Motivation

- Event-B employs the top-down refinement-based approach to system development

- Sometimes there are several refinement alternatives that can adequately implement a certain functional requirement

- These alternatives can have different impact on non-functional system requirements/attributes

- Unfortunately, Event-B does not provide a sufficient support for evaluation of this impact

- Our goal: integrate the quantitative verification of non-functional system properties into Event-B formal development
Event-B Model

Event-B model is a tuple \((C, S, A, \nu, I, \Sigma, E, Init)\) where:

- \(C\) is a set of model constants;
- \(S\) is a set of model sets;
- \(A\) is a set of axioms over \(C\) and \(S\);
- \(\nu\) is a set of system variables;
- \(I\) is a set of invariant properties over \(\nu, C\) and \(S\);
- \(\Sigma\) is a model state space defined by all possible values of the vector \(\nu\);
- \(E \subseteq \text{POW}(\Sigma \times \Sigma)\) is a non-empty set of model events;
- \(Init\) is a predicate defining an non-empty set of model initial states.
Events

Event \( e \in \mathcal{E} : \quad e \triangleq \textbf{when } G_e \textbf{ then } R_e \textbf{ end} \)

- \( G_e \) – event’s \textit{guard} (a conjunction of predicates)
- \( R_e \) – next-state relation called \textit{generalised substitution}
Events

Event $e \in \mathcal{E}: \quad e \equiv \textbf{when } G_e \textbf{ then } R_e \textbf{ end}$

- $G_e$ – event’s guard (a conjunction of predicates)
- $R_e$ – next-state relation called generalised substitution

Equivalent definition:

$$e(\sigma, \sigma') = G_e(\sigma) \land R_e(\sigma, \sigma')$$

$R_e$ is defined by a multiple (possibly nondeterministic) assignment over a vector of system variables
before(e) = \{ \sigma \in \Sigma \mid I(\sigma) \land G_e(\sigma) \}\}

after(e) = \{ \sigma' \in \Sigma \mid I(\sigma') \land (\exists \sigma \in \Sigma \cdot I(\sigma) \land G_e(\sigma) \land R_e(\sigma, \sigma')) \}\}

\mathcal{E}_\sigma = \{ e \in \mathcal{E} \mid \sigma \in \text{before}(e) \}\} is a subset of events enabled in \sigma

The behaviour of any Event-B machine is defined by a transition relation →:

\[
\begin{align*}
\sigma, \sigma' &\in \Sigma \land \sigma' \in \bigcup_{e \in \mathcal{E}_\sigma} \text{after}(e) \\
\sigma &\rightarrow \sigma'
\end{align*}
\]

Event-B model is a transition system with state space \( \Sigma \), transition relation \( \rightarrow \) and a set of initial states defined by \textit{Init}
Qualitative probabilistic choice (assignment): $x \oplus| P(v, x')$

- assigns to $x$ a new value $x'$ with some fixed (but unknown) probability
- was introduced to allow the reasoning about the fairness property in Event-B
- can be placed only instead of an existing nondeterministic assignment
Quantitative probabilistic assignment – a probabilistic assignment that contains a precise probabilistic information about the choice's nature:

\[ x \oplus| \ x_1 \oplus p_1; \ldots; x_m \oplus p_m \]

- \( \sum_{i=1}^{m} p_i = 1 \)
- assigns to \( x \) a new value \( x_i \) with some fixed and known non-zero probability \( p_i \)
- defines a next-state distribution for any \( \sigma \in \text{before}(e) \)
- always refines its corresponding nondeterministic counterpart
Incorporating Probabilities into Event-B (ctd.)

