Lecture 6: Outline

- Nondeterminism in Event-B
- Event-B model proof obligations
- Example: an electronic auction system

Nondeterminism in Event-B

- An Event-B machine models the system dynamics by describing its operations (events). Sometimes such a model of system dynamics is called a *state transition system*
- In their turn, the model events contain actions (state assignments), which describe system state changes/transitions
- Here a system state – a vector of the current system variable values
- The state changes can be deterministic (a typical state assignment with one possible outcome) or non-deterministic (one of from several/many state changes is possible)

Nondeterminism in Event-B (cont.)

- A simple example of a nondeterministic assignment:
  $res \ :\in \ \{Success,\ Failure\}$

  After this assignment, *res* can get any value from the set $\{Success,\ Failure\}$

- The most general form of non-deterministic assignment has the following syntax:

  $$x, y \ :\mid \ Condition(x, y, z, x', y')$$

  where x and y are the variables to be changed, z are the variables to be read only, and x' and y' are the new values for x and y

- Example:
  $time\_left \ :\mid \ time\_left' < time\_left$
  any value from the interval $0..time\_left - 1$ can be assigned

Nondeterminism in Event-B (cont.)

- Declaring the event parameters/local variables in ANY clause – a special case of non-determinism
- A general event
  ANY *p* **WHERE** *Cond(p,v)* **THEN** *Actions* **END**
  is "executed" as a sequential composition
  $p :\mid Cond(v, p');\ \ Actions$
- There is no difference between having event parameters that are non-deterministically initialised by such implicit assignment or introducing local event variables with the same effect
- Example: $time\_left\ :\mid time\_left' < time\_left$
  can be rewritten as
  ANY tt **WHERE** $tt \in \mathbb{N} \wedge tt < time\_left$
  THEN $time\_left := tt$ **END**

Nondeterminism in Event-B (cont.)

- Machine events with overlapping guards – another special case of system non-determinism
- Any of such events can be chosen for execution, thus different event actions may be used to change the system state $\Rightarrow$ the overall result is non-deterministic
- Example: Having two events
  WHEN $component \in active$ **THEN** $status :=$ Status1 **END**
  and
  WHEN $component \in active$ **THEN** $status :=$ Status2 **END**

  is equivalent to having one merged event

  WHEN $component \in active$
  THEN $status :\in \{\text{Status1}, \text{Status2}\}$ **END**

Nondeterminism in Event-B (cont.)

- Non-determinism by machine events with overlapping guards can be used to express the situations where we have no knowledge or control over which event will occur first
- If the events operate over disjoint sets of variables (i.e., the updated variables are different), such overlapping events can also be used to model parallel calculations

Event-B proof obligations

- Event-B proof obligations (POs) define what it is to be proved for an Event-B model
- They are automatically generated (and in most cases also proved) by the Rodin platform
- Proof obligations – mathematical statements (theorems) about model correctness, consistency, and well-definedness
- There are several kinds of POs: invariant preservation, feasibility, well-definedness, theorem, etc.

Kinds of proof obligations: invariant preservation

- Proof obligations to ensure that each machine invariant is preserved by any machine event
- Proof obligations has the label INV
- Suppose we have a general form of an event:

  ANY x
  WHERE $G(s, c, v, x)$
  THEN $v :| \, Cond(s, c, v, x, v')$ **END**

  where s and c are sets and constants from the context, v are machine variables, and x are event parameters or local variables.

  Note that any standard assignment $x := Exp(x, y)$ is just equivalent to $x :| \, x' = Exp(x, y)$

Proof obligations: invariant preservation (cont.)

- Then, for such a general event, the invariant preservation POs are generated according to the following rule:

$$
\begin{array}{lll}
 & A(s,c) & \text{Axioms and theorems} \\
 & Inv(s,c,v) & \text{Invariants and theorems} \\
 & G(s,c,v,x) & \text{Guards of the event} \\
 & Cond(s,c,v,x,v') & \text{The result condition of a non-det. assignment} \\
\vdash & & \\
 & Inv_i(s,c,v') & \text{Specific invariant (in a post-state)}
\end{array}
$$

- Simplified form (without explicitly mentioning the sets and constants):
  $\vdash \;\; G(v,x) \;\wedge\; Inv(v) \;\wedge\; Cond(v,x,v') \;\Rightarrow\; Inv_i(v')$

  For any possible state change, assuming all the invariants are true before the event and all the event guards are satisfied, the invariant in question remains true after the event is executed

Proof obligations: invariant preservation (cont.)

