Incorporating time to an integrated formal method

Steve Schneider

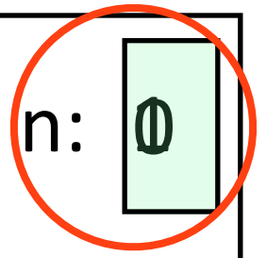
Structure of talk

- CSP||B and CSP||Event-B
- Application to railway modelling
- Challenges with introducing time

Integrating CSP and B/Event-B

Classical B machines and control

Machine M1 =
Variables n
Invariants $n \in \{0,1\}$
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

* preconditions do not determine control flow

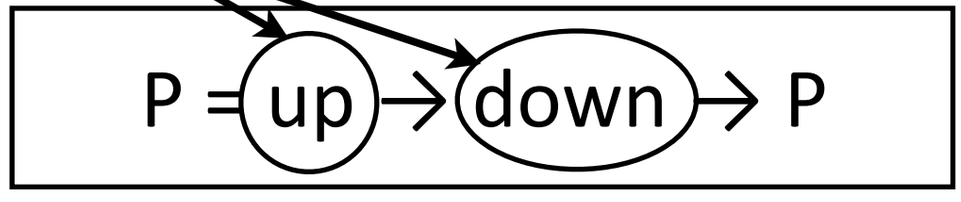
CSP||B:
CSP controllers for Classical B machines

Machine M1 =
Variables n
Invariants $n \in \{0,1\}$
Initialisation n:=0
Operations
up =
pre n=0 **then** n:=n+1 **end**
down =
pre n = 1 **then** n:=n-1 **end**

CSP process to describe the control flow for operations

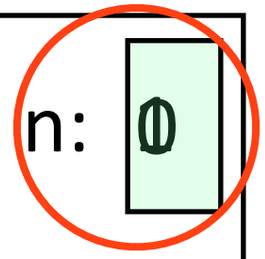
(Some) CSP events match (some) B operations

Previous results: Consistency conditions to ensure that the CSP does not drive the B to diverge



Event-B machines and control

Machine M1 =
Variables n
Invariants $n \in \{0,1\}$
Initialisation n:=0
Event **up** =
when n=0 **then** n:=n+1 **end**
Event **down** =
when n = 1 **then** n:=n-1 **end**



* Events are enabled when their guard is true, and blocked when their guard is false
* **up** can occur as the first event
* **down** can occur as the second event
* a second occurrence of **down** will then be blocked

* the control flow is determined by the guards

CSP

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

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}

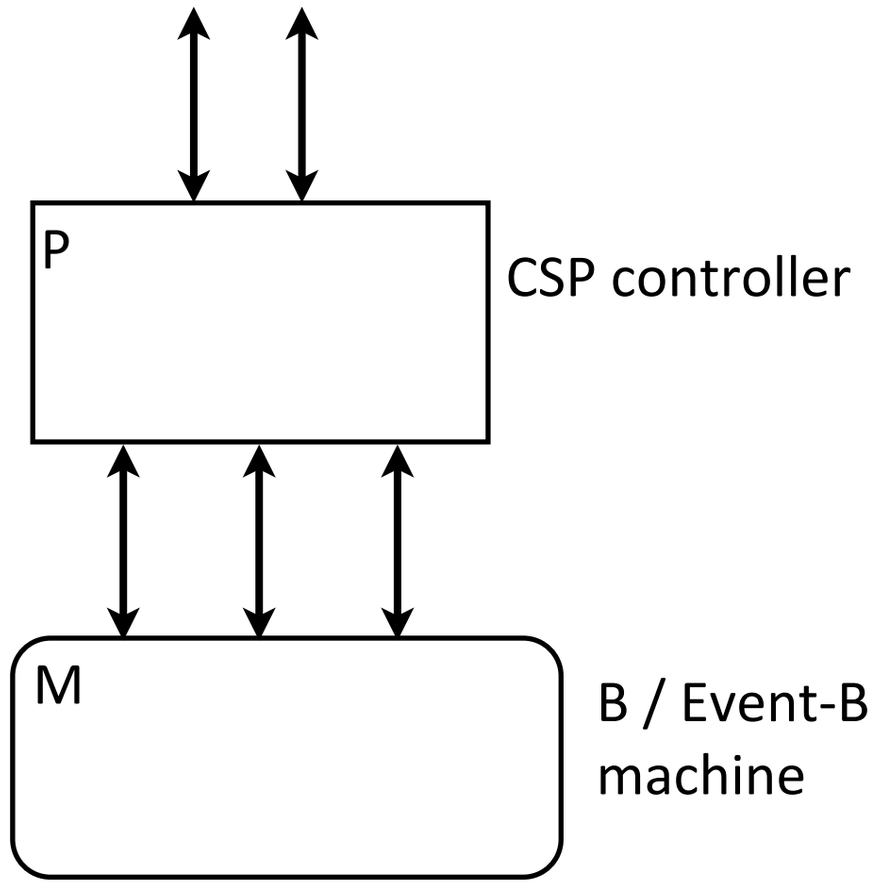
Combining CSP and B / Event-B: events and state

- Pure Event-B must use control variables for flow of control
- Pure B machines offer all their operations

- Combining with CSP separates concerns:
  - State (and some control) described in Event-B
  - Concurrency, communication, and control encapsulated naturally in CSP

Combining CSP and B / Event-B

- Some events in the machine correspond to events in the controller
- The interface is the alphabet of M
- Blocking can occur if P and M cannot agree to perform an event.
- Divergence can occur if an operation of M is called outside its precondition (not in Event B)



Semantic approach

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

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

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

Applying CSP||B

- Once M has a CSP semantics then it can be treated as a CSP process. Thus a *controlled component* **P||M** has a CSP semantics.
- 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
  - model-checking with ProB (invariant checking, LTL, CTL ...)

