A (Proposal for a) Rodin Plugin for Timed Machine

Joris Rehm

LORIA - Nancy Université

July 17, 2009

Context

- Real-time systems: worst case computation time must be under some duration. Example: a brake by wire system, the systems is not aware of time but must react fast. (See real-time controler or OS, scheduling)
- Timed systems: the system itself use time (quantitative temporal) properties. Example: the 2-slot Simpson algorithm use delays to ensure correct access to a shared memory. Or time is a part of the specification (example: pacemaker).

Context 2

- The goal: be able to verify timed properties in the Event-B world.
- Few (or no) change to the formalism. We will use a refinement pattern to introduce the model of time.

Time pattern

- I tried many explicit encoding of time in Event-B. (As the pattern is applied many time in a model, it can have great consequence over the proof complexity)
- Finally choose to reason about the timed elapsed since the last triggering of an event.
- Operator $S(e)$, "the duration elapsed since last run of the event e"

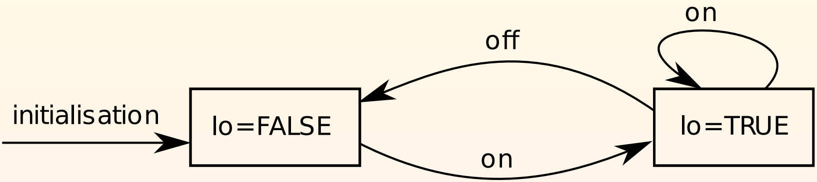
The plug-in requirements

- The user add timed invariant involving the operator S.
- The user can constraint an event f by adding lower bounds on the duration elapsed since another event f run $(X \leq S(e))$
- The user can constraint an event f by adding upper bounds on the duration elapsed since another event f run $(S(e) \leq Y)$
- (lower and upper bounds are really different)

The plug-in requirements

- The user can see and edit a timed machine.
- The user can save his timed machine.
- The plug-in can apply the time pattern and generate a normal machine (the proof is carried on this machine).

(Very Simple) Example : light timer



Untimed example

on $\widehat{=}$
Begin
act1: $lo := TRUE$
End

off $\widehat{=}$
When
grd1: $lo = TRUE$
Then
act1: $lo := FALSE$
End

Refinement: adding time

off $\widehat{=}$

When

grd1: $lo = TRUE$

Lower Bound

lb_off: $c - d \leq S(on)$

Upper Bound

ub_off: $S(on) \leq c + d$

Then

act1: $lo := FALSE$

End

TIMED INVARIANTS

ti1: $lo = TRUE \Rightarrow S(on) \leq c + d$

ti2: $lo = FALSE \Rightarrow c - d \leq S(on)$

Data refinement

- A total function f with a constant finite set E as domain can be refined to several ($card(E)$) variables.
- $f \in \{a, b, c\} \rightarrow F$
- we can consider variables f_a, f_b, f_c instead of expression $f(a), f(b), f(c)$
- Therefore we have two options for encoding the operator S: with a function $s(a), s(b), \ldots$ or with a set of variable $s\_a, s\_b, \ldots$.

The generated machine

on $\widehat{=}$
Begin
act1: $lo := TRUE$
act2: $s\_on := 0$
End

off $\widehat{=}$
When
grd1: $lo = TRUE$
lb_off: $c - d \leq s\_on$
Then
act1: $lo := FALSE$
End

The generated machine

tic $\widehat{=}$

Any shift

Where

grd1: $0 < shift$

ub_off: $lo = TRUE$

$\Rightarrow s\_on + shift \leq c + d$

Then

act1: $s\_on := s\_on + shift$

End

The general pattern

reset $\widehat{=}$

Any e

Where

grd1: $e \in E$

Then

act1: $s(e) := 0$

End

tic $\widehat{=}$

Any $shift$

Where

grd1: $0 < shift$

Then

act1: $s := \{e \cdot e \in E | e \mapsto s(e) + shift\}$

End

Data refinement

tic $\widehat{=}$

Any *shift*

Where

grd1: $0 < shift$

Then

act1: $s\_a := s\_a + shift$

act2: $s\_b := s\_b + shift$

act3: $\ldots$

End

Only a matter of taste?

- In the general case, to increment the variable in the tic event we can define an *add* function in a context.
- Which one do you prefer, and which one is better (currently) for Rodin?
- Wishlist item for the wiki: more rewriting rules for that kind of expressions

typ: $add \in \mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z})$
axm1: $add = (\lambda a \cdot a \in \mathbb{Z} | (\lambda b \cdot b \in \mathbb{Z} | a + b))$
axm2: $add = \{a \cdot a \in \mathbb{Z} | a \mapsto \{b \cdot b \in \mathbb{Z} | b \mapsto a + b\}\}$
axm3: $\forall a, b, c \cdot a \in \mathbb{Z} \land b \in \mathbb{Z} \land c \in \mathbb{Z} \Rightarrow$
$(a \mapsto b \in add(c) \Leftrightarrow b = a + c)$
axm4: $\forall a, b \cdot a \in \mathbb{Z} \land b \in \mathbb{Z} \Rightarrow (add(a)(b) = a + b)$

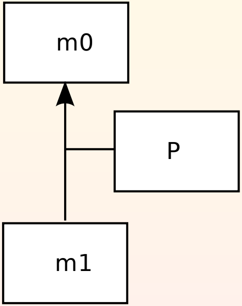
In case of distributed system

- Let's take a set of distributed devices, for example trains on the rail network or devices sending messages on a network.
- Typically in Event-B, all distributed entities share the same set of events and a parameter of the event give the involved device.
- Therefore for the operator S we need an additional parameter in order to refer to the last triggering of a particular device. For example $S(on(x))$ with x a device.

In case of distributed system

- In this case it's not possible to use data refinement shown before (as the number of devices is unknown)
- The pattern should be a little bit refined to represent the S operator with parameter.
- and the plug-in could manage that (probably not in first version)

What is a pattern (for us)?



- It's a double refinement (with no shared variables)
- The pattern is adapted (elements are replicated and renamed) and inserted into the studied model to make a new refinement
- The plug-in must ensure that the usage of the pattern is really a refinement.
- (It may be useful to extend rodin for modeling the extra refinement link between m1 and P.)

The generation procedure

- The plug-in must apply the pattern with our without the data simplification, as requested by the user (and if it's possible).
- For lower bound or invariant just replace the operator S by the encoding in normal variable.
- Each event that appears in S must reset the counter to zero.
- The progression of the time is encoded in a *tic* event
- For upper-bound $S(e) \leq X$ in a event f, the following guard must be added to the *tic* event:

$$GUARD(f) \Rightarrow S(e) + shift \leq X$$

Arithmetic

- High needs of better arithmetic provers
- The real numbers can be useful to have dense time model
- (actually it's simpler to do automated proofs on real numbers than on integer)

Conclusion

- It's quite fascinating to see that with only a few changes we can have timed machine.
- That means that refinement PO and invariant PO are enough powerful to encode (nicely) a model of time.
- Hope to complete the plugin's implementation as soon as possible (thanks for the nice tutorial).