MACHINE Hotel
VARIABLES
  $\ldots, reserved, occupied$
INVARIANT
  $reserved \in ROOM \nrightarrow CLIENT$
  $occupied \in ROOM \nrightarrow CLIENT$
  $occupied \subseteq reserved$
$\ldots$
EVENTS
  $new\_reservation \;=\;$ **ANY** $r,\; c$
    WHERE $r \notin dom(reserved) \;\wedge\; c \in CLIENT$
    THEN
      $reserved(r) \;:=\; c$
    END

Proof obligations: invariant preservation (cont.)

- For the last specific invariant $occupied \subseteq reserved$, the generated proof obligation is

$$
\begin{array}{l}
\qquad reserved \in ROOM \nrightarrow CLIENT \\
\qquad occupied \in ROOM \nrightarrow CLIENT \\
\qquad occupied \subseteq reserved \\
\qquad r \notin dom(reserved) \\
\qquad c \in CLIENT \\
\qquad reserved' = reserved \mathbin{\lhd\!\!\!-} \{r \mapsto c\} \\
\vdash \\
\qquad occupied \subseteq reserved'
\end{array}
$$

- Or, in the simplified form:

$$
\begin{array}{l}
occupied \subseteq reserved \wedge r \notin dom(reserved) \wedge c \in CLIENT \Rightarrow \\
\qquad occupied \subseteq reserved \mathbin{\lhd\!\!\!-} \{r \mapsto c\}
\end{array}
$$

Proof obligations: feasibility

- The purpose of feasibility POs is to ensure that a non-deterministic action is possible
- For a general event, the feasibility POs (labeled FIS) are generated according to the following rule:

	$A(s, c)$	Axioms and theorems
	$Inv(s, c, v)$	Invariants and theorems
	$G(s, c, v, x)$	Guards of the event
$\vdash$		
	$\exists v'.Cond(s, c, v, x, v')$	Nondeterministic action is possible

- Example: the event action

$time\_left\ :|\ time\_left' < time\_left$

(for $time\_left \in \mathbb{N}$) is only provably feasible if $time\_left > 0$

Proof obligations: well-definedness

- The purpose of well-definedness POs (labeled WD) is to ensure that all expressions are well-defined
- Examples of possibly ill-defined expressions:
  applying a partial function to the argument that is not in the function domain (a special case – division by zero),
  calculating the set cardinality (the number of elements) for a possibly infinite set, etc.
- In each separate case, the appropriate well-definedness condition is generated and asked to be proven
- A simple example: having an event guard
  $c = reserved(r)$
  would automatically lead to a generated WD proof obligation:
  $r \in dom(reserved)$

Proof obligations: theorems

- The theorem POs (labeled THM) are needed to ensure that the proposed context or machine theorems are provable
- You can propose a theorem by marking as such a specific axiom, invariant or guard that logically follow from their counterparts
- They seem to be redundant statements, however, once proven, they are automatically employed by the Rodin provers to prove other POs
- Often used as simpler intermediate statements (lemmas) that help in more complicated proofs

Other proof obligations

- If we refine/elaborate our models, we need to show that these refined models are consistent, i.e., they do not contradict the previously developed ones
- Rodin automatically generates additional proof obligations to ensure that
- Kinds of refinement proof obligations: simulation, guard strengthening, variants
- If refinement POs are successfully proven, all the verification results for the more abstract models are safely inherited

Simple example of a electronic auction system: requirements

1. The system (master component) arranges and manages an electronic auction for selling items
2. There is a number of registered sellers and buyers that are the users of an electronic action
3. Any seller can offer items to be sold
4. Once an item to be sold is submitted, the auction for this item starts. The system can manage many items at once
5. During the auction, any buyer can make a bid for the item
6. A bid is only accepted if it exceeds the previous maximal bid for the same item

Electronic auction system: requirements (cont.)