Application to Railway modelling

Steve Schneider
Helen Treharne
Matthew Trumble

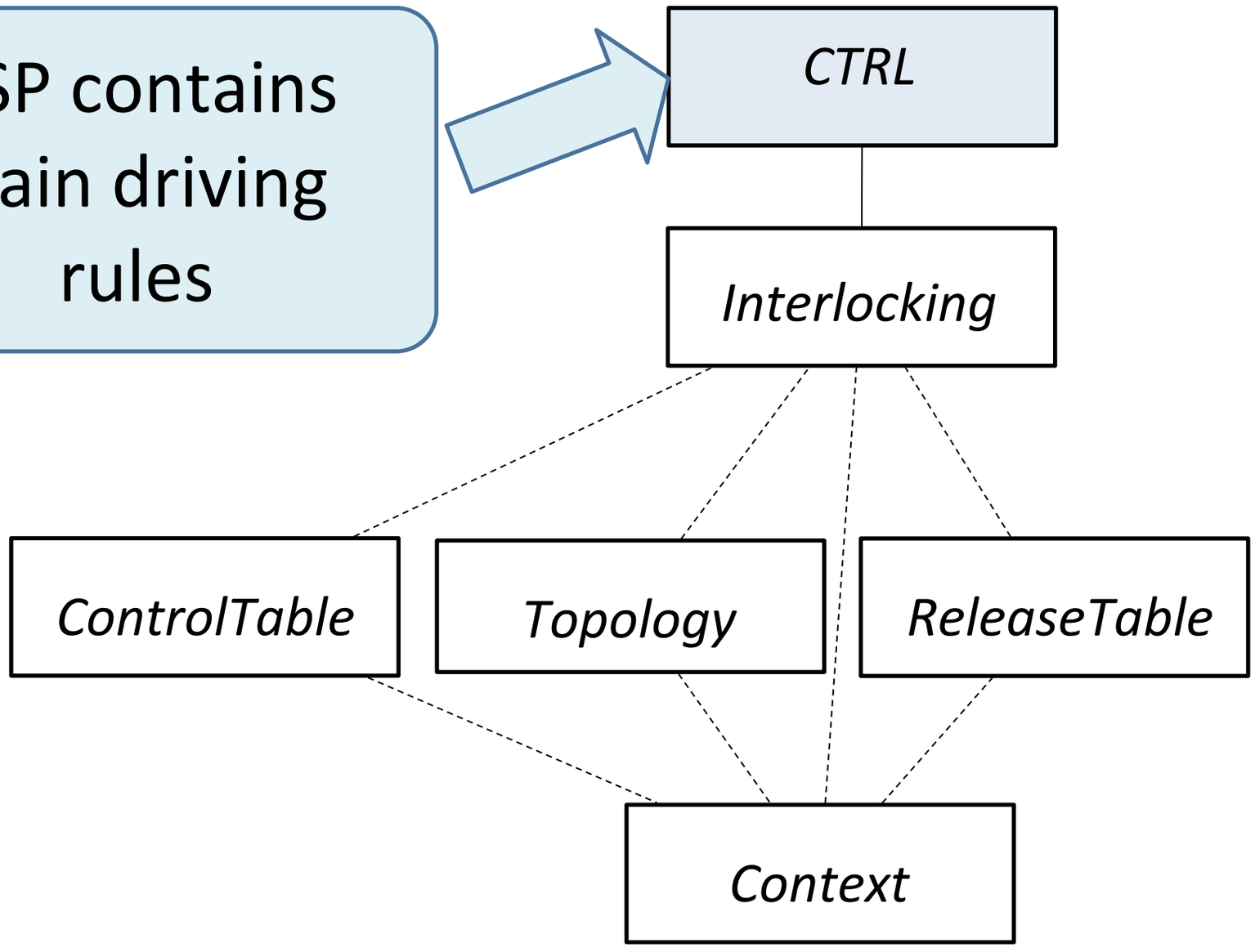
UNIVERSITY OF SURREY

Markus Roggenbach
Faron Moller
Nga Hoang Nguyen
Phil James

Swansea University
Prifysgol Abertawe

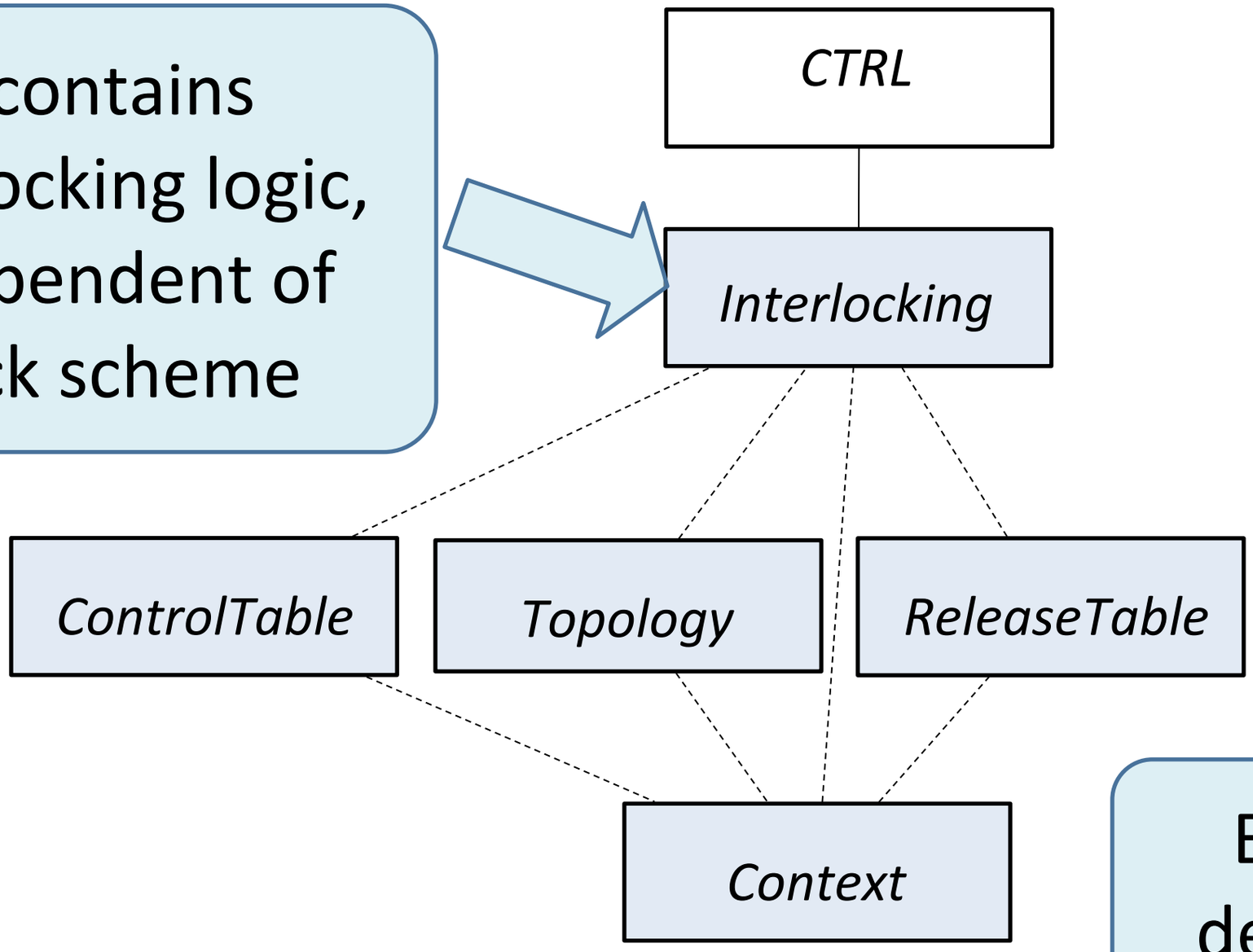
CSP||B Architecture

CSP contains train driving rules



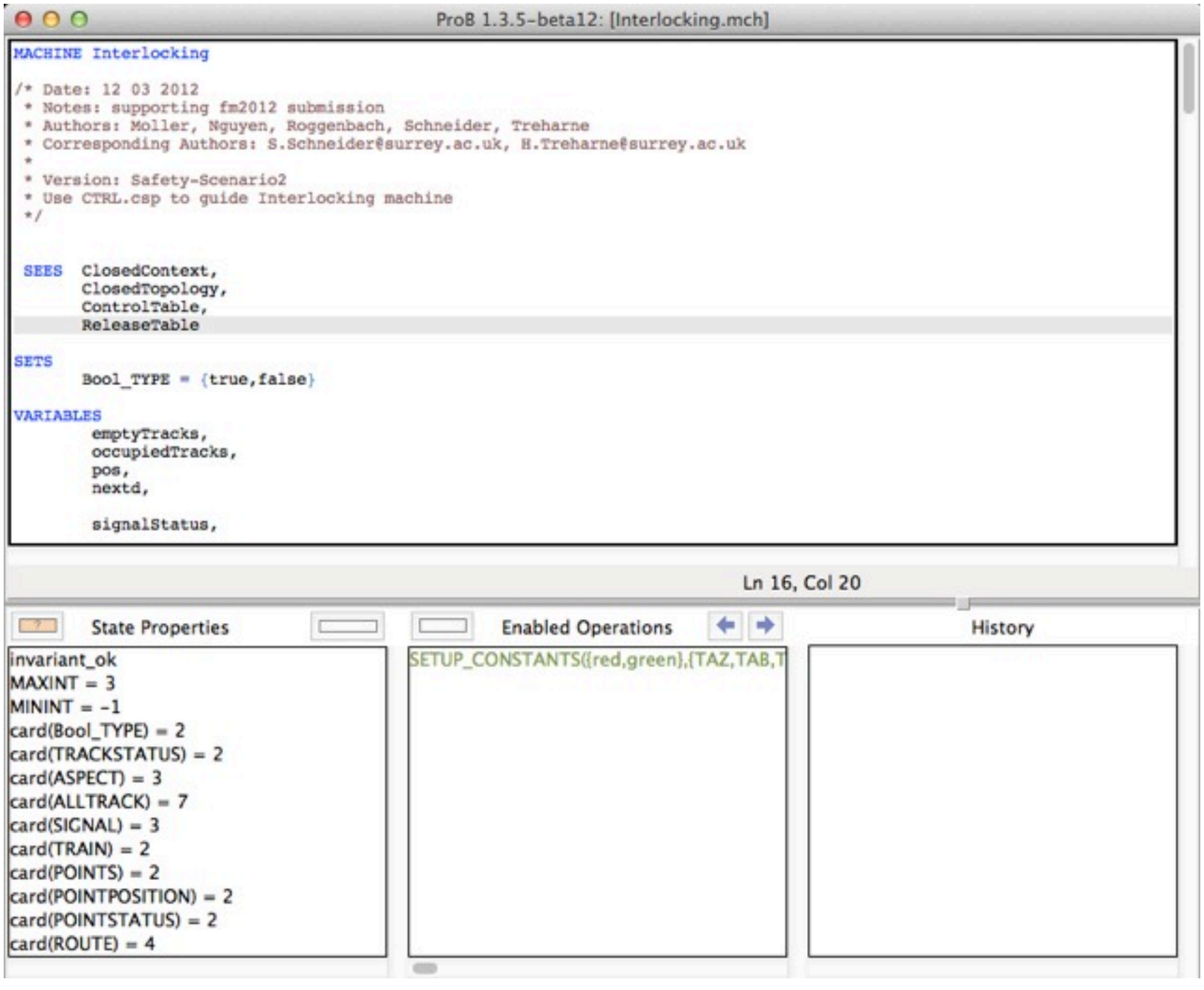
CSP||B Architecture

B contains interlocking logic, independent of track scheme



B machines defining track scheme

The ProB model checker



Example of CSP

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

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

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

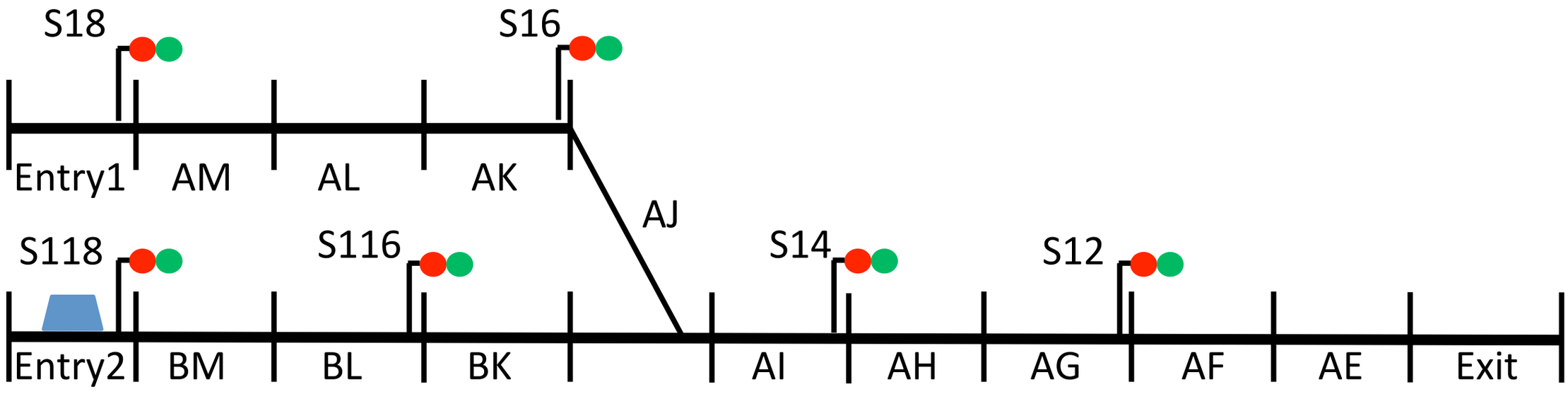
$$
ALL\_TRAINS = |||_{t \in TRAIN}\ TRAIN\_OFF(t)
$$

