \rightarrow STOP)$$

$$\langle\, [0,3) \times \{tea\}\ ,\ (3, kick)\ ,\ [4,6) \times \{coffee\}\ ,\ (7, tea)\ ,\ \emptyset\, \rangle$$

Timed CSP semantics?
Now need to distinguish these two:

Machine VM1 =
Variables n
Invariants n ∈ {0,1,2}
Initialisation n:=0 ⊓ n:=1
Operations
tea =
  when n=0 then n:=2 end
coffee =
  when n = 1 then n:=2 end
kick =
  when n<2 then n:=1-n end

Machine VM2 =
Variables n
Invariants n ∈ {0,1,2}
Initialisation n:=0 ⊓ n:=1
Operations
tea =
  when n=0 then n:=2 end
coffee =
  when n = 1 then n:=2 end
kick =
  when n<2 then skip end

$$P = (tea \rightarrow STOP) \stackrel{3}{\triangleright} (kick \rightarrow tea \rightarrow STOP)$$

$P \parallel VM1$ can't refuse tea for ever, $P \parallel VM2$ can

Untimed refusal traces and wp

$$\langle X_0, a_1, X_1, a_2, X_2, \ldots \rangle \in refusal\ traces(M) \iff$$

$\exists\ P_0\ ,\ P_1\ ,\ \ldots$

1. $\overline{wp}(init, P_0)$

$P_0 \Rightarrow \overline{wp}(a_1, P_1)$

$\vdots$

$P_i \Rightarrow \overline{wp}(a_{i+1}, P_{i+1})$

2. $\text{for each } i\text{: } P_i \Rightarrow \bigwedge_{a \in X_i} \neg g_a$

Example: $\langle\ \{tea\}\ ,\ kick\ ,\ \emptyset\ ,\ tea\ ,\ \emptyset \rangle$

$\overline{wp}(init, P_0)$

$P_0 \Rightarrow \overline{wp}(kick, P_1)$

$P_1 \Rightarrow \overline{wp}(tea, P_2)$

$P_0:\ \ n = 1$	$\neg g_{tea}:\ \ n \neq 0$
$P_1:\ \ n = 0$	$\bigwedge_{a \in X_1} \neg g_a:\ \ true$
$P_2:\ \ true$	$\bigwedge_{a \in X_2} \neg g_a:\ \ true$

Defining timed refusals via wp: relating timed and untimed observations

$$\langle\ \aleph_0\ ,\ (t_1, a_1)\ ,\ \aleph_1\ ,\ (t_2, a_2)\ ,\ \aleph_2\ \ldots\ \rangle \in timed\ failures(M) \iff$$

$$\langle\ \sigma(\aleph_0)\ ,\ a_1\ ,\ \sigma(\aleph_1)\ ,\ a_2\ ,\ \sigma(\aleph_2)\ \ldots\ \rangle \in refusal\ traces(M)$$

Just the events; times removed

NB: True for "untimed" B machines, because state changes are instant and states are stable between transitions.
Not true for arbitrary timed CSP processes

$$\langle\ [0,3) \times \{tea\}\ ,\ (3, kick)\ ,\ [4,6) \times \{coffee\}\ ,\ (7, tea)\ ,\ \emptyset\ \rangle$$

$$\langle\ \{tea\}\ ,\ kick\ ,\ \{coffee\}\ ,\ tea\ ,\ \emptyset\ \rangle$$

Timed CSP: Fixed Point theory

- Reed&Roscoe 1988 (+ see Schneider 1999):
  - Semantic model is a complete metric space
    - » d(P,Q) is 2^{-t} , where P and Q first differ at time t
  - Recursions are time guarded
  - Recursions are contraction mappings, giving unique fixed points
  - No divergences (processes don't go infinitely fast). Infinite internal transitions (e.g. from internal loops) take infinite time.