7. The initial (minimal) item bid is given by a seller. By default, it is 0
8. There is the (pre-defined beforehand) maximal time for the auction to be completed
9. Once the maximal time runs out, the auction for this item is over and the maximal bid is declared a winner
10. The buyer that made the highest bid must pay for the item
11. Once the payment is made to the seller, the buyer receives the item
12. The system stores the information about what the current items are managed by the auction, which is the highest bid for each item, how much current time is left for each item, what the current items that were just paid for.

Electronic auction system: requirements (cont.)

13. All the information related to a specific item is erased from the system after the item is is sold and the buyer receives the item
14. Additionally, the system stores the buyer and seller logs about which items have been sold and bought in the auction
15. The log information is accumulated (not erased)
16. The seller log is updated after the item payment is made
17. The buyer log is updated after the seller log
18. The information in both logs should be consistent for the items that are not currently being auctioned

Electronic auction system: context

CONTEXT

Auction_ctx

SETS

ITEM

SELLER

BUYER

CONSTANTS

Max_time

AXIOMS

$Max\_time \in \mathbb{N}1$

END

Electronic auction system: machine

MACHINE
Auction_mch
SEES
Auction_ctx
VARIABLES
in_auction, time_left, item_seller, highest_bids, bids_buyer, paid_items, seller_log, buyer_log
INVARIANT

$in\_auction \in \mathbb{P}(ITEM)$
$time\_left \in ITEM \pushfunc \mathbb{N}$
$dom(time\_left) = in\_auction$
$item\_seller \in ITEM \pushfunc SELLER$
$dom(item\_seller) = in\_auction$
$highest\_bids \in ITEM \pushfunc \mathbb{N}1$
$dom(highest\_bids) \subseteq in\_auction$

...

Electronic auction system: machine (cont.)

$$
\begin{array}{l}
bids\_buyer \in ITEM \nrightarrow BUYER \\
dom(bids\_buyer) = dom(highest\_bids) \\
paid\_items \in \mathbb{P}(ITEM) \\
paid\_items \subseteq in\_auction \\
seller\_log \in ITEM \nrightarrow SELLER \\
buyer\_log \in ITEM \nrightarrow BUYER \\
dom(buyer\_log) \subseteq dom(seller\_log) \\
dom(seller\_log) \cap in\_auction \subseteq paid\_items \\
dom(buyer\_log) \cap in\_auction \subseteq paid\_items \\
\forall i.i \notin in\_auction \;\Rightarrow\; (i \in dom(buyer\_log) \Leftrightarrow i \in dom(seller\_log))
\end{array}
$$

...

Electronic auction system: machine (cont.)

INITIALISATION

$in\_auction, time\_left, item\_seller := \varnothing, \varnothing, \varnothing$
$highest\_bids, bids\_buyer, paid\_items := \varnothing, \varnothing, \varnothing$
$seller\_log, buyer\_log := \varnothing, \varnothing$

EVENTS

new_item = **ANY** i, s
WHERE $i \in ITEM \wedge i \notin in\_auction$
$i \notin dom(buyer\_log) \wedge s \in SELLER$
THEN
$in\_auction := in\_auction \cup \{i\}$
$time\_left(i) := Max\_time$
$item\_seller(i) := s$
END

...

