Fairness in Event-B

By
Xuedong Li
Supervisor: Kai Engelhardt
Assessor: Carroll Morgan

A thesis submitted for the degree of
FIXME

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

Computer Science and Engineering,
UNSW, Australia.

June 2014

Contents

0.1 Acknowledgement

Testing LTL: $\Diamond\Box\phi \Rightarrow \Box\Diamond\phi$.

Testing citations [Eng13, MP91, HA11, MP92, MP88].

0.2 Introduction

Testing LTL: $\Diamond\Box\phi \Rightarrow \Box\Diamond\phi$.

Testing citations [Eng13, MP91, HA11, MP92, MP88].

0.3 Background

0.3.1 Event-B

Event-B is a formal method for system-level modelling and analysis [?]. Event-B uses a notation resembling set theory for modeling systems and stepwise refinement to connect systems at different levels of abstraction. Mathematical proofs are required to justify consistency of system models as well as the preservation of safety properties proved at higher levels of abstraction down to lower levels.

In Event-B, machines are built to describe the static properties and track the dynamic behavior of a model. For a machine M, the variables V are declared in the `Variable` section. The type of the variables and also the additional constraints I for the variables are declared in the `Invariants` section. The possible state changes for the machine is defined by events. For an event E, additional external variables t are introduced in the `Parameter` section. The preconditions G which enable an event's execution are given by guards. The actions A implement the detailed changing between pre state and after state. Proofs are required to ensure the after state satisfy the invariants.

Every machine in Event-B must have an event INITIALIZATION (E_0) to set all the variables as the model start. E_0 does not have any guards nor parameters and the actions must cover the setup for all the variables.

However, in Event-B, only one event can happen at a time, which means original Event-B does not support concurrent execution.

In the rest parts of the thesis report, we will use following notions:

M : Machine

v : Variables

J : Invariant

For all event $i \in I$, I is the set of all events

E_i : The event i, where E_0 is the INITIALIZATION event

g_i: The guard of Event i

a_i: The action of Event i

Since the event INITIALIZATION is different to the others, and can be executed once only. We introduce set I_1 to store the events other than INITIALIZATION.

$$I = I_1 \cap \{0\}$$

0.3.2 Temporal Logic

Temporal logic is a system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time. With modalities referring to time, linear temporal logic or linear-time temporal logic (LTL) is a modal temporal logic. In LTL, we can encode formulas about the future of paths, one example of this is "one condition will be always true after the system satisfy another condition". Some LTL notations we will use in the rest parts of this thesis report are:
$\Box\phi$ Globally: ϕ always happen (hold on the entire subsequent path)
$\Diamond\phi$ Finally: ϕ eventually happen (hold somewhere on the subsequent path)
$\bigcirc\phi$ Next: ϕ will happen next (hold at next state)
In Event-B, ϕ can be a state of the machine, or a status that an event is chosen to be executed.
The LTL for a system is defined under the assumption of infinite execution. It does not make any sense for using LTL in a finite execution system (a system will automatic halt after limited executions).

0.4 Fairness-for-Event-B

0.4.1 Introduction

Fairness is a property for system which have infinite execution, this property ensure that transitions (events) can be "fairly" chosen to execute when their preconditions (guards) are satisfied. Without fairness constraints, it's possible that some transitions can be always ignored even if they are enabled in some states. It does not make sense to deal with fairness constraints on systems with finite execution.