$$\mu X \,.\, (send!v \rightarrow (rec?ack \rightarrow STOP) \stackrel{3}{\triangleright} X)$$

$$\mu X \,.\, (rec1?x \rightarrow Q) \stackrel{3}{\triangleright} ((rec2?x \rightarrow R) \stackrel{3}{\triangleright} X)$$

Timed CSP semantics?
Instant response

Machine VM1 =
Variables n
Invariants n ∈ {0,1,2}
Initialisation n:=0 ⊓ n:=1
Operations
tea =
  when n=0 then n:=2 end
coffee =
  when n = 1 then n:=2 end
kick =
  when n<2 then n:=1-n end

$$
\begin{aligned}
M0 &= tea \rightarrow STOP \,\Box\, kick \rightarrow M1 \\
M1 &= coffee \rightarrow STOP \,\Box\, kick \rightarrow M0 \\
VM1 &= M0 \sqcap M1
\end{aligned}
$$

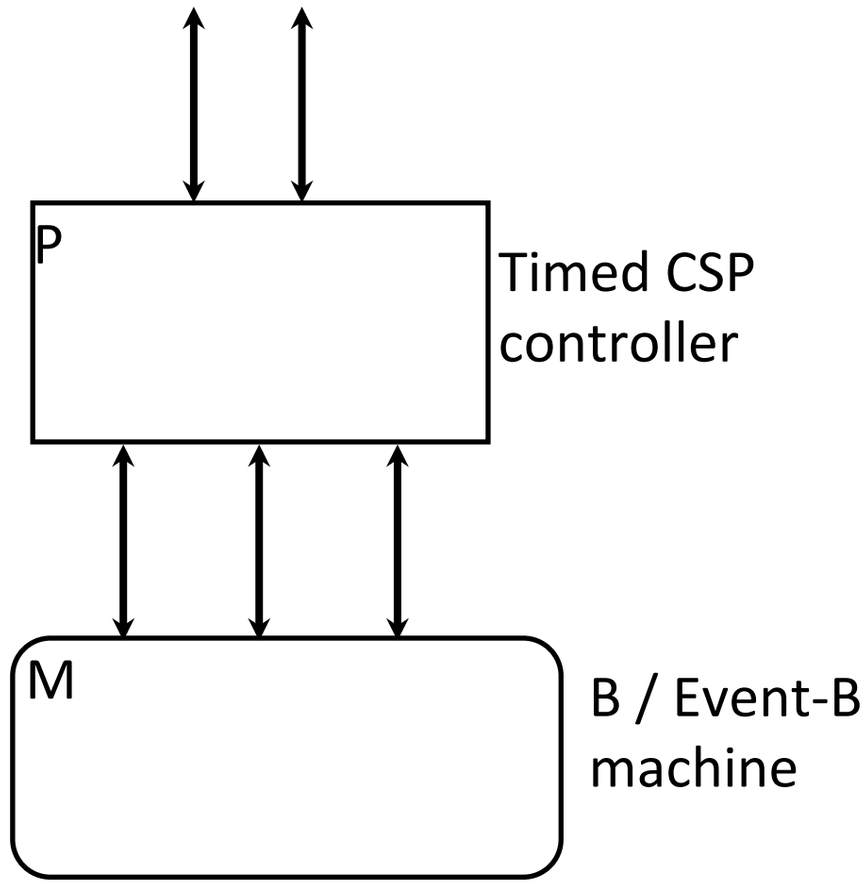
$$VM1 \backslash \{kick\}$$

A more recent semantics for timed CSP... ...does what we need

- Ouaknine&Worrell, "Timed CSP = Closed Timed ε-Automata", Nordic Journal of Computing 10 (2), 2003.
- Gives a more elaborate semantics for Timed CSP, including the following features in the language:
  - Arbitrarily fast processes (non-time-guarded recursions permitted)
  - Divergences (infinite tau loops in zero time), zeno processes?
  - Timestops and urgent events
- Semantics
  - cpo rather than cms
  - operational semantics (claimed to match denotational)

Combining Timed CSP and B / Event-B

- P and M both have a Ouaknine/ Worrell timed failures semantics.
- So P||M also has a semantics.
- The B machine can go arbitrarily fast...
- ...but in practice is driven by the Timed CSP controller, which we can make sure (and verify) is well-behaved: no divergences, no timestops, time-guarded loops.

