Sequence Refinement

A. Iliasov

August 6, 2010

According to the Event B refinement laws, new behaviour is introduced by adding new events to a refinement machine. Such new behaviour may correspond to a detalised description of some abstract behaviour especially when a previously atomic state transition has to be explained as a chain of state transitions. To satisfy the refinement relation, new events are only allowed to update new variables. New variables of a machine denote the part of a system state that is not mapped into the state of the abstract system, that is, they characterise a hidden state of a refined system. Since new events cannot be directly related to the abstract behaviour it is necessary to have a housekeeping event that would link the behaviour expressed on the hidden state with the refined abstract behaviour. Technically, the housekeeping event is a refinement of some abstract event. Such housekeeping event is discarded during the transition to an executable program source. The need to introduce auxiliary variables to connect housekeeping events with new events if an additional inconvenience since one would have to take care to ignore them in the subsequent refinements and disregard when building a runnable program.

We propose to consider an alternative set of refinement laws where an abstract state transition may be directly refined into a chain of new transitions. In other words, a refinement relation relating an abstract event and new events without the need for a housekeeping event and auxiliary variables. We not looking to replace or remove any of the existing refinement laws but rather augment them with a new set of laws.

The approach is best demonstrated by a simple example. The abstract model specifies single event *swap* that swaps the values of two variables.

MACHINE m
  VARIABLES a, b
  INVARIANT $a \in \mathbb{N} \land b \in \mathbb{N}$
  INITIALISATION $a :\in \mathbb{N} \parallel b :\in \mathbb{N}$
  EVENTS
    swap = BEGIN $a, b := b, a$ END
END

In the refinement we want to eliminate the case of assigning to two variables at the same time (e.g., the target architecture does not support such an operation). Instead, the swap is modelled using the addition and subtraction operations. The general idea is to introduce a pair of new variables, realise the swap algorithm on these variables, and then use the refinement of event *swap* in the role of a housekeeping event to prove the refinement relation. A refined model takes the following form.

REFINEMENT m1a
REFINES m
VARIABLES a, b, x, y, pc
INVARIANT
$x \in \mathbb{N} \land y \in \mathbb{N} \land pc \in 0 \ldots 3$
$pc = 0 \implies x = a \land y = b$
$pc = 1 \implies x = a + b \land y = b$
$pc = 2 \implies x = a + b \land y = b$
$pc = 3 \implies x = b \land y = a$
INITIALISATION
$a, x :| a' \in \mathbb{N} \land x' = a'$
$b, y :| b' \in \mathbb{N} \land y' = b'$
$pc := 0$
EVENTS
step1 = WHEN $pc = 0$ THEN $x := x + y \parallel pc := 1$ END
step2 = WHEN $pc = 1$ THEN $y := x - y \parallel pc := 2$ END
step3 = WHEN $pc = 2$ THEN $x := x - y \parallel pc := 3$ END
swap = WHEN $pc = 3$ THEN $a, b := x, y \parallel pc := 0$ END
END

The $pc = \ldots$ invariants are necessary to propagate the information on state evolution through the steps of the algorithm. Without these, it would be impossible to prove the action refinement of *swap*.

Let us consider an alternative refinement model where there no auxiliary variables and no housekeeping event.

REFINEMENT m1b
REFINES m
VARIABLES a, b
INVARIANT $a \in \mathbb{N} \land b \in \mathbb{N}$
INITIALISATION $a :\in \mathbb{N} \parallel b :\in \mathbb{N}$
EVENTS
swap1 = BEGIN $a := a + b$ END
swap2 = BEGIN $b := a - b$ END
swap3 = BEGIN $a := a - b$ END
END

Not shown above but present in the model is the information on the refinement relationship among the *swap_* and *swap*, and the restriction of the possible traces of *swap_* events. The latter is necessary as the refinement relationship holds only when *swap1* is followed by *swap2* and the by *swap3*. The traces restriction information is not encoded in the state of the model (that is, by auxiliary variables) but rather taken into the account when generating proof obligations.

Due to space constraints we are unable to give further details on the proof obligations and how and why they are compatible with the existing set of refinement laws. However, a beta version of the plugin is available for those interested to learn more about this work [1].

References

[1] "Sequence refinement plugin update site," http://iliasov.org/refplugin/.