Weak Fairness An transition set t is a weakly fair set if every transition in t is always eventually disabled or infinitely often taken.

$$\mathrm{WF}(i) \Leftrightarrow \Box\Diamond\neg g_i \vee \Box\Diamond \mathrm{taken}_i$$

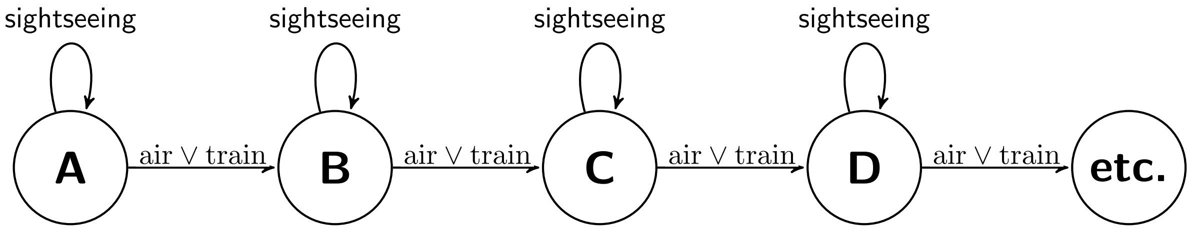
Strong Fairness An transition set t is a strongly fair set if every transition in t is eventually always disabled or infinitely often taken.

$$\mathrm{SF}(i) \Leftrightarrow \Diamond\Box\neg g_i \vee \Box\Diamond \mathrm{taken}_i$$

However, by working on the definitions for weak fairness and strong fairness, we can easily find that strong fairness imply weak fairness. Strong fairness is stronger than weak fairness.

$$\mathrm{SF}(i) \Rightarrow \mathrm{WF}(i)$$

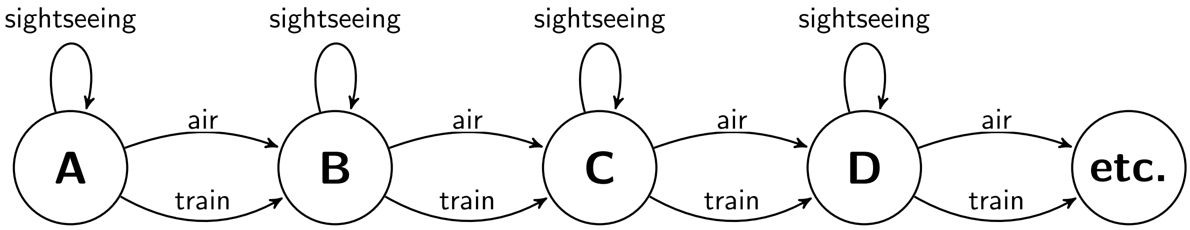
Event Merging For some situations we only have the fairness constraints for a set of events instead of fairness constraints for each of them. For example, Peter wants to travel all over the world, once he arrives at a new place, he might stay there and spend several days for sightseeing, and he can only take train or aiplane to travel. In this example, assume having an one-day sightseeing is modeled as the event $E_{\text{sightseeing}}$, the event E_{air} is for moving on air and the event E_{train} is for moving by train. We also introduce an event $E_{\text{air}\vee\text{train}}$, which is an event moving to another place either on air or by train.



The following constraint

$$\mathrm{WF}(air \vee train)$$

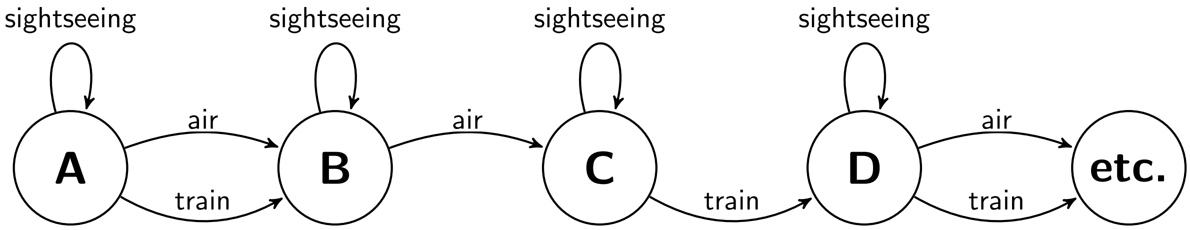
must be satisfied to ensure he does move from one place to another rather than spending the rst of his life sightseeing somewhere without ever moving again.