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;
```

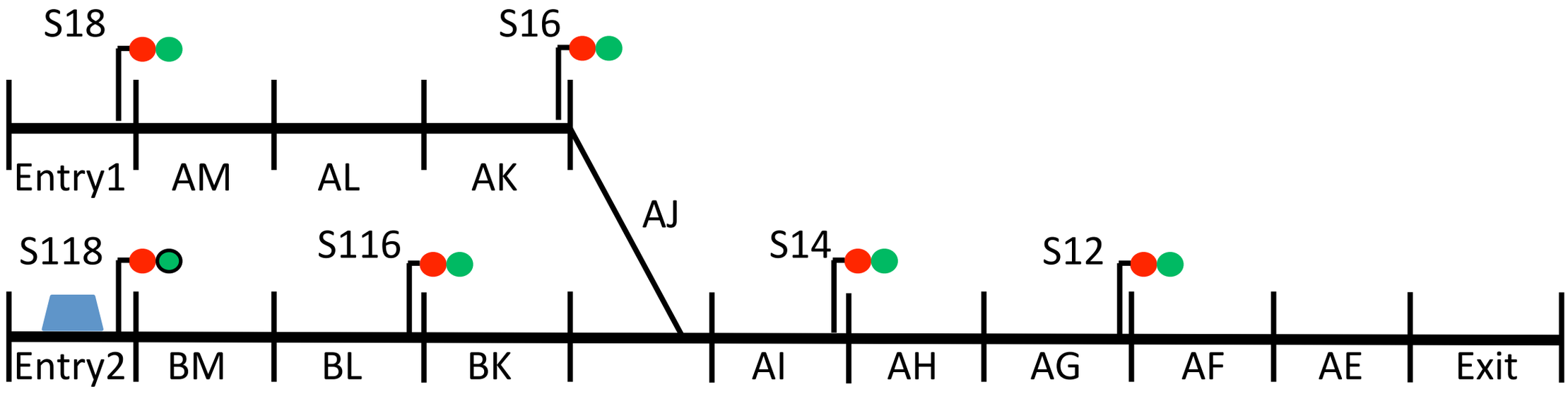
Exploring the detailed trace

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



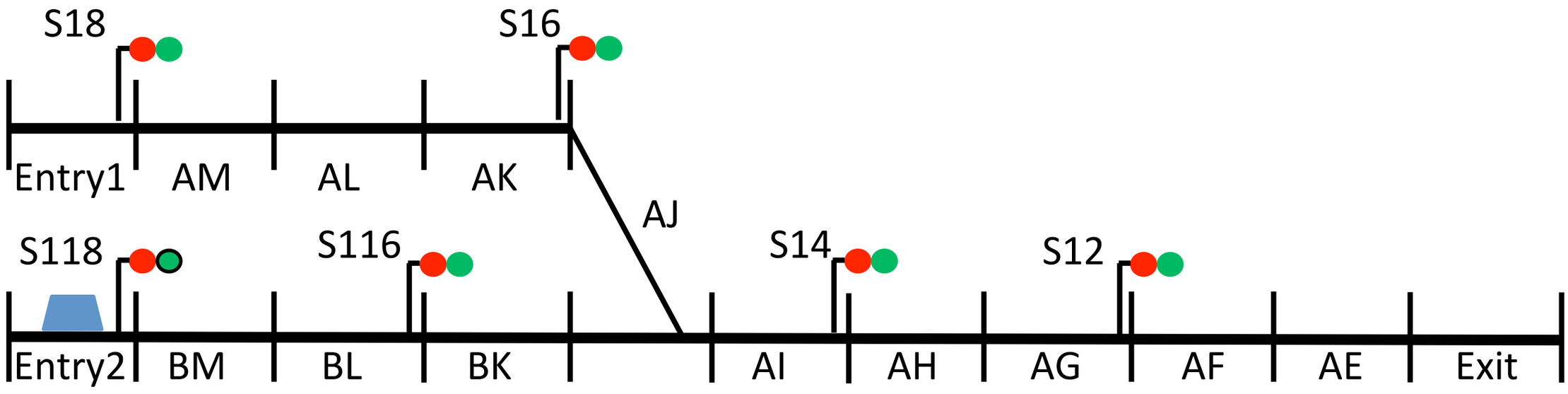
Exploring the detailed trace

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



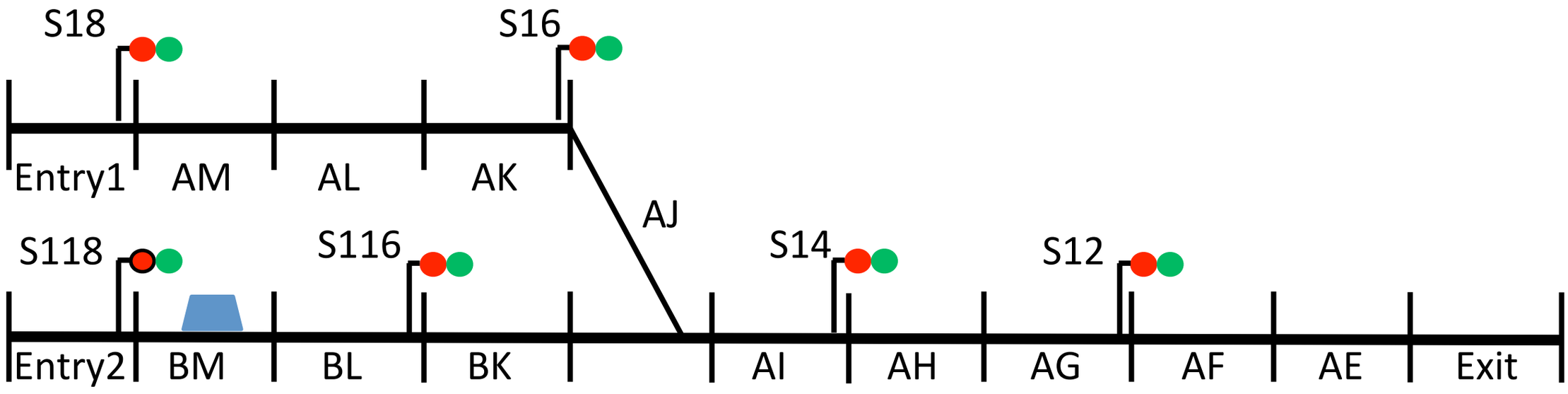
Exploring the detailed trace

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



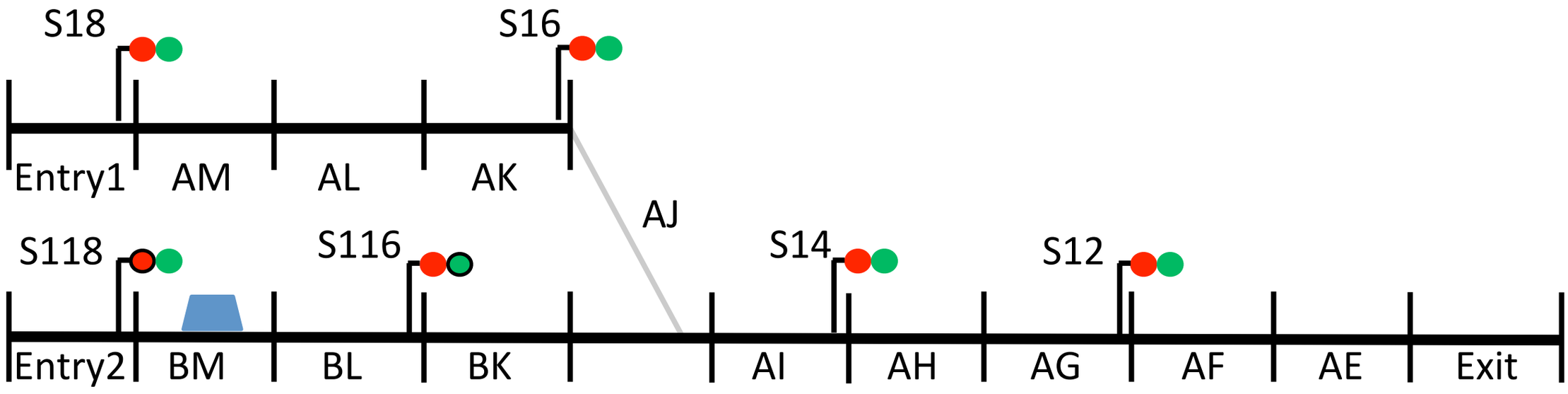
Exploring the detailed trace

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



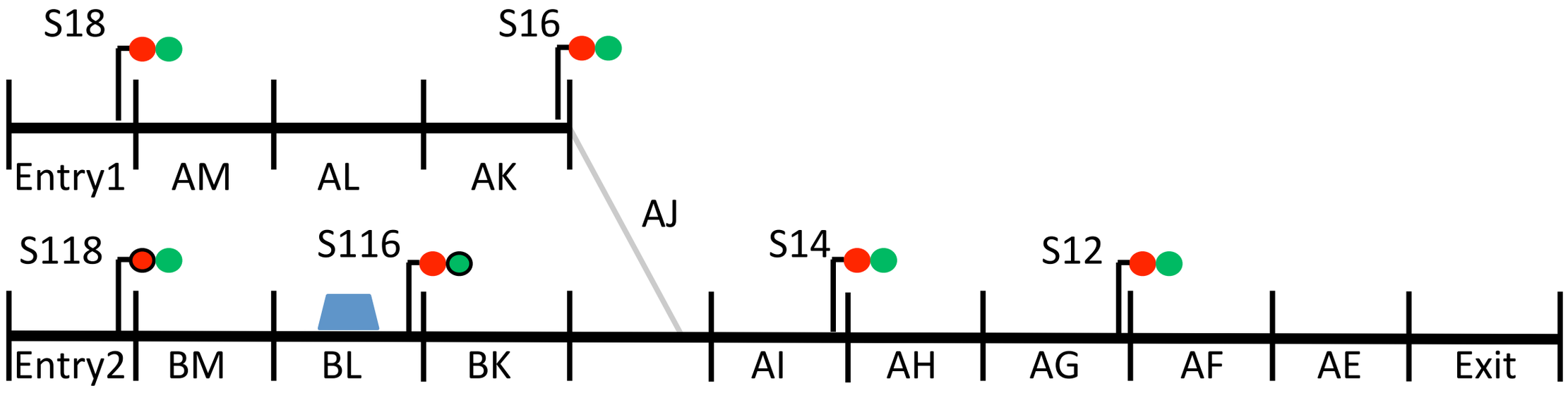
Exploring the detailed trace

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



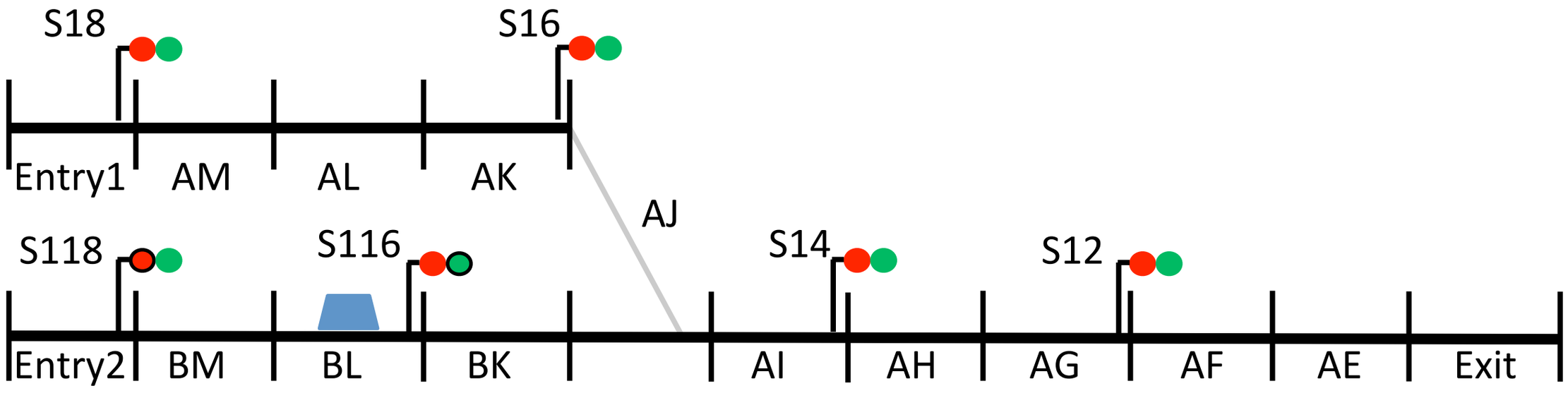
Exploring the detailed trace

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



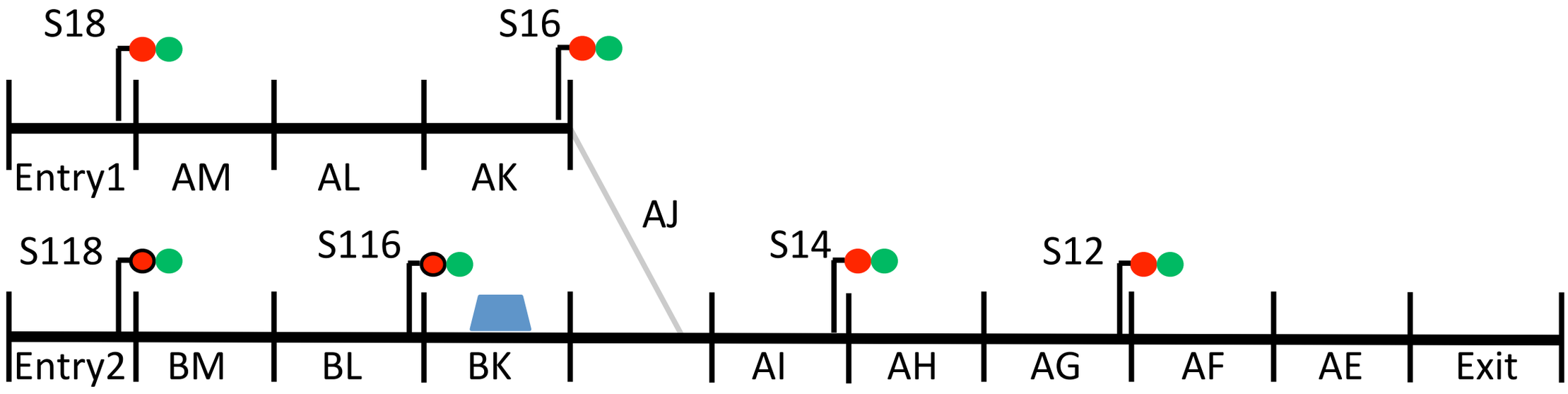
Exploring the detailed trace

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



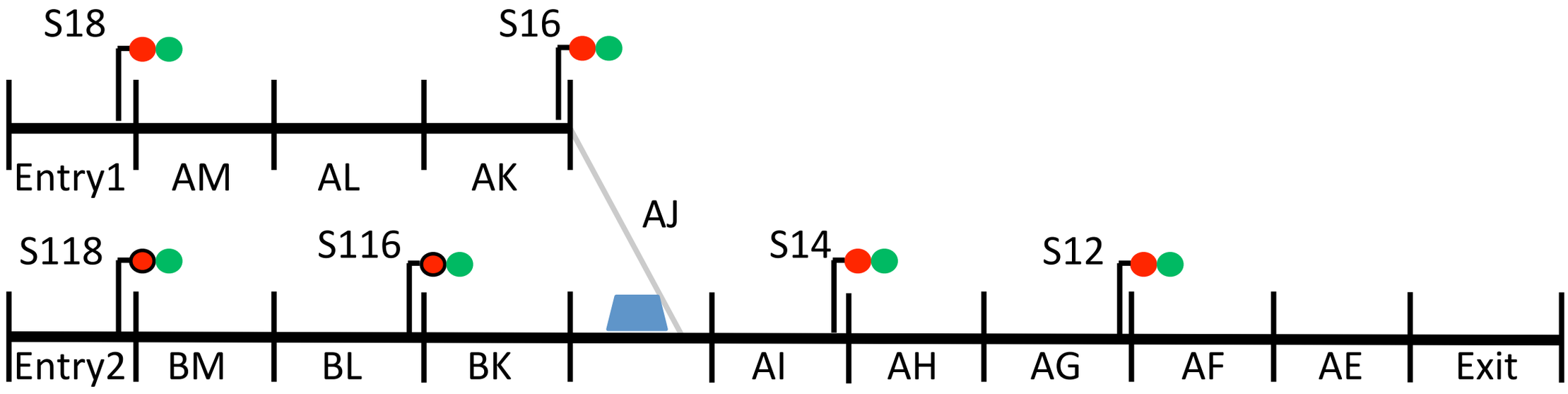
Exploring the detailed trace

