Reasoned Modelling Critics: Turning Failed Proofs into Modelling Guidance

Andrew Ireland
School of Mathematical & Computer Sciences
Heriot-Watt University

Gudmund Grov
School of Informatics
University of Edinburgh

Michael Butler
School of Electronics & Computer Science
University of Southampton

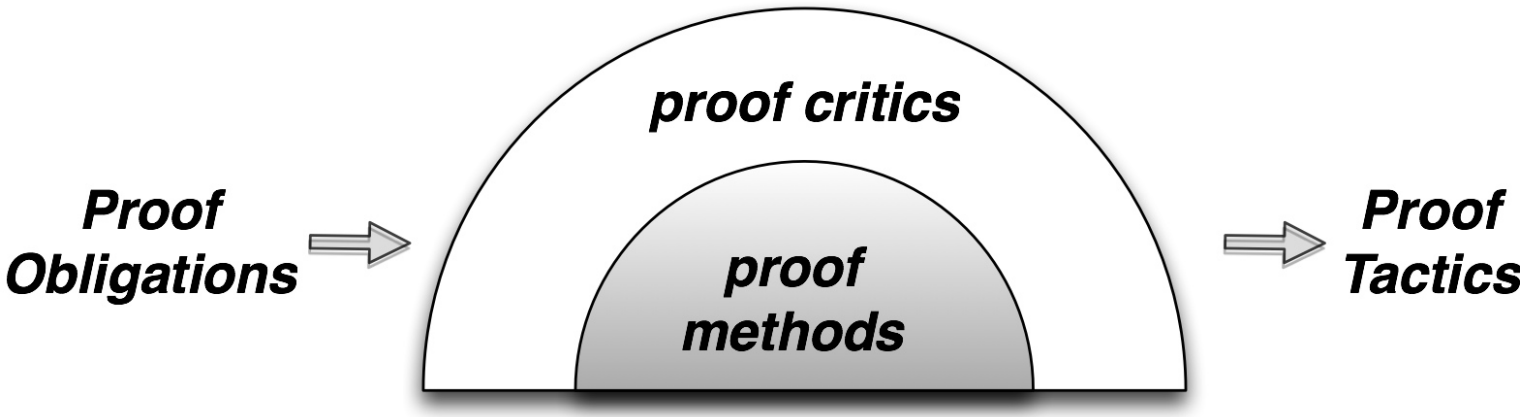
Motivations & Sponsors

- While the rigour of building formal models brings significant benefits, formal reasoning remains a major barrier to the wider acceptance of formalism within design.
- We aim to abstract away from the complexities of low-level proof obligations, providing high-level modelling guidance – we call this **reasoned modelling**.
- Accessibility and productivity – allow smart designers to make better use of their time.
- Generic vision, using Event-B as our starting point.
- EPSRC Funding: "A Cognitively Based Model of Theory Formulation and Reformulation"
  - University of Edinburgh (EP/F035594)
  - Heriot-Watt (EP/F037058):
    - Maria Teresa Llano (Research Student)
    - `www.macs.hw.ac.uk/sear`
  - Imperial College (EP/F036647)

Overview

- Reasoning patterns via proof plans
- Proof automation
- Proof-failure analysis: proof patching & modelling guidance
- Combining proof & modelling heuristics
- Ongoing & future work

Proof Planning Architecture



- A technique that uses high-level proof outlines, *i.e.* proof plans, to automate the search for proofs.
- Automatic proof failure analysis & proof patching.
- Promotes reuse and flexibility of proof strategies.

An Inductive Conjecture

- **Conjecture:**

$$\forall t : list(\tau).\ \ rotate(len(t), t) = t$$

- **Rewrite rules (definitions & lemmas):**

$$\begin{aligned}
len(nil) &\Rightarrow 0 \\
len(X :: Y) &\Rightarrow s(len(Y)) \\
nil <> Z &\Rightarrow Z \\
(X :: Y) <> Z &\Rightarrow X :: (Y <> Z) \\
rotate(0, Z) &= Z \\
rotate(s(X), nil) &= nil \\
rotate(s(X), Y :: Z) &= rotate(X, (Z <> (Y :: nil))) \\
(X <> Y) <> Z &= X <> (Y <> Z) \\
X <> (Y :: Z) &= (X <> Y :: nil) <> Z
\end{aligned}$$