Electronic auction system: machine (cont.)

$$
\begin{array}{l}
\mathit{new\_bid} \;=\; \textbf{ANY}\ i,\ n,\ b \\
\quad \textbf{WHERE}\ i \in \mathit{dom}(\mathit{highest\_bids}) \;\wedge\; n \in \mathbb{N}1 \\
\qquad\qquad n > \mathit{highest\_bids}(i) \;\wedge\; b \in \mathit{BUYER} \\
\quad \textbf{THEN} \\
\qquad \mathit{highest\_bids}(i) \;:=\; n \\
\qquad \mathit{bids\_buyers}(i) \;:=\; b \\
\quad \textbf{END} \\
\mathit{new\_bid\_first\_time} \;=\; \textbf{ANY}\ i,\ n,\ b \\
\quad \textbf{WHERE}\ i \notin \mathit{dom}(\mathit{highest\_bids}) \;\wedge\; i \in \mathit{in\_auction} \\
\qquad\qquad n \in \mathbb{N}1 \;\wedge\; b \in \mathit{BUYER} \\
\quad \textbf{THEN} \\
\qquad \mathit{highest\_bids}(i) \;:=\; n \\
\qquad \mathit{bids\_buyers}(i) \;:=\; b \\
\quad \textbf{END}
\end{array}
$$

...

Electronic auction system: machine (cont.)

$$
\begin{array}{l}
\ldots \\
\quad payment \ = \ \textbf{ANY} \ i \\
\quad\quad \textbf{WHERE} \ i \in in\_auction \ \wedge \ i \in dom(highest\_bids) \\
\quad\quad\quad\quad\quad\quad time\_left(i) = 0 \\
\quad\quad \textbf{THEN} \\
\quad\quad\quad paid\_items \ := \ paid\_items \cup \{i\} \\
\quad\quad \textbf{END} \\
\quad sell\_confirmed \ = \ \textbf{ANY} \ i, \ s \\
\quad\quad \textbf{WHERE} \ i \in paid\_items \ \wedge \ s \in SELLER \\
\quad\quad \textbf{THEN} \ seller\_log(i) := s \ \textbf{END} \\
\quad buy\_confirmed \ = \ \textbf{ANY} \ i, \ b \\
\quad\quad \textbf{WHERE} \ i \in paid\_items \ \wedge \ b \in BUYER \\
\quad\quad\quad\quad\quad\quad i \in dom(seller\_log) \\
\quad\quad \textbf{THEN} \ buyer\_log(i) := b \ \textbf{END}
\end{array}
$$

Electronic auction system: machine (cont.)

$$
\begin{array}{l}
\ldots \\
\mathit{item\_done} \;=\; \textbf{ANY } i \\
\quad \textbf{WHERE } i \in \mathit{in\_auction} \;\wedge\; i \in \mathit{dom(buyer\_log)} \\
\qquad\qquad\quad i \in \mathit{dom(seller\_log)} \\
\quad \textbf{THEN} \\
\qquad \mathit{in\_auction} \;:=\; \mathit{in\_auction} \setminus \{i\} \\
\qquad \mathit{time\_left}, \mathit{item\_seller} \;:=\; \{i\} \lhd \mathit{time\_left},\; \{i\} \lhd \mathit{item\_seller} \\
\qquad \mathit{bids\_buyer}, \mathit{paid\_items} \;:=\; \{i\} \lhd \mathit{bids\_buyer},\; \{i\} \lhd \mathit{paid\_items} \\
\qquad \mathit{highest\_bids} \;:=\; \{i\} \lhd \mathit{highest\_bids} \\
\quad \textbf{END} \\
\mathit{time\_progress} \;=\; \textbf{ANY } i,\; \mathit{new\_time} \\
\quad \textbf{WHERE } i \in \mathit{in\_auction} \;\wedge\; i \in \mathit{time\_left(i)} > 0 \\
\qquad\qquad\quad \mathit{new\_time} \in \mathbb{N} \;\wedge\; \mathit{new\_time} < \mathit{time\_left(i)} \\
\quad \textbf{THEN} \\
\qquad \mathit{time\_left} \;:=\; \mathit{new\_time} \\
\textbf{END}
\end{array}
$$

Time modelling

- Time is modelled by keeping track how much time is left for a particular item
- Time progress is specified by non-deterministic decrease of the stored time value (using a local variable *new_time*)
- Time decrease can be alternatively modelled as a non-deterministic assignment:

$$time\_left \;:|\; time\_left' < time\_left$$

- Even though time progress is universal, we can model time progress separately for each auctioned item (because sellings of different items are independent)
- Such separation (as well as non-determinism) allows us avoid bigger complexity here