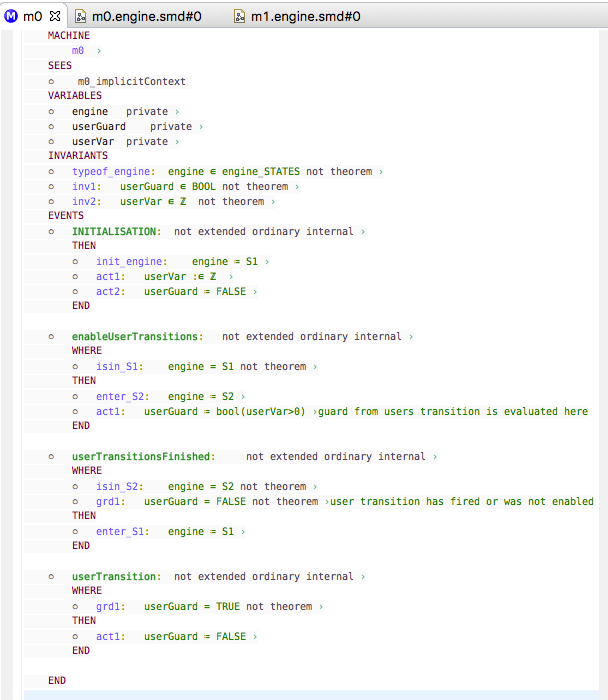
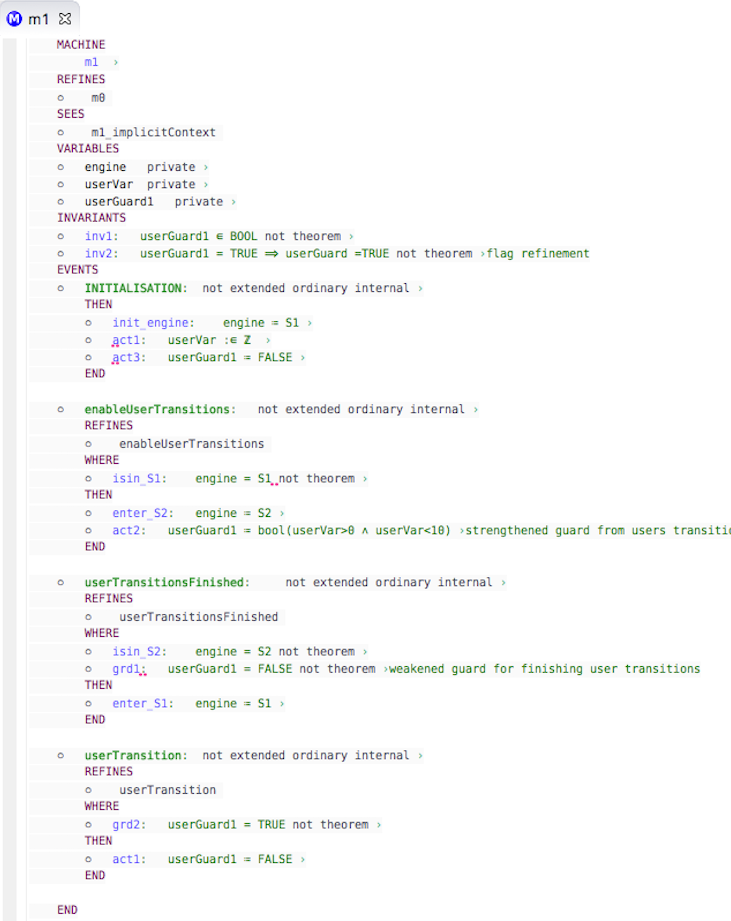
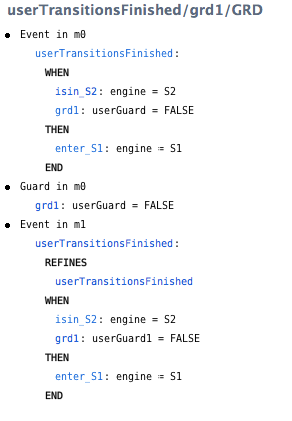
# Discussion on the problem with refining big-step/little-step semantics

A distilled example:





Unprovable guard weakening



In the example, the user's SCXML source model has one transition - i.e. represented here by event  ‘userTransition’ with a guard ‘userVar >0'

in the first refinement the user has strengthened the guard for ‘userTransition’ by adding a second conjunct..  ‘userVar >0 & userVar <10'.

However, to simulate the run to completion big-step we introduce the following events:-

‘enableUserTransitions' which atomically evaluates the guards of all the user transitions and enables those that should fire using BOOL flags (e.g. ‘userGuard1'). [Normally this event would also consume a trigger which affects the guards but i have omitted triggers here].

‘userTransitionsFired’ which waits for all the user transitions to be disabled (i.e. completion) before initiating a new big-step

It is this wait for completion which i think is a problem for refinement because it is guarded by 'not(userGuard1=TRUE)' which is weaker than 'not(userGuard=TRUE)'.

i don’t think the problem is caused by my choice of encoding of the semantics, it was the same when we did a much more abstract representation. Unless you can think of a way to represent this completion semantic without introducing an event that is guarded by the negation of the user guards?