Step-case Proof

Generalized conjecture:

$$\forall t, l : list(\tau).\ \ rotate(len(t), t <> l) = l <> t$$

Given:

$$\forall l' \in list(\tau).\ rotate(len(t), t <> l') = l' <> t$$

Proof:

$$\begin{aligned}
rotate(len(h :: t), h :: t <> l) &= l <> h :: t \\
rotate(s(len(t)), h :: t <> l) &= l <> h :: t \\
rotate(s(len(t)), h :: t <> l) &= l <> h :: t \\
rotate(len(t), (t <> l) <> h :: nil) &= l <> h :: t \\
rotate(len(t), t <> (l <> h :: nil)) &= (l <> h :: nil) <> t
\end{aligned}$$

Rippling: A Common Pattern of Reasoning

Given:

$$\forall l' \in list(\tau).\ rotate(len(t), t <> l') = l' <> t$$

Proof:

$$rotate(len(\ h :: t\ ^{\uparrow}\ ),\ h :: t\ ^{\uparrow}\ <> \lfloor l \rfloor) = \lfloor l \rfloor <>\ h :: t\ ^{\uparrow}$$

$$rotate(\ s(len(t))\ ^{\uparrow}\ ,\ h :: t\ ^{\uparrow}\ <> \lfloor l \rfloor) = \lfloor l \rfloor <>\ h :: t\ ^{\uparrow}$$

$$rotate(\ s(len(t))\ ^{\uparrow}\ ,\ h :: t <> \lfloor l \rfloor\ ^{\uparrow}\ ) = \lfloor l \rfloor <>\ h :: t\ ^{\uparrow}$$

$$rotate(len(t),\ (t <> \lfloor l \rfloor) <> h :: nil\ ^{\downarrow}\ ) = \lfloor l \rfloor <>\ h :: t\ ^{\uparrow}$$

$$rotate(len(t), t <> \left\lfloor\ (l <> h :: nil)\ ^{\downarrow}\ \right\rfloor) = \left\lfloor\ (l <> h :: nil)\ ^{\downarrow}\ \right\rfloor <> t$$

Rippling: Reasoning about Refinement

Givens:

$$am(t) = a \ \wedge from(t) = p$$

$$abal(p) = cbal(p) + sum((pending \lhd from)^{-1}[\{p\}])$$

Proof:

$$abal(p) = \boxed{cbal(p) - a}^{\uparrow} + sum((\boxed{pending \cup \{t\}}^{\uparrow} \lhd from)^{-1}[\{p\}])$$

$$abal(p) = \boxed{cbal(p) - a}^{\uparrow} + sum((\boxed{(pending \lhd from \cup (\{t\} \lhd from))}^{\uparrow})^{-1}[\{p\}])$$

$$abal(p) = \boxed{cbal(p) - a}^{\uparrow} + sum(\boxed{((pending \lhd from)^{-1} \cup (\{t\} \lhd from)^{-1})}^{\uparrow}[\{p\}])$$

$$abal(p) = \boxed{cbal(p) - a}^{\uparrow} + sum(\boxed{((pending \lhd from)^{-1}[\{p\}] \cup (\{t\} \lhd from)^{-1})[\{p\}]}^{\uparrow})$$

$$abal(p) = \boxed{cbal(p) - a}^{\uparrow} + \boxed{sum((pending \lhd from)^{-1}[\{p\}]) + sum((\{t\} \lhd from)^{-1}[\{p\}])}^{\uparrow}$$

$$\ldots$$

$$abal(p) = \boxed{cbal(p) - a}^{\uparrow} + \boxed{sum((pending \lhd from)^{-1}[\{p\}]) + a}^{\uparrow}$$

$$abal(p) = cbal(p) + \boxed{sum((pending \lhd from)^{-1}[\{p\}]) + a}^{\uparrow} - a^{\downarrow}$$

$$abal = cbal(p) + sum((pending \lhd from)^{-1}[\{p\}])$$

Proof Planning

$$rotate(\ s(\ len(t)\ )^{\uparrow}\ ,\ h :: t\ ^{\uparrow}\ <>\ \lfloor l \rfloor\ ) \quad = \quad \ldots$$

$$rotate(len(t),\ (\ t <>\ \lfloor l \rfloor\ )\ <>\ h :: nil\ ^{\downarrow}\ ) \quad = \quad \ldots$$