If every place has an airport and a train station, or none of them, travel is fair (can be always eventually chosen) iff air is fair (can be always eventually chosen) or train is fair (can be always eventually chosen). Since if one of the transportations is fair to choose, he can always eventually move to other places.

$$\mathrm{I} \wedge g_{air} = \mathrm{I} \wedge g_{train} \Rightarrow (\mathrm{WF}(air \vee train) \Leftrightarrow \mathrm{WF}(air) \vee \mathrm{WF}(train))$$

However, if there have at least one place that has only have airport, $\mathrm{WF}(train)$ can't ensure he can leave this place unless there have extra requirements force him occationally move on air (weak fairness for air when train is disabled). We also have similar problem for places only have train station.



We have:

$$(\mathrm{WF}(train) \wedge \mathrm{WF}(air@g_{air} \wedge \neg g_{train})) \vee (\mathrm{WF}(air) \wedge \mathrm{WF}(train@g_{train} \wedge \neg g_{air})) \Leftrightarrow \mathrm{WF}(t)$$

For this type of constraints, we introduce the following rule for merging events: For event E_i and event E_j, where $i, j \in \mathrm{I}$, the merged event $E_{i \vee j}$ is given as:

$$
\begin{aligned}
& g_{i \vee j} = g_i \vee g_j \\
& a_{i \vee j} | a'_{i \vee j} \wedge \\
& \qquad (((g_i \wedge \neg g_j) \Rightarrow (a'_{i \vee j} = a_i)) \\
& \qquad \vee ((\neg g_i \wedge g_j) \Rightarrow (a'_{i \vee j} = a_j)) \\
& \qquad \vee ((g_i \wedge g_j) \Rightarrow (a'_{i \vee j} = a_j \vee a_j)))
\end{aligned}
$$

The merged event $E_{i \vee j}$ can happen whenever either of its constituent events (E_i and E_j) could fire, and if $E_{i \vee j}$ fires, the effect is one of the enabled constituent events.
Some properties for fairness constraints of merged events are:
Assume $i, j \in \mathrm{I}$, event $E_{i \vee j}$ is the result of merging events E_i and E_j.

$$
(\mathrm{WF}(i) \wedge \mathrm{WF}(j @ g_j \wedge \neg g_i)) \vee (\mathrm{WF}(j) \wedge \mathrm{WF}(i @ g_i \wedge \neg g_j)) \Leftrightarrow \mathrm{WF}(i \vee j)
$$
$$
(\mathrm{SF}(i) \wedge \mathrm{SF}(j @ g_j \wedge \neg g_i)) \vee (\mathrm{SF}(j) \wedge \mathrm{SF}(i @ g_i \wedge \neg g_j)) \Leftrightarrow \mathrm{SF}(i \vee j)
$$

Event Splitting In an opposite way, we can also split an event into two mutually exclusive sub-events, by refining them with different mutually exclusive strengthened guards which imply the original guard.
For event h where $h \in I$, event i and event j are *split* events of h iff:

$$
g_i \vee g_j \Leftrightarrow g_h
$$
$$
\neg(g_i \wedge g_j)
$$
$$
a_i = a_j = a_h
$$

Note that the fairness constraints of the original event are not inherited to its sub-events, for the same reason explained in event merging part. However, similar to the outcome of the event merging process, we can say a event is fair if we can ensure that one of its sub-events is fair, since the sub-events have same guard as the original event.

$$
\mathrm{WF}(i) \vee \mathrm{WF}(j) \Rightarrow \mathrm{WF}(h)
$$
$$
\mathrm{SF}(i) \vee \mathrm{SF}(j) \Rightarrow \mathrm{SF}(h)
$$

Where event i and j are sub-events of event h.

Currently Event-B does not have suppport for LTL, most of properties described by

LTL (including fairness) can't be assumed or proved by Event-B. In the rest parts of this report, we will extend the original Event-B method and show the usage of fairness assumptions for proving response properties.