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



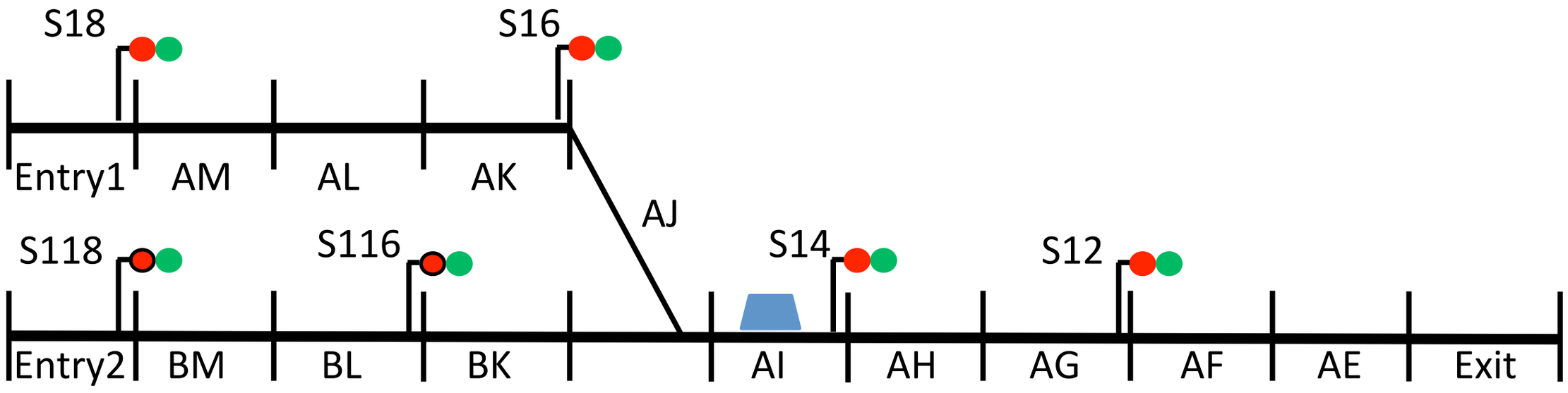
Exploring the detailed trace

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

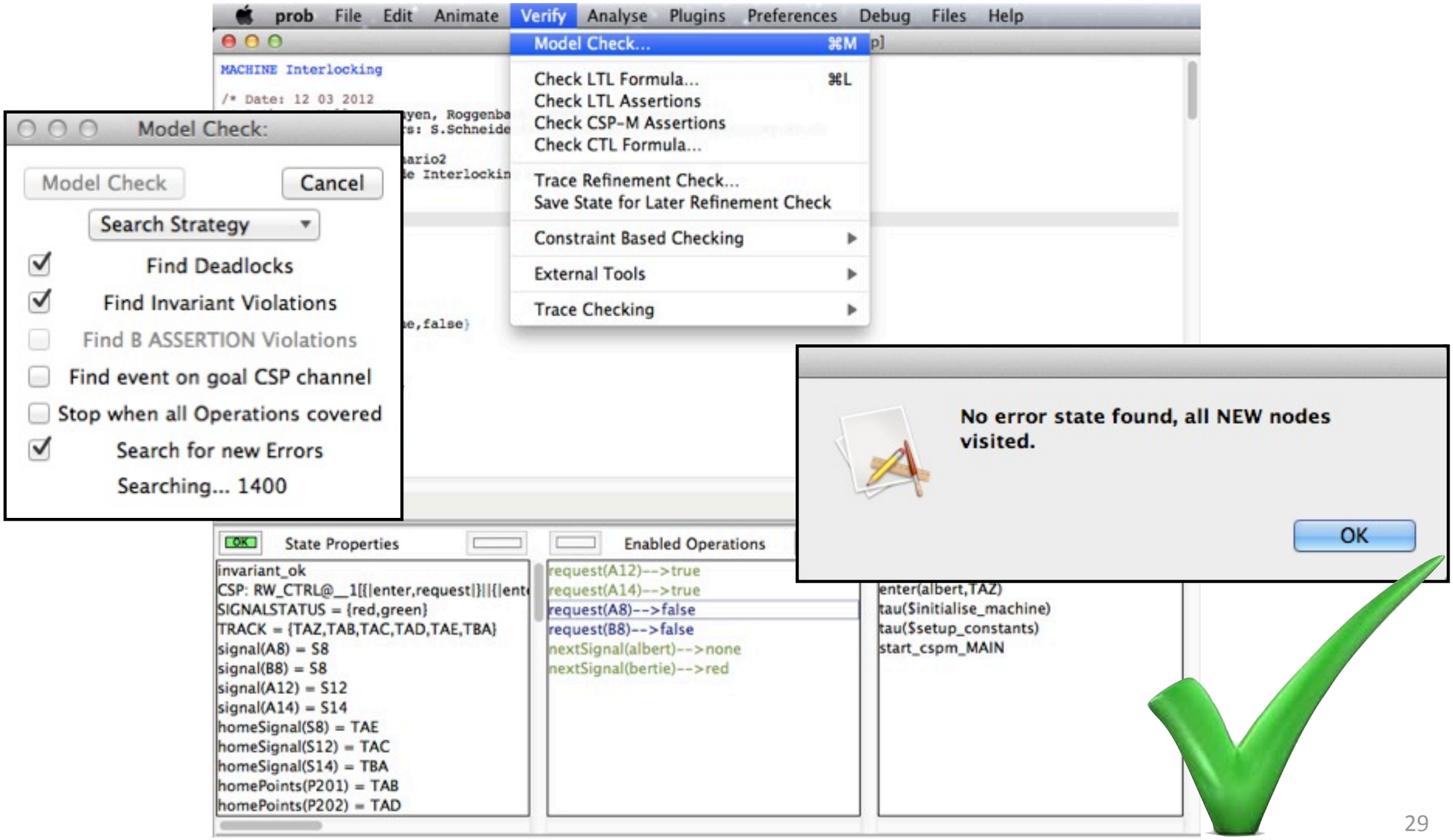


Exploring the detailed trace

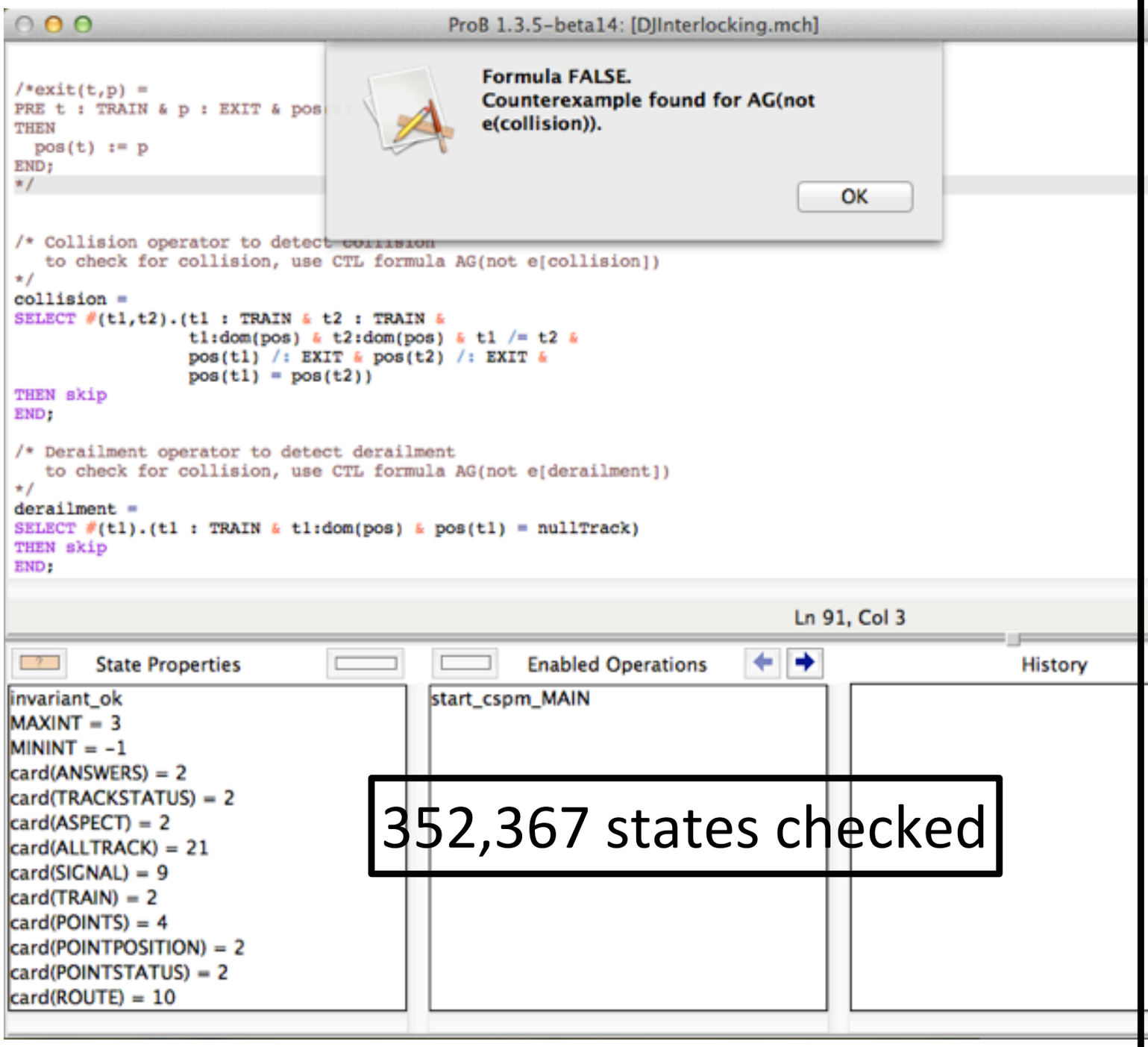
enter(bertie,Entry2)
request(R118A)-->yes
nextSignal(bertie)-->green
move(bertie)-->Entry2,BM
request(R116A)-->yes
move(bertie)-->BM,BL
nextSignal(bertie)-->green
move(bertie)-->BL,BK
move(bertie)-->BK,AJ
move(bertie)-->AJ,AI



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

Introducing time

railway designers are also interested in network capacity

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:

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

- (recursions time-guarded)

Timed CSP

- Operational semantics:
  - Event transitions: events occur instantaneously

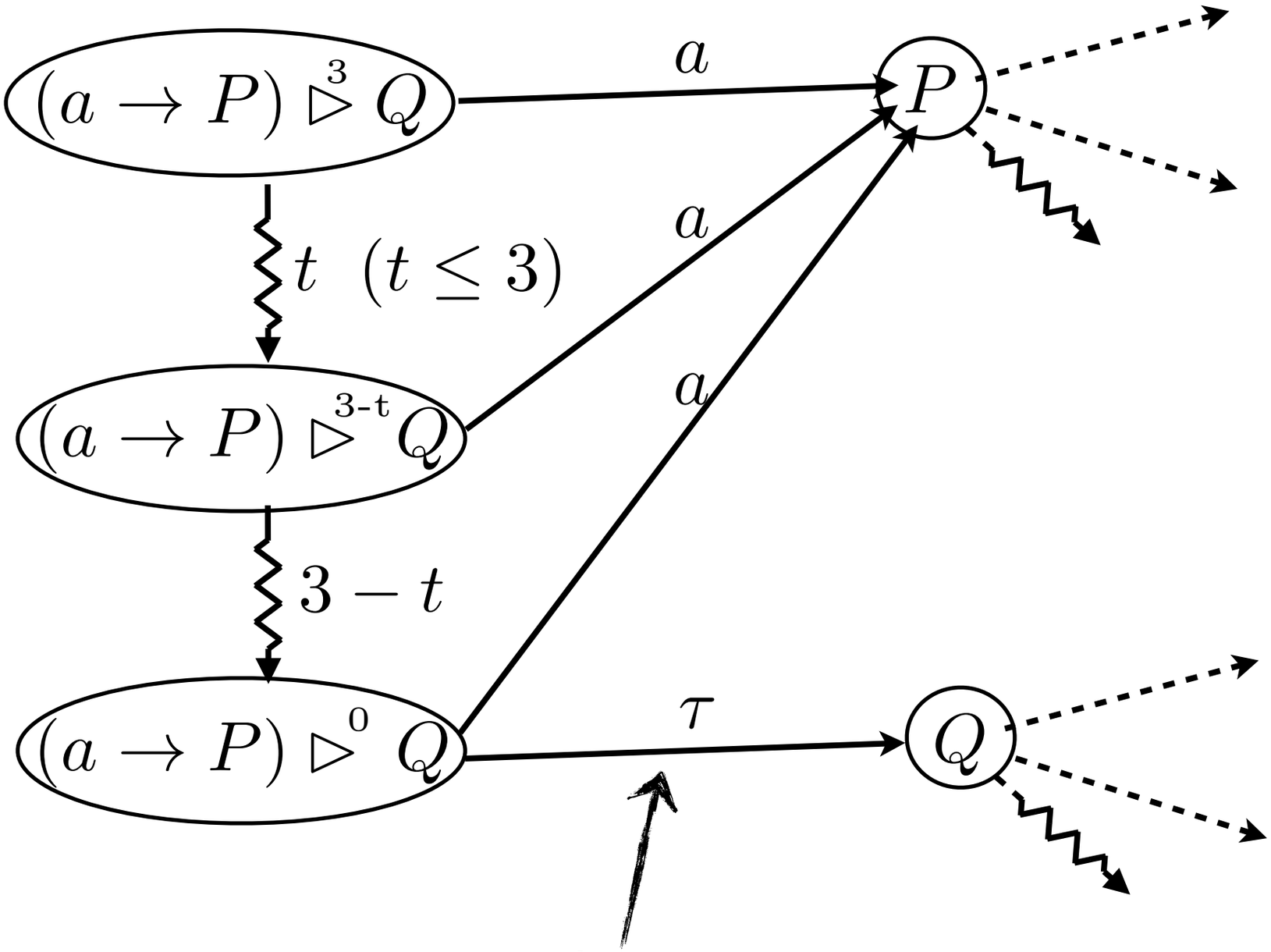
$$(a \to P) \quad \xrightarrow{a} \quad P$$

  - Delay transitions: time passes