Method preconditions (sideways ripple)

1. *There exists a wave-occurrence*, e.g.

   $$rotate(\ s(\ len(t)\ )^{\uparrow}\ ,\ h :: t <>\ \lfloor l \rfloor\ ^{\uparrow}\ )$$

2. *and a matching wave-rule*, e.g.

   $$rotate(\ s(X)\ ^{\uparrow}\ ,\ Y :: Z\ ^{\uparrow}\ ) :\Rightarrow rotate(X,\ (Z <>\ (Y :: nil))\ ^{\downarrow}\ )$$

3. *and any condition attached to the rewrite is provable,*
4. *and progress being made towards a* $\forall$*-variable*, e.g.

   $$rotate(len(t),\ (t <>\ \lfloor l \rfloor\ <>\ (h :: nil))\ ^{\downarrow}\ ) = \ldots$$

Rippling: A Common Pattern of Reasoning

Given:

$$rotate(len(t), t) = t$$

Proof:

$$\underbrace{rotate(\ s(len(t))^{\uparrow}\ ,\ h :: t^{\uparrow}\ )}_{\textbf{blocked}} = h :: t^{\uparrow}$$

search space pruned ⋮ ⋮

$$rotate(\ s(len(t))^{\uparrow}\ ,\ h :: t^{\uparrow}\ ) = h :: t^{\uparrow}$$

$$rotate(len(t),\ (t) <> h :: nil^{\downarrow}\ ) = h :: t^{\uparrow}$$

Proof-failure Analysis & Patching

$$\underbrace{rotate(\ s(len(t))^{\uparrow}\ ,\ h :: t^{\uparrow}\ )}_{\textbf{blocked}} = h :: t^{\uparrow}$$

Critic preconditions (sink speculation):

- *Preconditions* (1), (2) *and* (3) *of the wave method hold, e.g.*
  1. *There exists a wave-occurrence,* e.g.

     $$rotate(\ s(len(t))^{\uparrow}\ ,\ h :: t^{\uparrow}\ )$$

  2. *and a matching wave-rule,* e.g.

     $$rotate(\ s(X)^{\uparrow}\ ,\ Y :: Z^{\uparrow}\ ) :\Rightarrow rotate(X,\ (Z <> (Y :: nil))^{\downarrow}\ )$$

  3. *and any condition attached to the rewrite is provable.*
- *Precondition* (4) *is false,* i.e.
  4. **no $\forall$-variable.**