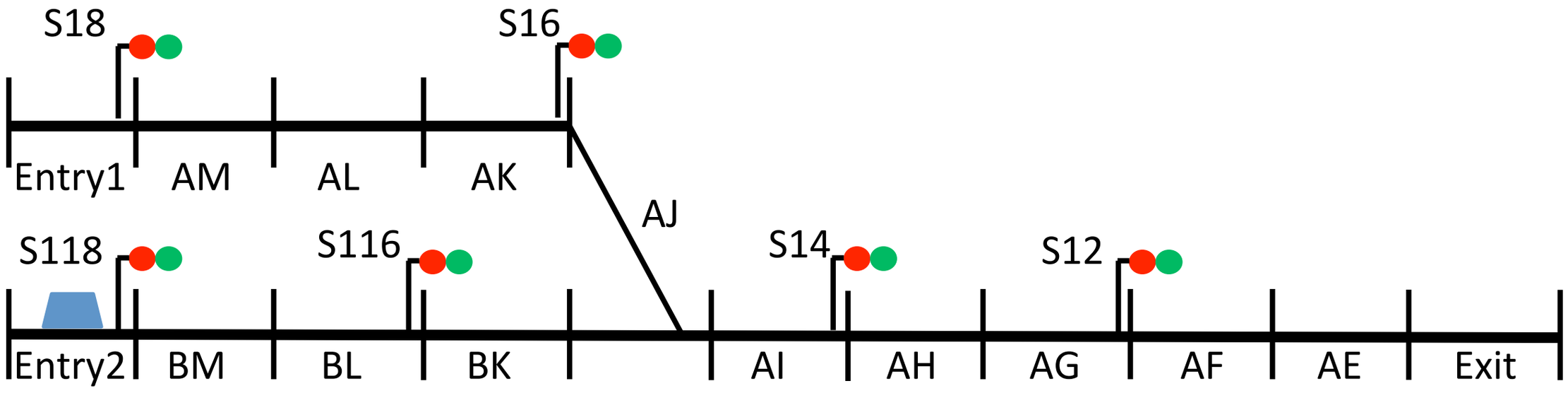


Further Challenges and Questions

- What are the best ways to manage (large) state with concurrency?
- Time? Probabily? Mobility ??
- Hierarchical use
- Pragmatics: more integrated tool support

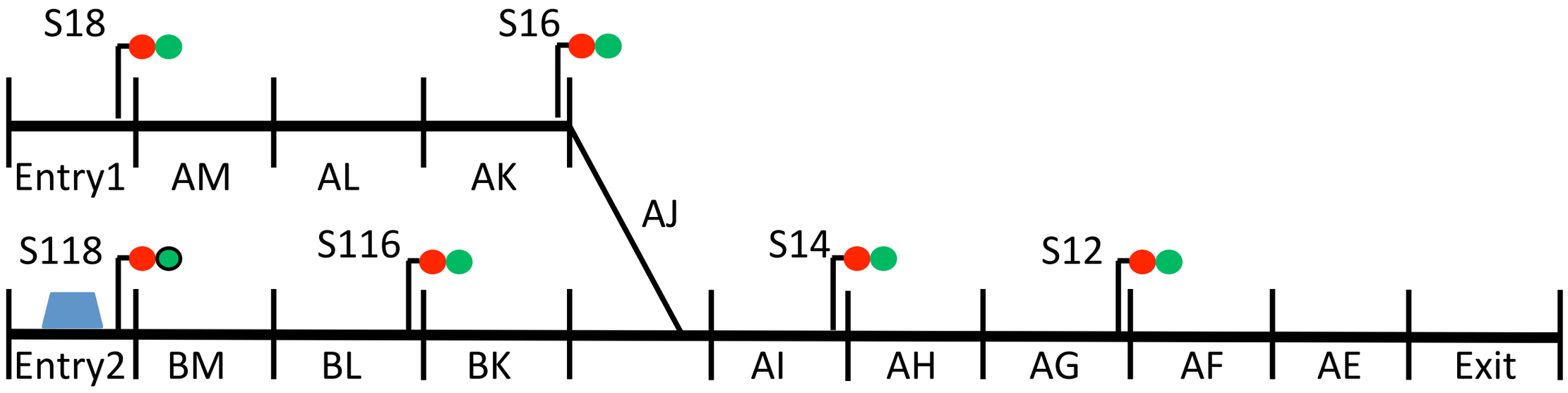
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



