## SCXML semantics

The SCXML semantics are described operationally here:-

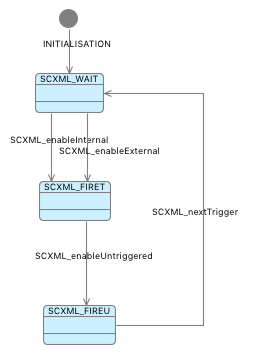
Our (abstract) interpretation is as follows

1. Initialise
2. Take one event from the internal queue and fire the set of transitions that are enabled by it at that time. I.e. they must be enabled at the time it checks, not subsequently after firing one or more of the set of transitions.
3. Fire the set of un-triggered transitions that are enabled after step 1). Again they must be enabled at the time it checks, not subsequently.
4. If the internal queue is not empty, repeat from 1)
5. If the internal queue is empty, and the external queue is empty, wait for an external event to arrive.
6. If the external queue is not empty, take one event from the external queue and fire the set of transitions that are enabled by it at that time. Again they must be enabled at the time it checks, not subsequently after firing one or more of the set of transitions.
7. repeat from 1)

There are some issues with this interpretation

1. The system cannot start because initially the internal queue will be empty.
2. The system cannot progress through a sequence of un-triggered transitions unless irrelevant external/internal triggers arrive. Therefore it is invalid to have more than one un-triggered transition without a triggered one in between.
3. Irrelevant triggers make the system progress through un-triggered transitions.

## SCXML engine

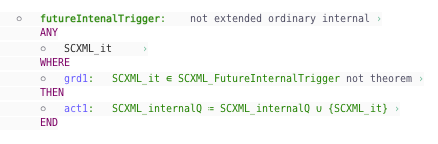
A *basis* machine and context are provided to define the generic elements of the SCXML engine. Specific translated SCXML models start by refining/extending this basis. The basis provides a cyclic engine representing an abstraction of the semantics of SCXML. There are two trigger queues; internal and external. (Currently these are modelled as sets. It would be more accurate to make them queues).

Initially the engine waits for triggers. When a trigger is present, internal triggers have precedence over external ones. A trigger is consumed and used to evaluate the enabledness of the transitions according to the current state of the SCXML application state chart and any other conditions involving system data. A Boolean flag is constructed for each transition recording its enabledness. This flag is used as the guard for the Event-B event that represents the corresponding SCXML transition. The engine then waits for all the transitions it enabled to fire. The transitions set new state and data as well as resetting their enabled flag. They may also raise new internal triggers by adding them to the internal queue.

When all flags are reset to FALSE, the engine evaluates the enabledness of the transitions that are not triggered. This is done in a similar way using Boolean flags. When all un-triggered transitions have fired the engine goes back to the start and consumes another trigger from the internal queue if there is one, or from the external queue if not.

In the basis machine, since the specific SCXML model and transitions are not present, the flags and guards evaluation is missing. The basis provides a starting point so that these engine events can be extended with the specific model information described above.

## Future Triggers

The basis also provides a mechanism for introducing new triggers. Since these triggers require changes to the trigger queues, which are present from the abstract basis, they cannot be handled by ‘new’ events. Abstract ‘futureInternalTrigger’ and ‘futureExternalTrigger’ events are provided for them to refine. A set of ‘FutureInternalTrigger’ and ‘FutureExternalTrigger’ are provided as an abstraction of the triggers that may be introduced in the future.

## 

New triggers are introduced by partitioning this set, leaving a residual for further future triggers



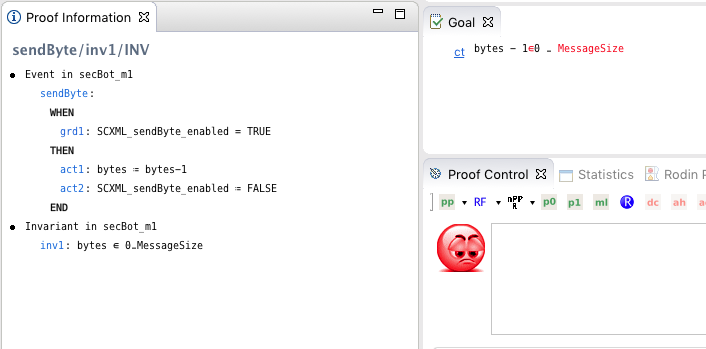
## Refinement of triggers

Note that internal triggers must be introduced as such. They cannot be introduced as external triggers and then later refined as internal ones. This is because the categorisation affects the priority and ordering of their consumption and hence the trace of events which is the definition of refinement. I.e. it is inherently not a refinement to replace an external trigger with an internal trigger. However, an internal trigger can be fired non-deterministically by an un-triggered and unguarded transition. This means that it would be available to fire at any time to enter the event in the queue and later constrained to a specific transition but that the processing of it by the engine would be consistent in future refinements.

## Proof that transitions satisfy invariants

Removing the guards to the SCXML engine means that it is less obvious that transitions satisfy invariants.

For example, the transition sendByte is enabled by the engine when bytes>0. and bytes is not decreased elsewhere. However, the prover cannot initially discharge the PO concerning the range of bytes.



Adding an invariant as follows allows automatic proof.

⚬ inv6: SCXML\_sendByte\_enabled = TRUE ⇒ bytes > 0 not theorem

If another transition decremented bytes, it would not satisfy this invariant.