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

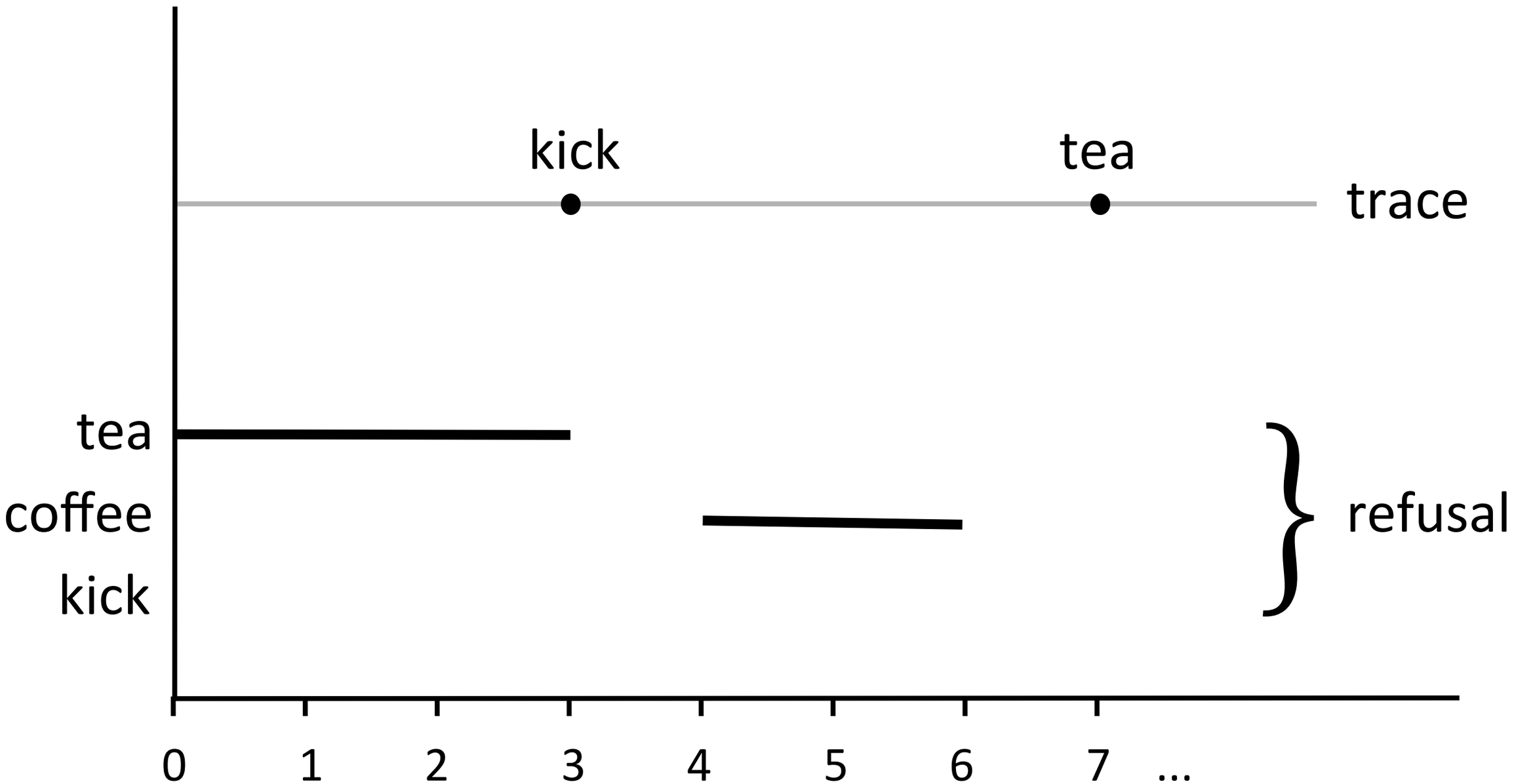
$$(P \overset{7}{\triangleright} Q) \quad \overset{2}{\rightsquigarrow} \quad (P' \overset{5}{\triangleright} Q)$$

Example fragment of a timed transition system



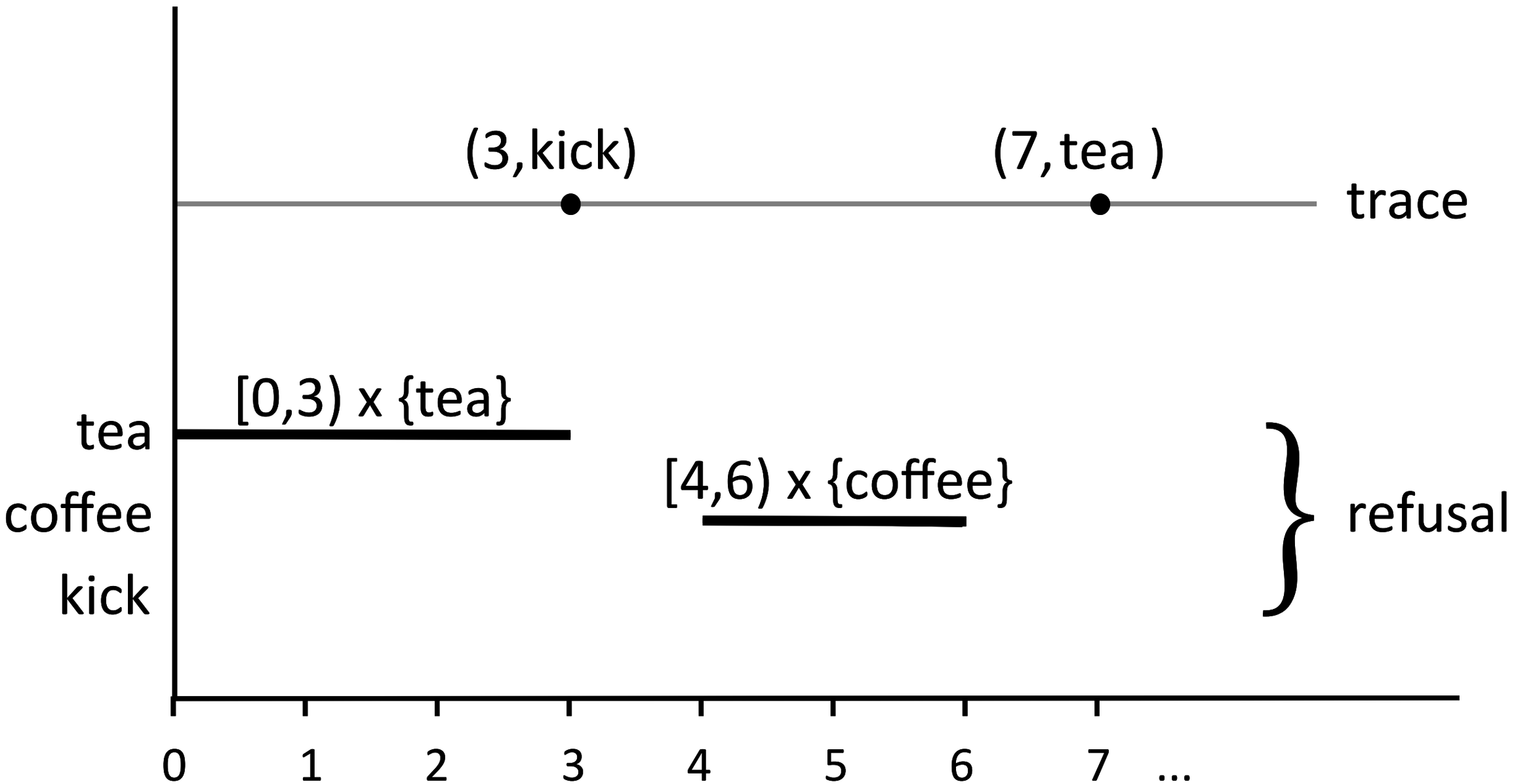
Observations for denotational semantics: Timed Failures

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



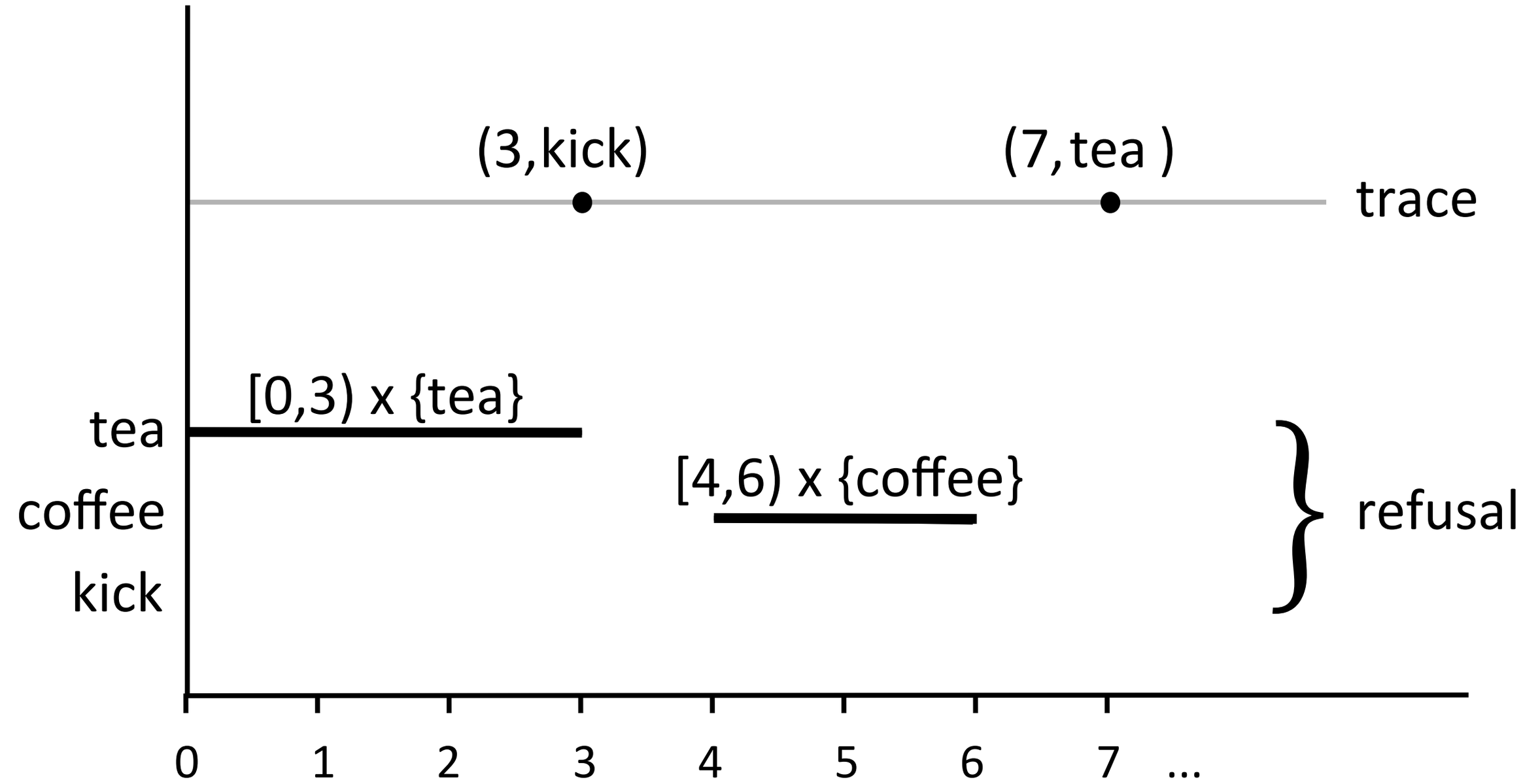
Observations for denotational semantics: Timed Failures

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



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

- NB: Refusal during the trace (as well as at the end). In fact this does give a problem.
- The presence of time allows **refusal testing**. e.g. observations which can tell the difference between these two processes* (which have the same untimed CSP semantics):

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

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

* example originally due (in some form) to RvG