0.4.2 The Extended Event-B method

We need to make several modifications to support fairness and other properties defined in LTL. We need to ensure that the machine have infinite executions and add an area to store LTL related assumptions.

Infinite execution To ensure the infinite execution for a machine, the only thing we need to check that there is always at least one event (except event E_0) is enabled to be executed, i.e., the invariants J always satisfy at least one guards of events in I_1. We introduce the theorem thm_{InfExe} for infinite execution:

$$\bigvee_{i \in I_1} g_i$$

If thm_{InfExe} keep holding, always at least one event can be executed.

Assumptions After ensuring the infinite execution, we also introduce a new part in the machine, called `Assumptions`, which store the LTL based assumptions.
For event i where $i \in I_1$, available assumptions are:
Weak Fair:

$$\mathrm{WF}(i)$$

and Strong Fair:

$$\mathrm{SF}(i)$$

and Response:

$$p \leadsto q$$

For state p and q, available assumption where $i \in I_1$, and p and q are states. Details about response property are in the next section.

0.4.3 Proving rules for response properties in Extended Event-B

Response property describes a relationship between two states p and q, we say $p \leadsto q$ iff q will always eventually happen once after p happened.

$$p \leadsto q \Leftrightarrow (p \Rightarrow \Diamond q)$$

Response have following general properties:
Reflexivity:

$$a \leadsto a$$

Transitivity:

$$\frac{p \leadsto q, q \leadsto r}{p \leadsto r}$$

Monotonicity:

$$\frac{p \Rightarrow p', p' \leadsto q', q' \Rightarrow q}{p \leadsto q}$$

Disjunction:

$$\frac{p \leadsto r, q \leadsto r}{(p \vee q) \leadsto r}$$

In addition, we need rules for specific situations. Before applying following rules, we assume the thm_{InfExe} and Event-B usual POs have been proved already.

Proof rule for the only events For state p, we say event E_i is the only event for p, written $(\text{Only}_i(p))$, if at state p, event i is the only event enabled.

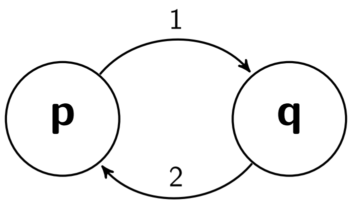
$$\frac{\begin{array}{c} i \in I_1 \\ p \in J \wedge g_i \\ p \notin J \wedge \bigvee_{j \in (I_1 \setminus \{i\})} g_j \end{array}}{\text{Only}_i(p)}$$

We introduce the Only set, $\text{Only}(i)$, which satisfy

$$\text{Only}_i(p) \Leftrightarrow p \in \text{Only}(i)$$

where for all $i \in I_0$

$$\text{Only}(i) = J \wedge g_i \wedge \bigvee_{j \in (I_1 \setminus \{i\})} g_j$$



An example is a machine presented by graph above. In this machine, event 1 is the only event at state p and event 2 is the only event at state q, $\mathrm{Only}(1) = p$ and $\mathrm{Only}(2) = q$. Now we want to prove $p \leadsto q$.
Since at state p we can choose event 1 only, which will move from p to q directly, so that q must be the next state after p, which satisfies $p \leadsto q$.

$$\mathrm{Only}_i(p) \land (p \land a_i = q') \Rightarrow (p \Rightarrow \bigcirc q) \Rightarrow (p \Rightarrow \Diamond q) \Leftrightarrow p \leadsto q$$

In general, we have rule $\mathrm{RESP}_{\mathrm{ONLY}}$

$$\frac{\begin{array}{c}\mathrm{Only}_i(p)\\ J \land g_i \land p \land a_i = q'\end{array}}{p \leadsto q}$$