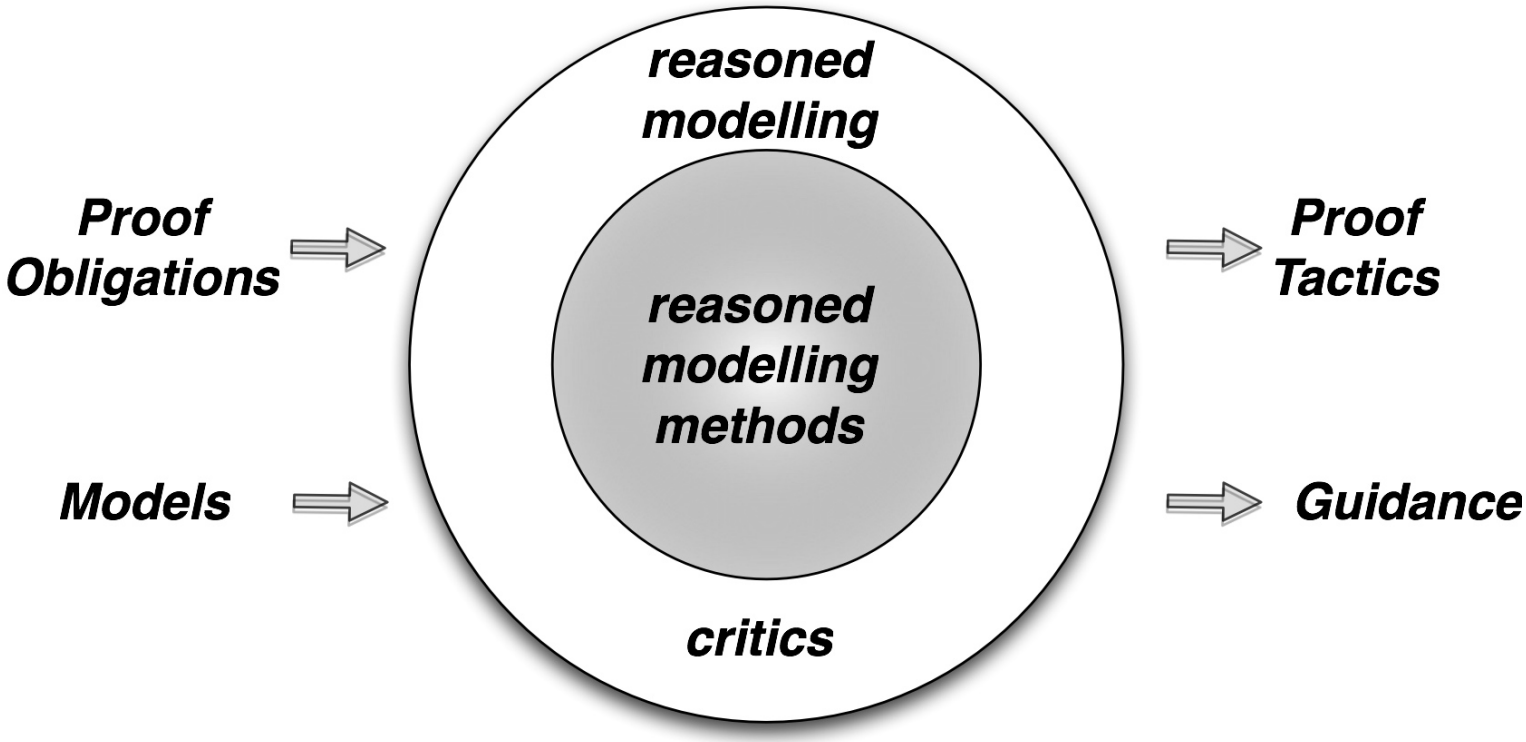
Proof patch: speculate conjecture generalization, *i.e.*

$$\forall t, l : list(\tau).\ rotate(len(t), F_1(t, l)) = G_1(t, l)$$

More Proof Patching

- **Conjecture generalization:** The generalization critic has also been applied to the problem of loop invariant discovery – could potentially be used to suggest Event-B gluing invariants.
- **Lemma discovery:** The constraints of rippling have been used to automate the discovery of missing lemmas – potentially useful in building up a theory within the context of an Event-B model.
- **Induction revision:** Heuristics for selecting induction rules may fail – by analysing proof-failures, induction revision patches such failures.
- **Case analysis:** Failure in applying conditional rewrite rules can be used to suggest case splits within a proof.
- **Fixing faulty conjecture:** Abduction has been used to suggest ways in which non-theorems can be "usefully" transformed into theorems – abductive reasoning will play an important role within reasoned modelling.

Proposed Reasoned Modelling Architecture



- Reasoned modelling aims to turn proof-failures into modelling suggestions, while maintaining the benefits of proof planning.
- Crucially, the critics layer will combine proof-failure analysis with modelling heuristics.

Cruise Control Example

MACHINE cruise_ctrl
VARIABLES brake, cc
INVARIANTS inv1: $cc = on \Rightarrow brake = off$
EVENTS

INITIALISATION $\widehat{=}$
BEGIN
act1 : brake := off
act2 : cc := off
END

EVENT enable_cc $\widehat{=}$
BEGIN
act1 : cc := on
END

EVENT pressbrake $\widehat{=}$
BEGIN
act1 : brake := on
END

...
END

Analysis of Failed Invariant Proof: pressbrake

EVENT pressbrake $\widehat{=}$
  BEGIN
    act1 : brake := on
  END

Given:

$$(cc = on \Rightarrow brake = off)$$

Goal:

$$\{brake \mapsto on\}(cc = on \Rightarrow brake = off)$$
$$(cc = on \Rightarrow on = off)$$

- **Failure analysis:** revise the model so that $(cc = on) \Rightarrow false$
- **Modelling alternatives:**
  1. Add a guard of the form **cc = off** so as to restrict the applicability of **pressbrake**.
  2. Add an action of the form **cc:=off** so as to ensure that braking is not prevented by the state of **cc**.
  3. Add **cc = off** as an invariant of the system.
- Modelling heuristics required in order to rank alternative modelling suggestions, i.e. increase the productivity of designers.