1. Timed refusal testing

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

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

- timed refusal trace:

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

- possible for P1, not for P2

Timed CSP semantics?
Now need to distinguish these two:

Machine VM1 =
Variables n
Invariants $n \in \{0,1,2\}$
Initialisation $n:=0 \sqcap n:=1$
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 $n \in \{0,1,2\}$
Initialisation $n:=0 \sqcap n:=1$
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 (new idea last week...)

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

1. $\exists\ P_0\ ,\ P_1\ ,\ \ldots$

$$\overline{wp}(init, P_0)$$
$$P_0 \Rightarrow \overline{wp}(a_1, P_1)$$
$$\vdots$$
$$P_i \Rightarrow \overline{wp}(a_{i+1}, P_{i+1})$$

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

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

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

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

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

$P_0:$	$n = 1$
$P_1:$	$n = 0$
$P_2:$	$true$

$\neg g_{tea}:\quad n \neq 0$

$\bigwedge_{a \in X_1} \neg g_a:\quad true$

$\bigwedge_{a \in X_2} \neg g_a:\quad true$

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

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

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

2&3. 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 $n \in \{0,1,2\}$
Initialisation $n:=0 \sqcap n:=1$
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**

- VM1 can 'accept' a kick and instantly switch state.
- It can accept as many kicks as can be provided by its environment.
- Its semantics must allow it to go arbitrarily fast:

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

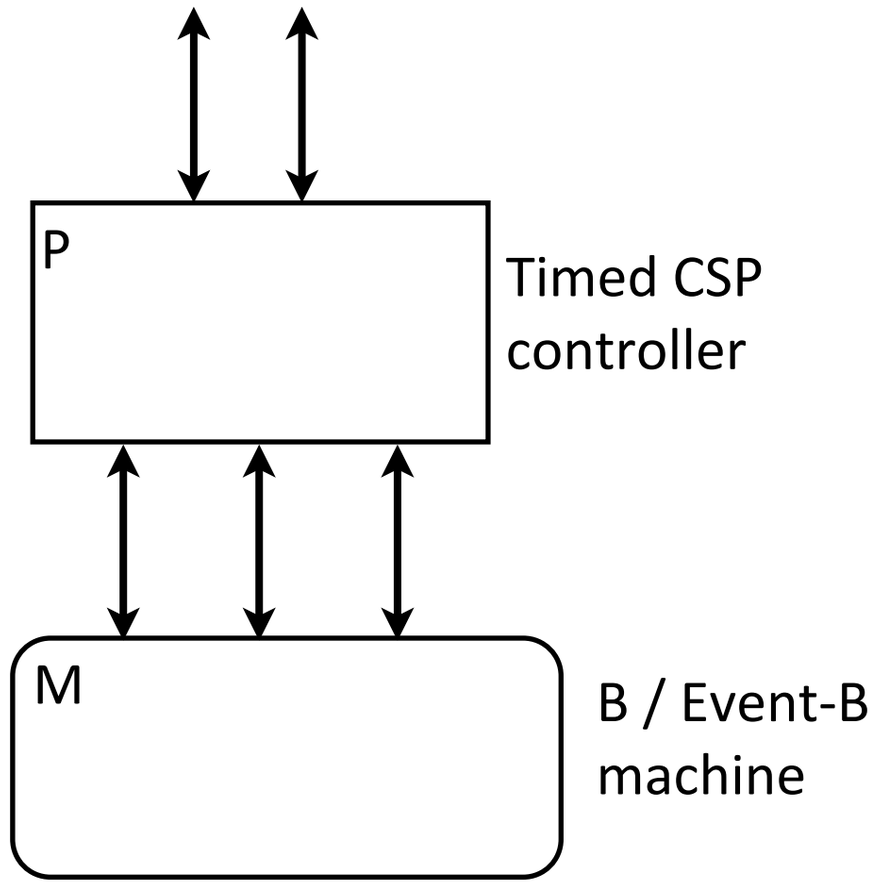
- How does $VM1 \backslash \{kick\}$ behave ?

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.

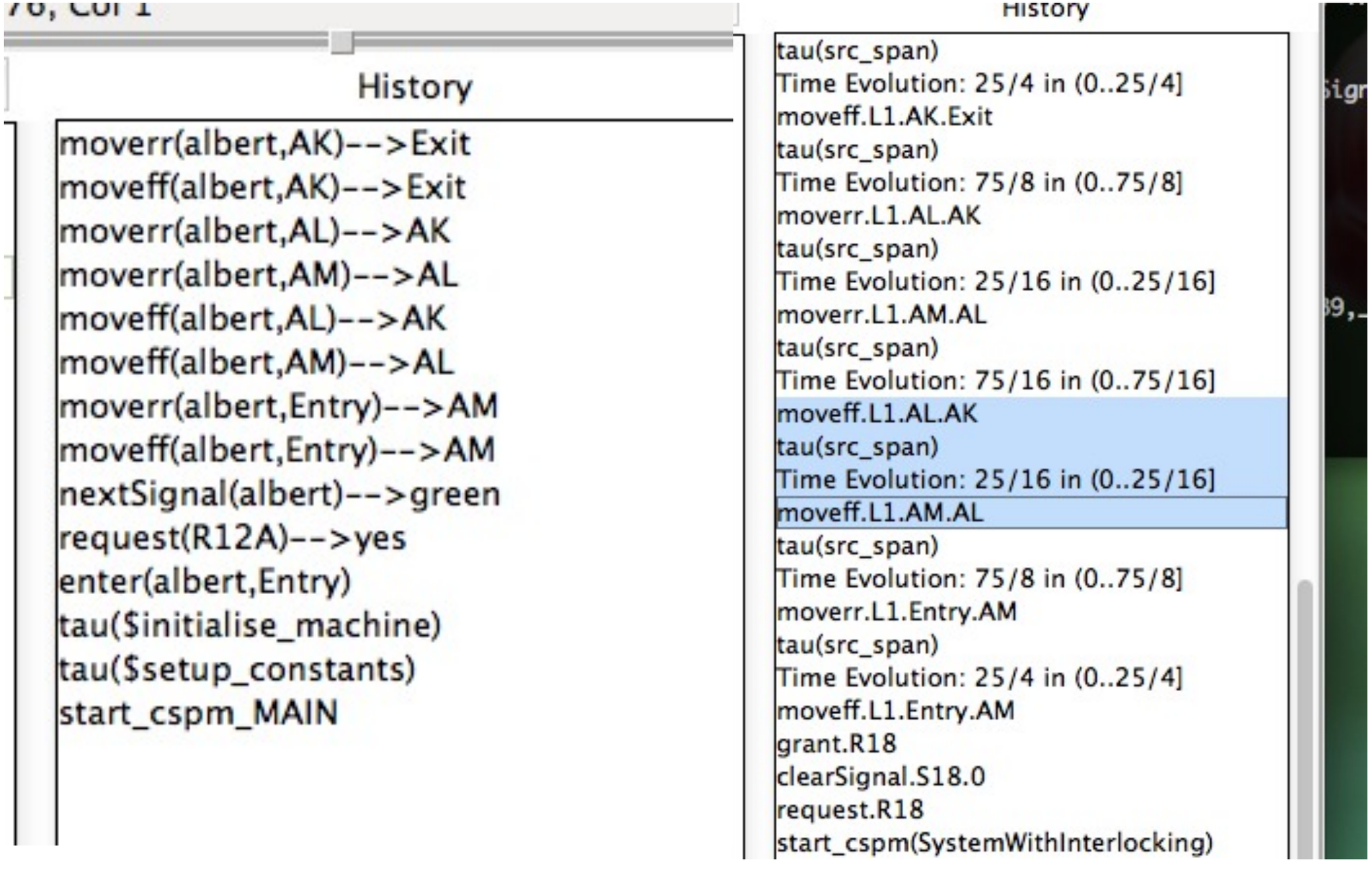


Timed CSP Train Description (extract)