Proof rule for the weakly fair events Manna and Pnueli had present a proved single-step rule to validate response properties under weak fairness assumption for the helpful transition. Their rule is:

$$\frac{\begin{array}{c}p \Rightarrow (q \lor \varphi)\\ \forall \tau \in T(\rho_\tau \land \varphi) \Rightarrow (q' \lor \varphi')\\ (\rho_{\tau_h} \land \varphi) \Rightarrow q'\\ \varphi \Rightarrow (q \lor \mathrm{En}(\tau_\mathrm{h}))\end{array}}{p \leadsto q}$$

where T is the set of transitions, τ_h is the helpful transition, $\mathrm{En}(\tau_\mathrm{h})$ is the precondition enable τ_h to be taken and ρ_τ is the after state for transition τ.
In Event-B, the transition set T can be represented by the events set I_1, the helpful transition τ_h can be represented by the helpful event h, the precondition $\mathrm{En}(\tau_\mathrm{h})$ can be represented by $J \land g_h$, and the after state ρ_τ can be represented by $J \land g_(i) \land a_i$. By

replacing φ by ϕ, the Event-B version of the rule (RESP_{WF}) is:

$$\frac{\begin{array}{c} J \wedge p \Rightarrow q \wedge \phi \\ \forall i \in I_1(J \wedge g_i \wedge a_i \wedge \phi \Rightarrow q' \vee \phi') \\ \text{WF}(h) \\ J \wedge g_h \wedge a_h \wedge \phi \Rightarrow q' \\ J \wedge \phi \Rightarrow q \vee g_h \end{array}}{p \leadsto q}$$

Proof rule for the strongly fair events The single-step rule for strong fairness is similar to the one for weak fairness. However, if a helpful event h is strongly fair, we only need $J \wedge \phi \Rightarrow \diamond(q \vee g_h)$, instead of $J \wedge \phi \Rightarrow q \vee g_h$, since SF($h$) ensure h can be eventually chosen if $\diamond g_h$ applies.
From the definition of response, we have

$$J \wedge \phi \Rightarrow \diamond(q \vee g_h)$$
$$\Leftrightarrow$$
$$J \wedge \phi \leadsto q \vee g_h$$

After the modification, the rule for strong fairness (RESP_{SF}) is:

$$\frac{\begin{array}{c} J \wedge p \Rightarrow q \wedge \phi \\ \forall i \in I_1(J \wedge g_i \wedge a_i \wedge \phi \Rightarrow q' \vee \phi') \\ \text{SF}(h) \\ J \wedge g_h \wedge a_h \wedge \phi \Rightarrow q' \\ J \wedge \phi \leadsto q \vee g_h \end{array}}{p \leadsto q}$$

The Well-Founded rule The above rules can only handle the proof established by a single step. We can figure out a sequence of single-step proofs, and build a prooving chain from these proofs to prove response with fixed number of steps. However, we can't simplily apply this idea to the proofs with unknown amount of steps, since we need to determine a single-step proof chain, which is impossible for unknown amount steps. To establish this type of response proving, we introduce a chain proof rule based on the Well-Founded Structure.
We define a well-founded structure in form of $(A, B, \succ)$ where
A is a set of elements.
B is a subset of A.

$\succ$ is a binary relationship defined on A, and $\succ$ restricted to B is well founded, i.e., starting at a fixed value $b_0 \in B$, there does not exist an infinite sequence of elements of elements of B which satisfy that

$$b_0 \succ b_1 \succ b_2 \succ ...$$

One example of a well-founded structure is $(\mathbb{R}, \mathbb{N}, >)$, where $\mathbb{R}$ is the set of all real numbers, $\mathbb{N}$ is the set of non-negative integers, and $>$ is the greater than relation. For any fixed $b_0 \in \mathbb{N}$, there have no more than b_0 non-negative integers less than b_0, so that the length of all the possible sequences start at b_0 are no more than b_0, which is finite. So that the structure $(\mathbb{R}, \mathbb{N}, >)$ is a well-founded structure.
Now we introduce the rank function $r(s)$, which is a mapping from state s to set A of a well-founded structure $(A, B, \succ)$.
For the state set $S = \{s_0, s_1, s_2, ...\}$, $\phi = \bigvee_{s \in S} s$
for all $s_p, s_q \in S$,

$$\frac{\begin{array}{c} r(s_p) \in B \\ r(s_q) \in B \\ \neg\Box s_p \\ s_p \rightsquigarrow s_q \Rightarrow (s_p = s_q) \vee (r(s_p) \succ r(s_q)) \end{array}}{\neg\Box\phi}$$