**Quantitative probabilistic assignment** – *continuous-time* form

\[
x \oplus | x_1 @ \lambda_1; \ldots; x_m @ \lambda_m
\]

- \( \lambda_i \in \mathbb{R}^+ \) – constant *rates*
- each \( \lambda_i \) is a parameter of the *exponentially distributed sojourn time* that the system will spend in the state \( \sigma \) before it goes to the new state \( \sigma' \)
- analogue of the discrete-time version with \( p_i = \frac{\lambda_i}{\sum_{j=1}^{m} \lambda_j} \)
- replaces a nondeterministic choice between the possible successor states by the probabilistic choice associated with the (exponential) race condition
refine all the machine events by the probabilistic ones – eliminate *local* nondeterminism

the choice between several simultaneously enabled events is still nondeterministic

$p_e(\sigma, \sigma')$ – the probability to go from $\sigma$ to $\sigma'$ via the event $e$, where $\sigma \in \text{before}(e)$ and $p_e(\sigma, \sigma') > 0$ iff $R_e(\sigma, \sigma')$

The behaviour of any probabilistically augmented Event-B machine is defined by a transition relation $\xrightarrow{\mathcal{P}}$:

$$\sigma, \sigma' \in \Sigma \land \sigma' \in \bigcup_{e \in \mathcal{E}_\sigma} \text{after}(e) \quad \frac{\sigma \xrightarrow{\mathcal{P}} \sigma'}{\sigma, \sigma' \in \Sigma \land \sigma' \in \bigcup_{e \in \mathcal{E}_\sigma} \text{after}(e)}$$

where $\mathcal{P} = \prod_{e \in \mathcal{E}_\sigma} p_e(\sigma, \sigma')$ and $\prod$ denotes a nondeterministic choice.
Event-B as a Probabilistic Transition System (ctd.)

▶ refine all the machine events by the probabilistic ones

▶ \( \lambda_e(\sigma,\sigma') \) – the transition rate from \( \sigma \) to \( \sigma' \) via the event \( e \), where \( \sigma \in \text{before}(e) \) and \( R_e(\sigma,\sigma') \)

The behaviour of any Event-B machine probabilistically augmented with rates is defined by a transition relation \( \Xi \):

\[
\sigma, \sigma' \in \Sigma \land \sigma' \in \bigcup_{e \in E} \text{after}(e) \quad \frac{\Xi}{\sigma \rightarrow \sigma'},
\]

where \( \Xi = \sum_{e \in \mathcal{E}_\sigma} \lambda_e(\sigma,\sigma') \).

All nondeterministic behaviour is eliminated – not always realistic or suitable for modelling.
Event-B as a Probabilistic Transition System (ctd.)

▶ with probabilistic transition relation $\mathcal{P} \rightarrow$, an Event-B machine becomes either a Markov decision process (MDP) or a discrete-time Markov chain (DTMC)

▶ with probabilistic transition relation $\Lambda \rightarrow$, an Event-B machine becomes a continuous-time Markov chain (CTMC)

▶ probabilistic model checking can be used to verify the non-functional properties (quality attributes) of systems modelled in Event-B
Properties to Verify

Reachability properties:

- the algorithm eventually terminates successfully with probability 1
- the (maximum) probability of losing more than $N$ messages by time $T$
- the long-run probability that the queue is more than 75% full

Reward-based properties:

- the expected time taken to reach a certain state
- the expected number of lost messages during a certain time interval
Properties to Verify (ctd.)

Properties are usually specified using temporal logics like:

- **PCTL** (Probabilistic Computation Tree Logic) for discrete-time models
- **CSL** (Continuous Stochastic Logic) for continuous time models
Properties are usually specified using temporal logics like:

- **PCTL** (Probabilistic Computation Tree Logic) for discrete-time models
- **CSL** (Continuous Stochastic Logic) for continuous time models

Sometimes verification of system quality attributes allows us to strengthen the notion of Event-B refinement.
Reliability Refinement

Reliability is the probability that an entity $E$ can perform a required function under given conditions for the time interval $[0, t]$:

$$R(t) = P\{E \text{ not failed over time } [0, t]\}$$

Due to unrecoverable failure occurrences we can distinguish between operational and non-operational system states.