A Priority Heuristic

- Where proof-failure analysis suggests that the value of a variable should be changed within the context of a given event, we adopt the following heuristics:
  - If the priority of the candidate variable is lower than the priorities of all the variables updated by the event, then it is strongly suggestive that the change should be achieved via a new action - **action speculation**
  - If the priority of the candidate variable is higher than the priorities of all the variables updated by the event, then it is strongly suggestive that the change should be achieved via a new guard - **guard speculation**
- A higher priority variable should not be restricted (guard) or changed (action) in order to achieve a change with respect to a lower priority variable.

Heuristics via Meta-data

MACHINE cruise_ctrl
VARIABLES brake, cc
INVARIANTS inv1: $cc = on \Rightarrow brake = off$
META $priority(cc) < priority(brake)$
EVENTS
INITIALISATION $\widehat{=}$
BEGIN
act1 : brake := off
act2 : cc := off
END
EVENT enable_cc $\widehat{=}$
BEGIN
act1 : cc := on
END
EVENT pressbrake $\widehat{=}$
BEGIN
act1 : brake := on
END
...
END

Reasoned Modelling Critics

critic (priority action speculation)

INPUTS:

PO_SET POs

MODEL_SET $\{M\}$

PRECONDITIONS:

1. $\exists failed\_po \in \{\langle \_, \_, \_, PO\rangle \in POs \mid failed\_proof(PO)\}.$
   $failed\_po = \langle M, E, \_/INV, (\Delta, X \Rightarrow Y \vdash \sigma(X \Rightarrow Y))\rangle$
2. $\exists \tau \in sub.\ disjoint\_sub(\tau, \sigma) \ \wedge \ provable(\Delta \vdash (\tau \cup \sigma)X \Rightarrow false)$
3. $priority(\tau, M) < priority(\sigma, M)$

OUTPUTS:

GUIDE $add\_action(sub2act(\tau), E, M)$

1. there exists an unproven invariant PO (implication)
2. there exists a variable update that does not interfere with the existing actions and which makes the PO provable
3. update does not violate priority heuristics

Reasoned Modelling Critics: User Guidance

MACHINE cruise_ctrl
VARIABLES brake, cc
...
EVENT pressbrake $\widehat{=}$
BEGIN
act1 : brake := on
END

MACHINE cruise_ctrl
VARIABLES brake, cc
...
EVENT pressbrake $\widehat{=}$
BEGIN
act1 : brake := on
act2 : cc := off
END

Note that the preconditions of the associated critic provides the justification for the generated user guidance.

Ongoing & Future Work

- More than just local analysis of proof failures, e.g. a global analysis of multiple failures may suggest invariant strengthening rather than guard speculation – *Abrial's "Cars on a Bridge" example, multiple failures.*
- Explore alternative notions of priority, e.g. the temporal nature of system priorities.
- Investigate how established notions of design assessment could be used to inform guidance, e.g. reduce the risk of premature commitment to design decision by ranking abstract suggestions over more concrete suggestions.
- Prototype implementation underway.
- Investigations are case study driven:
  - DEPLOY Project (EU ICT 214158), Michael Butler et al
  - `www.deploy-project.eu`