Proof:
Assume that start at state $s_0 \in S$, ϕ keep holding to state s_{n-1}, for state s_n, which next to the state s_{n-1}, we have

$$\begin{gathered} r(s_{n-1}) \in B \\ r(s_n) \in B \\ \neg\Box s_{n-1} \\ s_{n-1} \rightsquigarrow s_n \Rightarrow (s_{n-1} = s_n) \vee (r(s_{n-1}) \succ r(s_n)) \end{gathered}$$

Since $\neg\Box s_{n-1}$, we have $\Diamond(s_{n-1} \neq s_n)$, which implies $\Diamond(r(s_{n-1}) \succ r(s_n))$.
However, since $r(s_{n-1}) \in B \wedge r(s_n) \in B$, number of the possible choices for $r(s_n)$'s value applying $r(s_{n-1}) \succ r(s_n)$ is finite, and decreases once s_n changes. Eventually $r(s_n) \in B$ can't hold anymore, which means eventually ϕ can't hold anymore.

Now we introduce the Well-Founded rule for response (RESP_{WFR}):

$$\frac{\begin{array}{c} J \wedge p \Rightarrow q \wedge \phi \\ \phi \rightsquigarrow \phi \vee q \\ \forall s \cdot s \Rightarrow \phi(r(s) \in B \wedge \neg\Box s) \\ \forall s_p, s_q \in S(s_p \rightsquigarrow s_q \Rightarrow (s_p = s_q) \vee (r(s_p) \succ r(s_q))) \end{array}}{p \rightsquigarrow q}$$

Proof:

$$\frac{\begin{array}{c} \forall s \cdot s \Rightarrow \phi(r(s) \in B \wedge \neg\Box s) \\ \forall s_p, s_q \in S(s_p \rightsquigarrow s_q \Rightarrow (s_p = s_q) \vee (r(s_p) \succ r(s_q))) \end{array}}{\neg\Box\phi}$$

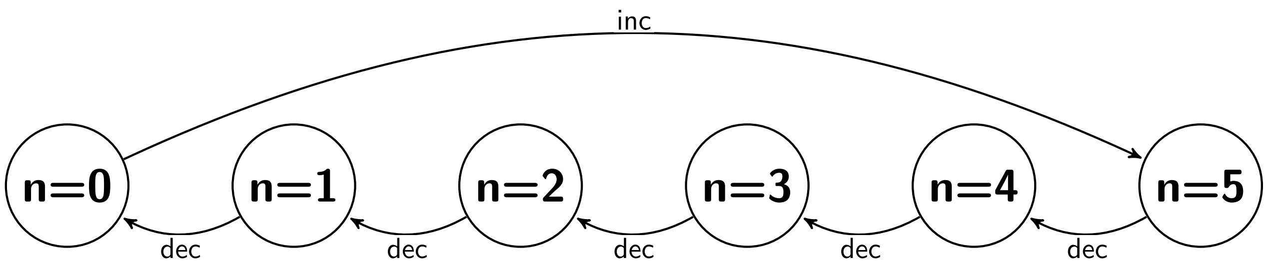
$$\frac{\begin{array}{c} \neg\Box\phi \\ \phi \rightsquigarrow \phi \vee q \end{array}}{\phi \rightsquigarrow q}$$

$$\frac{\begin{array}{c} p \rightsquigarrow \phi \\ \phi \rightsquigarrow q \end{array}}{p \rightsquigarrow q}$$

0.5 Sample proofs for response properties

0.5.1 Proof 1

In this subsection, we want to prove $n = 1 \rightsquigarrow n = 3$ for Machine M0 below.