The set of operational states of any Event-B model can be defined by some predicate $J$:

$$\Sigma_{op} \equiv \{\sigma \in \Sigma \mid J(\sigma)\}$$

The reliability function can be formally defined as $R(t) = P\{\square \leq t J\}$.
For two fully probabilistic Event-B models $M_1$ and $M_2$ we say that $M_2$ is a refinement of $M_1$ if and only if

1. $M_2$ is an Event-B refinement of $M_1$ ($M_1 \subseteq M_2$)

2. $\forall t \in \mathbb{N} \cdot P\{\square \leq t \mathcal{J}_1\} \leq P\{\square \leq t \mathcal{J}_2\}$

Remark. If the second condition holds not for all $t$, but for some interval $[0, T]$, $T \in \mathbb{N}_1$, we say that $M_2$ is a partial refinement of $M_1$ for $t \leq T$. 


Reliability Refinement (ctd.)

For two probabilistic-nondeterministic Event-B models $\mathcal{M}_1$ and $\mathcal{M}_2$ we say that $\mathcal{M}_2$ is a refinement of $\mathcal{M}_1$ if and only if

1. $\mathcal{M}_2$ is an Event-B refinement of $\mathcal{M}_1$ ($\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$)

2. $\forall t \in \mathbb{N} \cdot P_{\min}\{\Box \leq_t J_1\} \leq P_{\min}\{\Box \leq_t J_2\}$

$P_{\min}\{\Box \leq_t J\}$ is the “worst case scenario” reliability

Remark. If the second condition holds not for all $t$, but for some interval $[0, T]$, $T \in \mathbb{N}_1$, we say that $\mathcal{M}_2$ is a partial refinement of $\mathcal{M}_1$ for $t \leq T$. 
The PRISM Tool

- PRISM is a probabilistic symbolic model checker

- Modelling of:
  - Discrete-Time Markov Chains (DTMCs)
  - Markov Decision Processes (MDPs)
  - Continuous-Time Markov Chains (CTMCs)
  - Probabilistic Timed Automata (PTAs)

- Verification of:
  - Probabilistic Computation Tree Logic (PCTL/PCTL*)
  - Continuous Stochastic Logic (CSL)
  - Linear Temporal Logic (LTL)

- http://www.prismmodelchecker.org/
PRISM Modelling Language

- State-based language
- Model is constructed as a parallel composition of modules
- `module Module_name`
  
  ```plaintext
  var : Type init ...;

  []  grd1 → p_1 : action_1 + ... + p_n : action_n;

  []  grd2 → q_1 : action'_1 + ... + q_m : action'_m;

  ...

  endmodule
  ```
Event-B to PRISM

Deterministic assignment (dtmc/mdp)

\[\text{when } g \text{ then } x, y := x_1, y_1 \text{ end}\]

\[\text{module } \text{evt}\]
\[x : \text{Type}_1 \text{ init } \ldots ; y : \text{Type}_2 \text{ init } \ldots ;\]
\[g \rightarrow (x' = x_1) \& (y' = y_1);\]
\[\text{endmodule}\]

Probabilistic assignment (all)

\[\text{when } g \text{ then } x \oplus | x_1 @ p_1 ; \ldots ; x_n @ p_n \text{ end}\]

\[\text{module } \text{evt}\]
\[x : \text{Type} \text{ init } \ldots ;\]
\[g \rightarrow p_1 : (x' = x_1) + \cdots + p_n : (x' = x_n);\]
\[\text{endmodule}\]
Parallel probabilistic assignment (dtmc/mdp)

\[
\text{evt} \triangleq \text{when } g \\
\text{then} \\
\begin{align*}
  x & \oplus | x_1 \oplus p_1; \ldots; x_n \oplus p_n \\
  y & \oplus | y_1 \oplus q_1; \ldots; y_m \oplus q_m
\end{align*}
\text{end}
\]

\[
\begin{align*}
\text{module } \text{evt}_x \\
  x : \text{Type}_1 \text{ init } \ldots; \\
  \text{[name]} g \rightarrow p_1 : (x' = x_1) + \cdots + p_n : (x' = x_n); \\
\text{endmodule}
\end{align*}
\]

\[
\begin{align*}
\text{module } \text{evt}_y \\
  y : \text{Type}_2 \text{ init } \ldots; \\
  \text{[name]} g \rightarrow q_1 : (y' = y_1) + \cdots + q_m : (y' = y_m); \\
\text{endmodule}
\end{align*}
\]
Conversion from Event-B to PRISM (ctd.)

