UML-B State Machine Diagrams

May, 2010

Colin Snook  University of Southampton
State Machines

State machines provide a way to model behaviour (transitions) Constrained by some data (source state) The transition’s behaviour is to change the data (to target state)

Transition $t$ can only fire when the state is $S1$ when $t$ fires it changes the state to $S2$

*How could we represent this in Event-B?*
State Machines to Events

EVENTS

\[ t \triangleq \text{WHEN } \langle \text{in S1} \rangle \text{ THEN } \langle \text{becomes S2} \rangle \text{ END } \]

where, \( \langle \text{in S1} \rangle \) and \( \langle \text{becomes S2} \rangle \) depend on the data that represents state
State machine as a type

We could treat the whole statemachine as an enumerated type
the current state is given by a variable
(called \textit{state\_function} translation in UML-B)

\textbf{VARIABLES}

\begin{itemize}
  \item \texttt{sm : sm\_STATES}
\end{itemize}

\textbf{SETS}

\begin{itemize}
  \item \texttt{sm\_STATES = \{S1,S2\}}
\end{itemize}

\begin{tikzpicture}
  \node[draw, state] (S1) at (0,0) {S1};
  \node[draw, state] (S2) at (2,0) {S2};
  \draw (S1) -- (S2);
  \node at (1,0) {$t$};
\end{tikzpicture}

\textbf{EVENTS}

\begin{itemize}
  \item \texttt{t \triangleq WHEN <in S1> THEN <becomes S2> END}
\end{itemize}

\textit{what are \texttt{<in S1>} and \texttt{<becomes S2>} in this case?}
State machine as a type

VARIABLES
\[ \text{sm} : \text{sm\_STATES} \]

SETS
\[ \text{sm\_STATES} = \{S1,S2\} \]

EVENTS
\[ t \equiv \text{WHEN sm} = S1 \text{ THEN sm} := S2 \text{ END} \]
State machine collection of variables

Or we could treat each state as a separate variable
(called state_sets translation in UML-B)

VARIABLES
  S1 : BOOL
  S2 : BOOL
where, one of S1, S2 is TRUE at any moment

EVENTS
  t ≜ WHEN <in S1> THEN <becomes S2> END

what are <in S1> and <becomes S2> in this case?
State machine collection of variables

VARIABLES

S1 : BOOL
S2 : BOOL

EVENTS

\[ t \triangleq \begin{align*}
\text{WHEN} & \quad S1 = \text{TRUE} \\
\text{THEN} & \quad S1 := \text{FALSE} \\
& \quad S2 := \text{TRUE}
\end{align*} \]

END
Initial transition

Statemachine as type

INITIALISATION
sm := S1

States as variables

INITIALISATION
S1 := TRUE
S2 := FALSE
State Invariant

Something that must be true whenever the system is in that state.

Translations:

(sm = S2) \implies (v1 = TRUE)

or

(S2 = TRUE) \implies (v1 = TRUE)
Transitions are events.
So you can give them parameters, guards and actions etc.

![Diagram showing transition parameters, guards, and actions]

- **Parameter:** \( b \in \text{BOOL} \)
- **Guard:** \( b = \text{TRUE} \)
- **Action:** \( v1 := b \)
Nested Statemachines

Statemachines can be nested inside states

Entry and Exit transitions must align with parent state using Elaborates property
Example

A factory machine can be switched on and off. When it is on it can then be started and becomes active. When it is active it can run repeatedly until it is stopped.

A separately controlled guard mechanism can be opened and closed.
[The guard is opened automatically when the machine is stopped]

Safety Requirement:
The machine should never be in the active state (where runs can occur) with the guard in the open position.

Model the machine and guard as separate statemachines.
Add an invariant to model the safety requirement.
Determine the transitions guards needed to represent the interlocks between the machine and the guard controller.
Use the Pro-B model checker/Animator to ensure that the safety invariant is never violated
Example – Factory Machine

Example of a state machine diagram for a factory machine.
Example – Factory Machine Safety Guard

```
guard: machine_on ≠ active
```

omitted for model checking demo on next page
Statemachine Animation showing invariant violation
Statemachines in Classes

Statemachines can be added to classes. Effectively, each class instance has a “copy” of the statemachine.
State machines in Classes

State machine as a type (state_function)

VARIABLES

\( \text{sm} \in C \rightarrow \text{sm\_STATES} \)

SETS

\( \text{sm\_STATES} = \{S1, S2\} \)

EVENTS

\[
\begin{align*}
\text{t (self)} & \triangleq \text{WHERE} \\
& \text{sm}(\text{self}) = S1 \\
& \text{THEN} \\
& \text{sm}(\text{self}) := S2 \\
& \text{END}
\end{align*}
\]
State machines in Classes

States as variables (state_sets)

VARIABLES

\[ S_1 \subseteq \mathcal{P}(C) \]
\[ S_2 \subseteq \mathcal{P}(C) \]

where \( S_1 \) and \( S_2 \) (and...) are disjoint

EVENTS

\[ t(\text{self}) \triangleq \begin{align*}
\text{WHERE} & \quad \text{self} \subseteq S_1 \\
\text{THEN} & \quad S_1 := S_1 \setminus \{\text{self}\} \\
& \quad S_2 := S_2 \cup \{\text{self}\}
\end{align*} \]

END
Initial transition (state_sets)

For variable instance classes, initial transition is treated as a constructor

```
ini =
STATUS
  ordinary
ANY
  self     // constructed instance of class C
WHERE
  self.type : self ∈ C_SET \ C
THEN
  C_constructor : C = C ∪ {self}
  sm_enterState_S1 : S1 = S1 ∪ {self}
END
```
Final transition (state_sets)

For variable instance classes, final transition is treated as a destructor

```
final ≡
STATUS
  ordinary
ANY
  self       // contextual instance of class C
WHERE
  self.type : self ∈ C
  sm_isin_S2 : self ∈ S2
THEN
  sm_leaveState_S2 : S2 = S2 \ {self}
  CDestructor : C := C \ {self}
END
```
State Invariant  (state_sets)

Something that must be true whenever an instance of the class is in that state.

Translation:
\[
\forall \text{self} \cdot ((\text{self}\in C) \Rightarrow ((\text{self}\in S2) \Rightarrow (\text{v1}(\text{self}) = \text{TRUE})))
\]
Summary

Statemachines for modelling behaviour
  ► nested – statemachines in states
  ► invariants in states
  ► transitions are events (with parameters, guards, actions)

Choice of 2 translations

Can be lifted to classes
  ► initial/final transition as constructor/destructor (variable instance classes)

State-machines can be animated and model checked
  ► (front-end for Pro-B)