For machine M0, we know that:

$$J = (n \in \mathbb{N})$$
$$I_1 = \{inc, dec\}$$
$$g_{inc} = \{n|n < 1\}, a_{inc} = (n' = n + 5), \text{Only}(inc) = \{n|n < 1\}$$
$$g_{dec} = \{n|n > 0\}, a_{dec} = (n' = n - 1), \text{Only}(dec) = \{n|n > 0\}$$
$$thm_{InfExe} : (n < 1 \vee n > 0)$$

Infinite execution proof for M0 Since $J \Leftrightarrow (n \in \mathbb{N}) \Rightarrow (n < 1 \vee n > 0) \Leftrightarrow thm_{InfExe}$, M0 have infinite execution.

Response proof for M0 For proving $n = 1 \rightsquigarrow n = 0$ we have:

$$p = (n = 1), q = (n = 0)$$
$$(n = 1) \in \text{Only}(dec) \Leftrightarrow \text{Only}_{\text{dec}}(n = 1)$$

By applying to rule $\text{RESP}_{\text{ONLY}}$, we have:

$$\frac{\begin{array}{c}\text{Only}_{dec}(n = 1)\\ J \wedge g_{dec} \wedge (n = 1) \wedge a_{dec} = (n' = 0)\end{array}}{n = 1 \rightsquigarrow n = 0}$$

Similarly, we can prove $n = 5 \leadsto n = 4$ and $n = 4 \leadsto n = 3$.
For proving $n = 0 \leadsto n = 5$ we have:

$$p = (n = 0), q = (n = 5)$$
$$(n = 0) \in \mathrm{Only}(inc) \Leftrightarrow \mathrm{Only}_{\mathrm{inc}}(n = 0)$$

By applying to rule $\mathrm{RESP}_{\mathrm{ONLY}}$, we have:

$$\frac{\begin{array}{c}\mathrm{Only}_{inc}(n = 0) \\ J \wedge g_{inc} \wedge (n = 0) \wedge a_{inc} = (n' = 5)\end{array}}{n = 0 \leadsto n = 5}$$

Since we proved $n = 1 \leadsto n = 0$, $n = 0 \leadsto n = 5$, $n = 5 \leadsto n = 4$ and $n = 4 \leadsto n = 3$, by applying to the transitivity of response, we get $n = 1 \leadsto n = 3$. Testing citations [Eng13, MP91, HA11, MP92, MP88].

MACHINE M0

VARIABLES

n

INVARIANTS

`inv1` : $n \in \mathbb{N}$

thm_InfExe : $n < 1 \vee n > 0$

EVENTS

Initialisation

begin

`act1` : $n := 0$

end

Event *inc* $\widehat{=}$

when

`grd1` : $n < 1$

then

`act1` : $n := n + 5$

end

Event *dec* $\widehat{=}$

when

`grd1` : $n > 0$

then

`act1` : $n := n - 1$

end

END

References

[Eng13] Kai Engelhardt. Proving response properties of event-b machines. Lecture notes for COMP2111, accessed at 2013/11/12, `http://www.cse.unsw.edu.au/~cs2111/13s1/lec/note09.pdf`, May 2013.

[HA11] Thai Son Hoang and Jean-Raymond Abrial. Reasoning about liveness properties in Event-B. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*, volume 6991 of *LNCS*, pages 456–471. Springer-Verlag, 2011.

[MP88] Zohar Manna and Amir Pnueli. The anchored version of the temporal framework. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 201–284. Springer-Verlag, 1988.

[MP91] Zohar Manna and Amir Pnueli. Tools and rules for the practicing verifier. In Richard F. Rashid, editor, *Computer Science: A 25th Anniversary Commemorative*, pages 121–156. ACM Press and Addison-Wesley Publishing Co., 1991.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.