Parallel probabilistic assignment (ctmc)

\[\textbf{evt} \supseteq \textbf{when } g \textbf{ then} \]
\[\hspace{1cm} x \oplus | x_1 \circ \lambda_1; \ldots; x_n \circ \lambda_n \]
\[\hspace{1cm} y \oplus | y_1 \circ \mu_1; \ldots; y_m \circ \mu_m \]
\[\textbf{end} \]

\[\textbf{module evt} \]
\[\hspace{1cm} x : Type_1 \textbf{ init } \ldots; \]
\[\hspace{1cm} y : Type_2 \textbf{ init } \ldots; \]
\[\hspace{1cm} [] g \rightarrow \lambda_1 : (x' = x_1) + \cdots + \lambda_n : (x' = x_n); \]
\[\hspace{1cm} [] g \rightarrow \mu_1 : (y' = y_1) + \cdots + \mu_m : (y' = y_m); \]
\[\textbf{endmodule} \]
Example: Single Server Queue

$M/M/1/N$ system
Buffer with a capacity $N$, restricted buffering
Unreliable server (may fail and recover after the failure)

- $\lambda$ – arrival rate
- $\mu$ – service rate
- $\delta$ – server’s failure rate
- $\gamma$ – server’s repair rate
Example: Single Server Queue (ctd.)

Variables $Q, fail, lost$

Invariants $Q \in 0..N \land fail \in BOOL \land lost \in BOOL$

Events

- **empty_queue** $\triangleq$
  
  where $Q = 0$
  
  then $Q := Q + 1$
  
  end

- **in_out_queue** $\triangleq$
  
  where $Q > 0 \land Q < N \land fail = FALSE$
  
  then $Q \in \{Q - 1, Q + 1\}$
  
  end

- **in_queue** $\triangleq$
  
  where $Q > 0 \land Q < N \land fail = TRUE$
  
  then $Q := Q + 1$
  
  end

- **out_queue** $\triangleq$
  
  where $Q = N \land fail = FALSE$
  
  then $Q, lost := Q - 1, FALSE$
  
  end

- **full_queue** $\triangleq$
  
  where $Q = N$
  
  then $lost := TRUE$
  
  end

- **server_fail** $\triangleq$
  
  where $fail = FALSE$
  
  then $fail := TRUE$
  
  end

- **server_repair** $\triangleq$
  
  where $fail = TRUE$
  
  then $fail := FALSE$
  
  end
Example: Single Server Queue (ctd.)

<table>
<thead>
<tr>
<th>Variables</th>
<th>Q, fail, lost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariants</td>
<td>Q ∈ 0..N ∧ fail ∈ BOOL ∧ lost ∈ BOOL</td>
</tr>
<tr>
<td>Events</td>
<td></td>
</tr>
<tr>
<td>empty_queue</td>
<td>⊳where Q = 0 then Q ⊕</td>
</tr>
<tr>
<td>in_out_queue</td>
<td>⊳where Q &gt; 0 ∧ Q &lt; N ∧ fail = FALSE then Q ⊕</td>
</tr>
<tr>
<td>in_queue</td>
<td>⊳where Q &gt; 0 ∧ Q &lt; N ∧ fail = TRUE then Q ⊕</td>
</tr>
<tr>
<td>out_queue</td>
<td>⊳where Q = N ∧ fail = FALSE then Q, lost ⊕</td>
</tr>
<tr>
<td>full_queue</td>
<td>⊳where Q = N then lost ⊕</td>
</tr>
<tr>
<td>server_fail</td>
<td>⊳where fail = FALSE then fail ⊕</td>
</tr>
<tr>
<td>server_repair</td>
<td>⊳where fail = TRUE then fail ⊕</td>
</tr>
</tbody>
</table>
Example: Single Server Queue (ctd.)