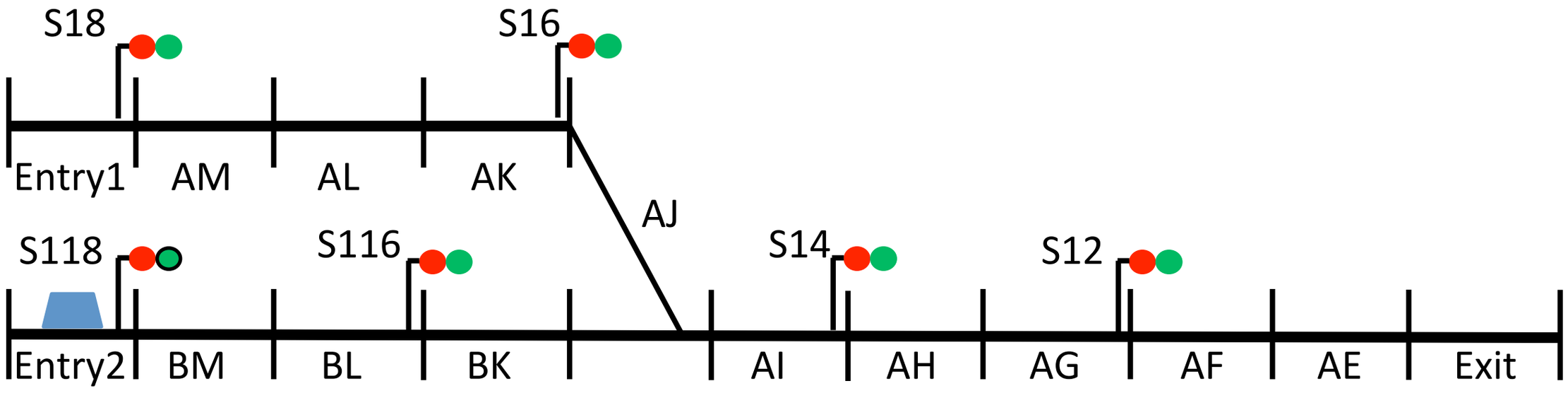
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