I think it is a fundamental conflict between guard strengthening refinement and run to completion semantics (which is used by all the common Harel state-chart modelling languages s.a. UML, SysML SCXML). The stronger the guards the weaker completion becomes.

Hi Colin

You are right, it is not a refinement.

Consider a simple example with user transition UT.  The abstract UT has guard ‘true' and the refined UT has guard ‘false'.  The refined encoding has event trace:

[ enableUserTransitions, userTransitionsFired ]

which is not a trace of the abstract model.  The abstract model has trace:

[ enableUserTransitions, UT, userTransitionsFired ]

What is the motivation for using this more elaborate encoding.  I can see why it is used for simulation, but is it relevant for a proof approach?

Michael

What is the motivation for using this more elaborate encoding.

At first we didn’t model run to completion (R2C) semantics at all. This would be a problem for Engineers that expect a different semantics to their models.

Then we did a very abstract encoding of R2C that also had this same problem. This was closer but still didn’t give engineers what they are expecting.

I can see why it is used for simulation, but is it relevant for a proof approach?

The overall goal is to provide a notion of refinement in R2C state-charts.

For that I presume we need to

a) model the user model according to R2C semantics

b) prove the refinement of the models

but perhaps it is a mistake to try to do trace refinement of both the model and the simulator engine. I.e. We don’t really care about the R2C simulator artefact events themselves, they just implement the grouping of transition firing in each step. So if we ignore them we have

[]

which is a refinement of

[UT]

The order of firing of user transitions within a step is unspecified, so we want to prove the refinement of step traces like

[STEPm, STEPn, … ]

where each step is

STEPi = ||| Tij,

where Tij  (j: 1..n) are the n transitions that are enabled in step i.

Since we know that the completion event doesn’t change any user variables, can't we just ignore the problematic completion PO?

i.e. The POs for a user transition T still ensure that

if  STEPi\_n+1  <:  STEPi\_n

and this  implies

StepTrace\_n+1 <: StepTrace\_n

, which is enough to prove the refinement of the user model

Colin

So the simulation events do not need to be related by refinement.  How about you make the refined simulation events refinements of skip and refine the abstract events by ‘miracle’ (an event with a false guard).  Does that work?

Michael

Thanks, that is a nice idea!  I will try it.

But i think we may lose the link between the step traces completely.

i.e. the simulation events (and their simulation state variables) provide the grouping of transitions within each step.

If they refine skip don’t I have to change the simulation state variable each refinement? And hence lose the link between the abstract and refined step trace.

I am beginning to think we need a new PO generator! - i.e. a formal semantics for SCXML refinement.

Colin

You need to think about what refinement should mean in a big-step based semantics.  By big-step I mean the grouping of transitions into a big-step and preventing the next step from happening until all the transitions in the current step have executed (completion).

Solution 1:

Currently a big step is only enabled when all the transitions of the previous step have executed, right?  In a refinement you are weakening the enabling of the next step since you allow it to be enabled when a smaller set of transitions of the previous set have executed.  Maybe you should not allow a reduction in the set of transitions in each big step in a refinement?  This would mean you need to be more careful about guard strengthening in refinement: if you strengthen a guard, then you must introduce a new convergent event within the same step that covers the condition that you have removed in the guard strengthening, i.e.,

grd(E) =>  grd(New) or grd(E’)

In concrete states corresponding to grd(E) eventually New will become disabled (since it is convergent) and the above PO means that grd(E’) will hold.

But this means you cannot evaluate the guards of the events at the beginning of the step: typically E’ is not enabled at the beginning of the step but is enabled by New.  This may be an issue in any case?

Solution 1 requires the above enabledness PO to be generated.

Solution 2:

An alternative way of thinking of the enabling of a big step is to weaken it at the abstract level: require that a (non-empty?) subset of the transitions have executed in order to enable the next step. You should also block the transitions of the current step that have not yet executed once the decision to move to the next step is made.  This makes it explicit at the abstract level that a valid implementation is one that allows a reduced set of transitions to be executed within a big step.

Solution 2 works with the existing POs.

I suspect the engineers you are trying to satisfy would prefer Solution 1?

Regards

Michael

Solution 1 doesn’t fit my understanding of the big step semantics. i will check but I think the new events cannot enable E’ within a single big step. Assuming this is correct we want New to converge over several big steps rather than within one big step. Hence solution 2 does exactly what we want. In refinements we elaborate the preliminary steps that replace the phantom big steps.

The only odd thing for the Engineer is that the simulation can freewheel at the abstract levels.. but that is a true reflection of the abstraction.  To make the simulation nicer for the long-suffering Engineer, perhaps we could have a concept of ‘final’ for a transition which means it IS required for the next big step. If final is set, the transition guard cannot be strengthened any more. grd(E) <=>  grd(E’) (with a static check that final is never revoked).

Colin