const double $\lambda = 0.2$;
const double $\mu = 0.6$;
const double $\delta = 0.05$;
const double $\gamma = 0.1$;
cons int $N = 15$;

module Queue
    $Q : [0..N]$ init 0; $lost : \textbf{bool}$ init false;
    $[] Q = 0 \rightarrow \lambda : (Q' = Q + 1);$ \\
    $[] Q > 0 \& Q < N \& \neg fail \rightarrow \lambda : (Q' = Q + 1) + \mu : (Q' = Q - 1);$ \\
    $[] Q > 0 \& Q < N \& fail \rightarrow \lambda : (Q' = Q + 1);$ \\
    $[] Q = N \& \neg fail \rightarrow \mu : (Q' = Q - 1) \& (lost' = \text{FALSE});$ \\
    $[] Q = N \rightarrow \lambda : (lost' = \text{TRUE});$
endmodule

module Server
    $fail : \textbf{bool}$ init false;
    $[] \neg fail \rightarrow \delta : (fail' = \text{TRUE});$ \\
    $[] fail \rightarrow \gamma : (fail' = \text{FALSE});$
endmodule
Example: Verification

Result of verification: $P_{\geq} [G \leq T \text{ !lost}]$. 
Example: Single Server Queue with a Spare Server

$M/M/1+1/N$ system

Two servers – one active and one (cold) spare

Switcher is absolutely reliable, $\tau$ is a switching rate

Once an active server fails, the switcher activates the spare server allowing the failed one to recover

Once a failed server recovers, it becomes the spare one
module Queue

\[ Q : [0..N] \text{ init } 0; \ lost : \text{bool} \text{ init false}; \]
\[ \begin{align*}
& [] Q = 0 \rightarrow \lambda : (Q' = Q + 1); \\
& [] Q > 0 & Q < N & \text{active} \rightarrow \lambda : (Q' = Q + 1) + \mu : (Q' = Q - 1);
\end{align*} \]

endmodule

module Server

\[ \begin{align*}
& \text{srv} : [1..2] \text{ init } 1; \ \text{fail}_1 : \text{bool} \text{ init false}; \ \text{fail}_2 : \text{bool} \text{ init false}; \\
& [] \text{srv} = 1 & !\text{fail}_1 \rightarrow \delta : (\text{fail}_1' = \text{TRUE}); \\
& [] \text{srv} = 2 & !\text{fail}_2 \rightarrow \delta : (\text{fail}_2' = \text{TRUE}); \\
& [] \text{fail}_1 \rightarrow \gamma : (\text{fail}_1' = \text{FALSE}); \\
& [] \text{fail}_2 \rightarrow \gamma : (\text{fail}_2' = \text{FALSE}); \\
& [] \text{srv} = 1 & \text{fail}_1 & !\text{fail}_2 \rightarrow \tau : (\text{srv}' = 2); \\
& [] \text{srv} = 2 & \text{fail}_2 & !\text{fail}_1 \rightarrow \tau : (\text{srv}' = 1);
\end{align*} \]

endmodule

formula \text{active} = (\text{srv} = 1 & !\text{fail}_1) \ | (\text{srv} = 2 & !\text{fail}_2);
Example: Verification (ctd.)

Result of verification: $P_{=?}[G \leq T !lost]$. 
Example: Verification (ctd.)

Result of verification: \( R\{ \text{"num\_lost"} \} \equiv [C \leq T] \)
$M/M/1+1/N \ vs \ M/M/2/N$?
$M/M/1+1/N$ vs $M/M/2/N$?

Result of verification: $P = ? [G \leq T !lost]$. 
Result of verification: \( P =? [G \leq T \, !\text{lost}] \).