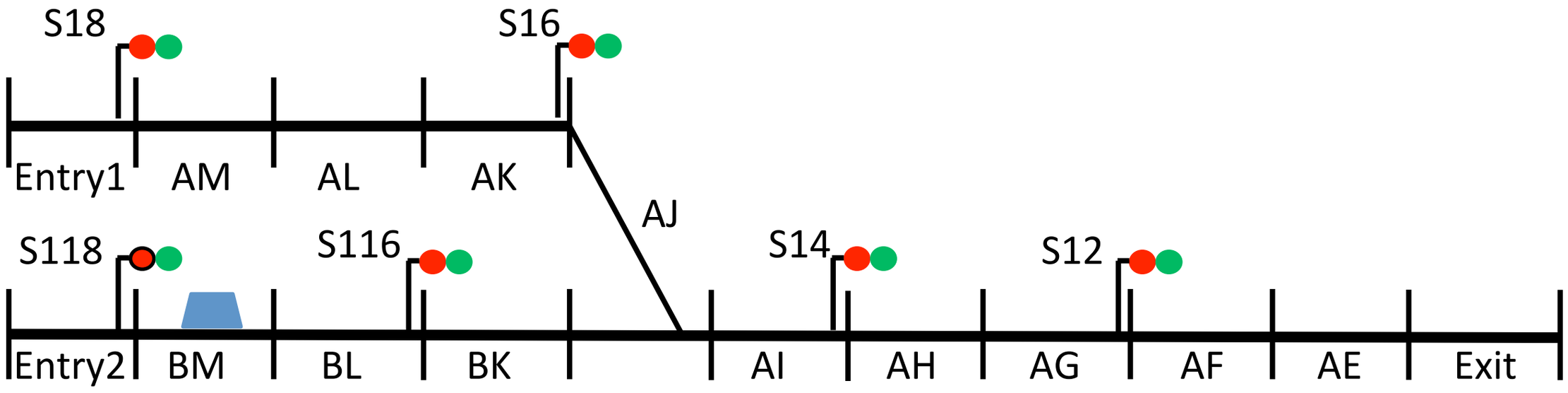
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



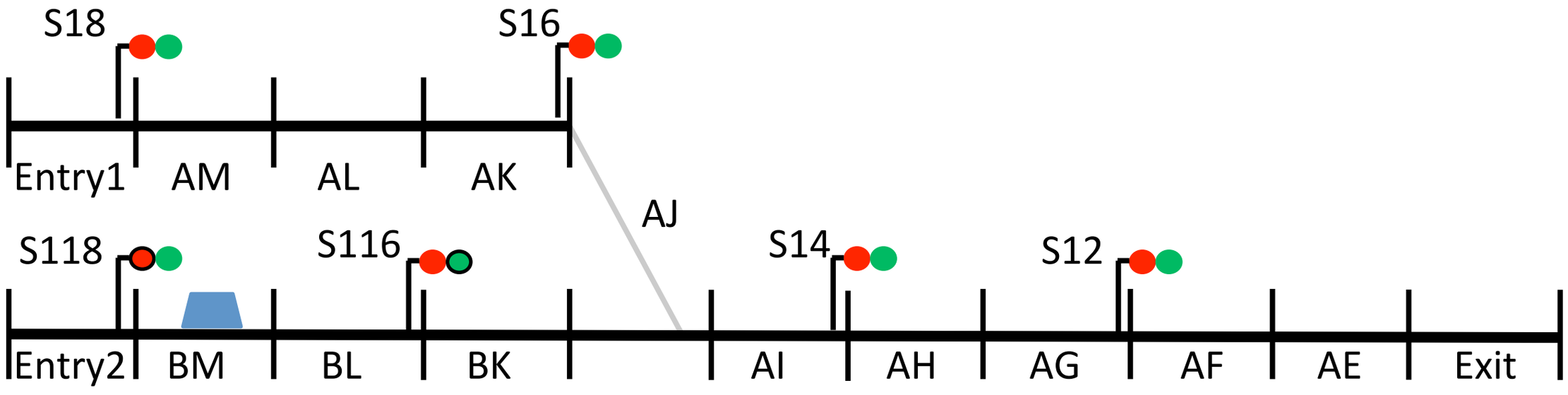
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