Ongoing & Future Work

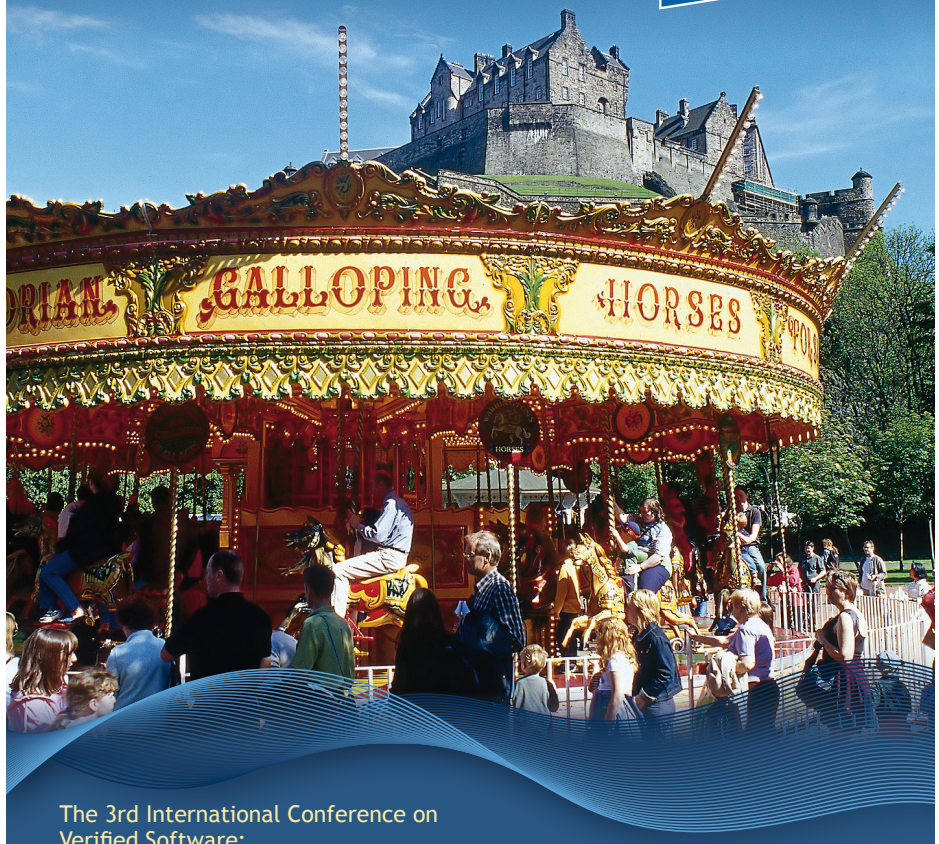
- Refinement patterns:
  - Currently developing **refinement methods** which combine common patterns of refinement at the level of models, gluing invariants and proof – Maria Teresa Llano (PhD).
  - Where a refinement method partially applies, critics will be used to suggest how the failure can be overcome via modelling suggestions.
  - Main focus is on posit-and-prove style refinement, but potential for guiding abstraction, refinement, decomposition and composition.
- Related EPSRC project:
  - AI4FM: AI for automatic proof search in Formal Methods.
  - Newcastle (EP/H024050), Edinburgh (EP/H024204), Heriot-Watt (EP/H023852)
  - `www.ai4fm.org`

Conclusion

- Proof planning has a successful track-record in terms of proof automation and automatic proof-failure analysis & proof patching.
- We aim to extend proof planning with a modelling layer, in particular to combine proof-failure analysis with common patterns of modelling in order to provide guidance.

VSTTE 2010

HERIOT WATT UNIVERSITY



The 3rd International Conference on Verified Software:
Theories, Tools and Experiments (VSTTE)
August 16th-19th, 2010
Heriot-Watt University, Edinburgh Campus
Scotland, UK

Key submission dates:

March 29th, 2010: Conference paper submission deadline

May 21st, 2010: Workshop paper submission deadline

Keynote speakers:

Tom Ball (Microsoft Research, Redmond)

Gerwin Klein (National ICT Australia)

Matthew Parkinson (University of Cambridge)

Web link:

http://www.macs.hw.ac.uk/vstte10

Email: vstte10@macs.hw.ac.uk

NSF EPSRC Engineering and Physical Sciences Research Council ALTRAN PRAXIS Microsoft Research sicsa* sse1 Software Systems Engineering Initiative

Analysis of Failed Invariant Proof: enable_cc

EVENT enable_cc $\widehat{=}$
  BEGIN
    act1 : cc := on
  END

Given:

$$(cc = on \Rightarrow brake = off)$$

Goal:

$$\{cc \mapsto on\}(cc = on \Rightarrow brake = off)$$
$$on = on \Rightarrow brake = off$$
$$brake = off$$

- **Failure analysis:** revise the model so that $(brake = off)$
- **Modelling alternatives:**
  1. Add a guard of the form **brake = off** so as to restrict the applicability of **enable_cc**.
  2. Add an action of the form **brake:=off** so as to ensure that the enabling of **cc** is not prevented by the state of **brake**.
  3. Add **brake = off** as an invariant of the system.

Reasoned Modelling Critics: User Guidance

MACHINE cruise_ctrl
VARIABLES brake, cc
...
EVENT enable_cc $\widehat{=}$
BEGIN
act1 : cc := on
END

MACHINE cruise_ctrl
VARIABLES brake, cc
...
EVENT enable_cc $\widehat{=}$
WHERE
grd1 : brake = off
BEGIN
act1 : cc := on
END