```
-- TrainBehaves(id,l,ff,rr):
-- l: LineID
-- ff: front
-- df: distance to travel on the front track
-- rr: rear
-- dr: distance to travel on the rear track

TrainBehaves(id,l,ff,df,rr,dr) =
if {ff,rr}=={Exit} then Train(id)
else
(df<dr and df>0 &
   (WAIT df/speed(l,ff));TrainBehaves(id,l,ff,0,rr,dr-df))
[]
(df<dr and df==0 &
   [] t:next(ff) @ moveff.l.ff.t ->
      TrainBehaves(id,l,t,trackLength(t),rr,dr))
[]
(dr<=df and dr>0 &
   (WAIT dr/speed(l,ff));TrainBehaves(id,l,ff,df-dr,rr,0))
[]
(dr<=df and dr==0 &
   [] t:next(rr) @ moverr.l.rr.t ->
      TrainBehaves(id,l,ff,df,t,trackLength(t)))
```

Trace example in ProB
(Timed CSP operational rules added)



(dr<=df and dr>0 & (WAIT dr/speed(l,ff));TrainBehaves(id,l,ff,df-dr,rr,0)) in Timed CSP

Conclusion
This is all very new: where does it leave us?

- Timed failures semantic model required (!!)
- ...and proof of correspondence to Operational Semantics
- Rules for reasoning about Timed CSP||B/Event B and establishing properties of various kinds
- Modelling railway designs and reasoning about safety and capacity of track designs