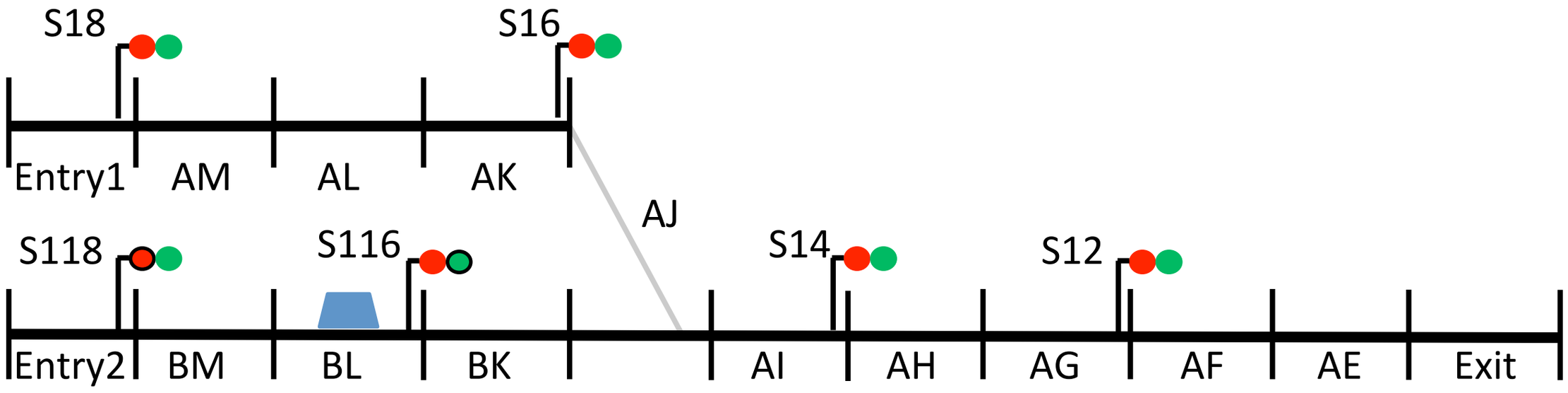
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



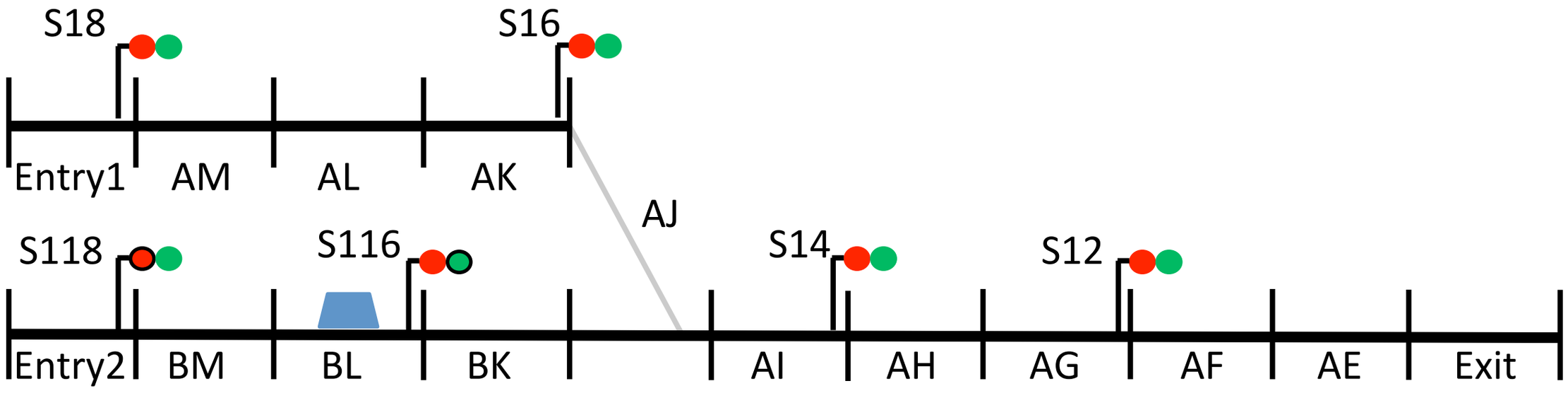
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



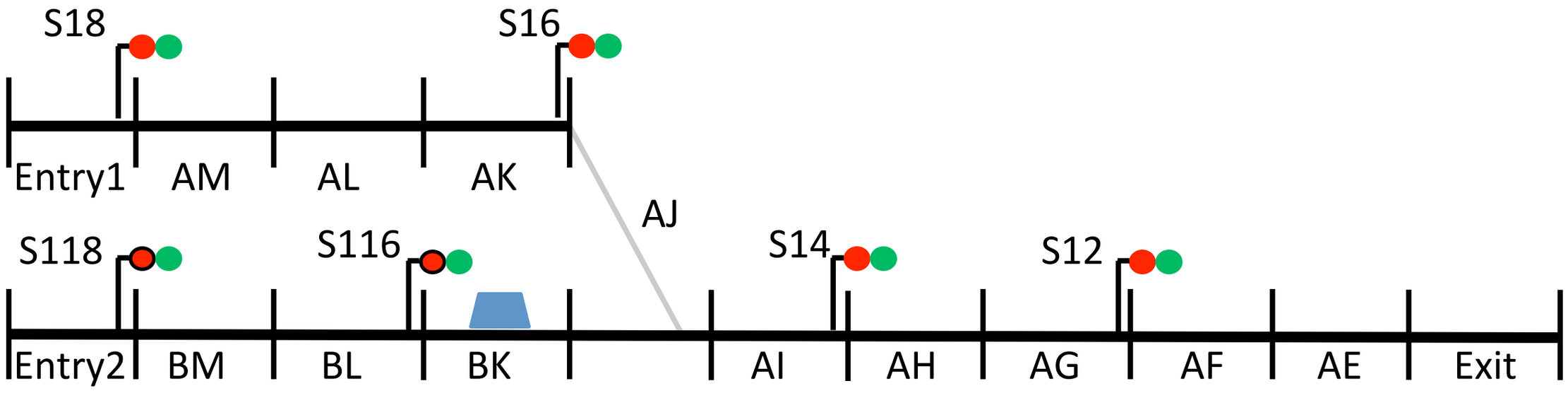
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



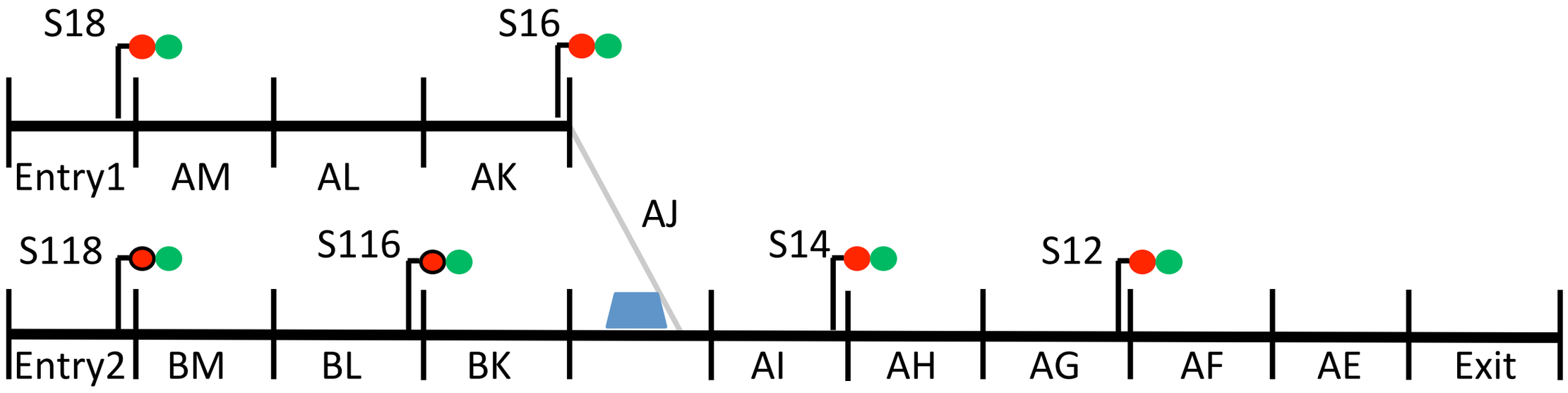
Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



Exploring the detailed trace

enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI