Lecture 8: Outline

- Refinement in Event-B revisited
- Refinement proof obligations
- Example: a file transfer protocol
- Homework: a flight ticket system

Refinement in Event-B (reminder)

- A way to gradually develop formal models, elaborating on missing implementation details
- Allows to model the system at different abstraction levels, handle its complexity, and structure its requirements
- Consistency of model refinements is supported by the Rodin platform
- All the verification results (e.g., proved invariants) are inherited by the refined models
- Model refinement by refinement also facilitates automation by splitting/reducing overall complexity of model proofs

The notion of model refinement (cont.)

- Essential property: transitivity. Allows us to build a refinement chain of gradual development (unfolding) of the system;
- Mathematically:

$$\frac{M_1 \sqsubseteq M_2 \sqsubseteq .... \sqsubseteq M_n}{M_1 \sqsubseteq M_n}$$

- All proven properties of more abstract models are preserved by refinement;
- Many formalisations based on the idea of model refinement, e.g., Refinement Calculus, VDM, Perfect Developer ...

Refinement of a context component

- For a context component, it is called *extension*;
- The keyword **EXTENDS** associates with an extended abstract context. All the definitions of the given context are transitively inherited
- Context extension allows
  - introducing new data structures (sets and constants), as well as
  - adding more constraints (axioms) for already defined ones.

Refinement of a machine component

- The keyword **REFINES** associates with a given abstract machine. All the invariants of the given machine are transitively inherited
- Moreover, the refined versions of inherited events have the clause **refines** <*old_event*>. All the remaining events are considered as new
- For a machine component, there are several possible kinds of refinement:
  - simple extension of an abstract model by new variables and events (*superposition refinement*);
  - constraining the behaviour of an abstract model (*refinement by reducing model non-determinism*);
  - replacing some abstract variables by their concrete counterparts (*data refinement*).

Superposition refinement

- Adding new variables and events;
- Reading and updating new variables in old event guards and actions;
- Interrelating new and old variables by new invariants;
- **Restriction**: the old variables cannot be updated in new events!
- Example of superposition refinement: in the sluice system (Lecture 3), the first refinement step introduced the controller phases (as a new variable). The old events became a part of the CONT phase, while other phases were modelled by new events

Refinement of non-determinism

- Focuses on the old (abstract) model events:
  - Strengthening the guards;
  - Providing several versions of the same event;
  - Refining non-deterministic actions (assignments).

$evt\ =$
WHERE g **THEN**
$\quad detected :\in BOOL$
END

$evt1$ **refines** $evt\ =$
WHERE $g \wedge g'$ **THEN**
$\quad detected := TRUE$
END

$evt2$ **refines** $evt\ =$
WHERE $g \wedge g''$ **THEN**
$\quad detected := FALSE$
END

Data refinement

- Replacing some old variables by their concrete counterparts
- A part of concrete invariant, *gluing invariant*, describes the logical relationships between the old and new variables
- The gluing invariant is used in all proofs to show the correctness of such a replacement.

$$
\begin{aligned}
&(comm\_failure = TRUE) \Leftrightarrow \\
&\quad (msg\_sent = \mathrm{FALSE} \vee (msg\_sent = \mathrm{TRUE} \wedge msg\_lost = \mathrm{TRUE}))
\end{aligned}
$$

Data refinement: example

- In the sluice system abstract models (Lecture 3), system failures are modelled by the abstract boolean variable *failure*
- Once the information about door sensors is introduced, we can directly model detection of sensor failures
- The abstract variable *failure* then can be replaced (data refined) by the concrete variables *door1_sen_failure, door2_sen_failure, and other_failures*
- This is reflected by the added gluing invariant:

$$failure = TRUE \Leftrightarrow (door1\_sen\_failure = \mathrm{TRUE} \vee door2\_sen\_failure = \mathrm{TRUE} \vee other\_failures = \mathrm{TRUE})$$

Refinement proof obligations

- In addition to standard POs (like invariant preservation), a refined model should satisfy the following properties
  - guards of the old events are strengthened (or remain the same);
  - actions of the old events *simulate* those of the abstract ones – each refined model transition (execution step) is allowed by the abstract model;
  - the new events do not take over forever (do not get into infinite loop).
- In all POs, the gluing invariant is used to relate the old and new model states.

Kinds of proof obligations: guard strengthening for old events

- Proof obligations to ensure that refined events are called only in the situations that were abstractly modelled by the corresponding abstract counterparts
- Proof obligations has the label GRD
- Suppose we have a general form of an abstract event:
  ANY x
  WHERE $G_a(s, c, v, x)$
  THEN $v :| \, Cond_a(s, c, v, x, v')$ **END**

  while the concrete its version is
  ANY y
  WHERE $G_c(s, c, w, y)$
  THEN $v :| \, Cond_c(s, c, w, y, w')$ **END**

Proof obligations: guard strengthening for old events (cont.)

- Then, for such events, the guard strengthening POs are generated according to the following rule:

$$
\begin{array}{lll}
 & A(s,c) & \text{Axioms and theorems} \\
 & Inv_a(s,c,v) & \text{Abstract invariant} \\
 & Inv_c(s,c,v,w) & \text{Concrete invariant} \\
 & G_c(s,c,w,y) & \text{Concrete guards of the event} \\
\vdash & & \\
 & G_a(s,c,v,x) & \text{Abstract guards of the event}
\end{array}
$$

- Note that concrete invariant can refer to both abstract (v) and concrete (w) model variables

Kinds of proof obligations: simulation POs

- Proof obligations to ensure that the actions of a refined event are not contradictory with those of the corresponding abstract event
- Proof obligations has the label SIM
- Note that both GRD and SIM proof obligations concern with only the "old" events, defined in an abstract model and then refined in a refinement step

Proof obligations: simulation (cont.)

- Then, for such events, the simulation POs are generated according to the following rule:

$A(s, c)$	Axioms and theorems
$Inv_a(s, c, v)$	Abstract invariant
$Inv_c(s, c, v, w)$	Concrete invariant
$G_c(s, c, w, y)$	Concrete guards of the event
$Cond_c(s, c, w, y, w')$	The concrete action condition
$\vdash$	
$Cond_a(s, c, v, x, v')$	The abstract action condition

Termination (convergence) of iterative events

- Sometimes we need to ensure that some iterative model events cannot take over forever (no infinite loops). Particular case: new events in a refined model
- This can be done by providing an expression (bounded from below with some minimal value) that is decreasing as a result of such event execution. Such an expression is called "variant"
- Decreasing a variant means that we definitely progressing towards "loop termination", since by definition such expression cannot be decreased indefinitely. Typically a variant is a natural number expression
- The corresponding generated POs to ensure that are labelled VAR

Termination (convergence) of iterative events (cont.)

- For such events, the termination POs are generated according to the following rule:

$$
\begin{array}{lll}
 & A(s,c) & \text{Axioms and theorems} \\
 & Inv(s,c,v) & \text{Model invariant} \\
 & G(s,c,v,x) & \text{Guards of the event} \\
 & Cond(s,c,v,x,v') & \text{Nondeterministic action condition} \\
\vdash & & \\
 & n(s,c,v') < n(s,c,v) & \text{Decreasing of the variant}
\end{array}
$$

where n is a model variant

File transfer protocol: requirements

The example taken from the J.-R. Abrial's book (with slight modifications).

1. The protocol (a version of two phase handshake protocol) ensures the copying of a file from one site to another one
2. The file consists of a sequence of items
3. The file is sent item by item between the two sites
4. The sites exchange sending and receiving/acknowledgement operations through data channels
5. The exchanged information should be kept at minimum (only necessary)
6. Special procedures (like parity check) can be used to ensure data consistency

File transfer protocol: abstract context

CONTEXT
  C0
SETS
  DATA
CONSTANTS
  fsize, *f*
AXIOMS
  $fsize \in \mathbb{N}1$
  $f \in 1..fsize \to DATA$
END

File transfer protocol: notes

- The file to be sent is constant during execution of the file transfer
- Therefore, we can model it as a constant function in the abstract model context
- Another known pre-defined constant is the file size – fsize

File transfer protocol: abstract machine

MACHINE
M0
SEES
C0
VARIABLES
g
INVARIANT
$g \in 1..fsize \mapsto DATA$
INITIALISATION
$g \ := \ \varnothing$
...

File transfer protocol: abstract machine (cont.)

EVENTS

receive $=$

status *anticipated*

WHEN $g \neq f$

THEN

$g \ :\in \ 1..fsize \mathrel{+\!\!\!\rightarrow} DATA$

END

final $=$

WHEN $g = f$

THEN

skip

END

File transfer protocol: notes

- The received file (g) store some part of the original file (f). The variable is nondeterministically updated until the whole file is received
- We need to ensure that the receive event will eventually terminate. However, it is not enough information in the model to formulate a variant and prove termination (convergence)
- Compromise solution: the event gets the status "*anticipated*". It is a promise that, at some later refinement step, its convergence will be proved. In other words, its a postponed property proof
- *skip* in the event *final* stands for the absence of any actions

File transfer protocol: the first refinement

- We introduce a counter to receive the file piece by piece and refine the *receive* event
- Using the counter, we make it precise which portion of file is currently received and stored in g
- The counter also allows to formulate a model variant and turn the *receive* event into convergent
- The event guards also are rewritten in terms of the counter and the file size

File transfer protocol: first refinement

MACHINE
  M1
REFINES *M0*
SEES
  C0
VARIABLES
  g, r
INVARIANT
  $r \in 1..fsize + 1$
  $g = (1..r - 1) \lhd f$
VARIANT
  $fsize + 1 - r$
INITIALISATION
  $g \ := \ \varnothing$
  $r \ := \ 1$
...

File transfer protocol: first refinement (cont.)

EVENTS

receive $=$

status *convergent*

refines *receive*

WHEN $r \leq fsize$

THEN

$g(r) := f(r)$

$r := r + 1$

END

final $=$

refines *final*

WHEN $r = fsize + 1$

THEN

skip

END

File transfer protocol: second refinement

- Drawback: the receiver (in the *receive* event) has a direct access to the file f, which is situated at the sender site
- We need more distributed version of our protocol
- A model refinement introduces another event, *send*, changing the execution trace from

  init $\rightarrow$ receive $\rightarrow$ ... $\rightarrow$ receive $\rightarrow$ final

  to

  init $\rightarrow$ send $\rightarrow$ receive $\rightarrow$ ... $\rightarrow$ send $\rightarrow$ receive $\rightarrow$ final

File transfer protocol: second refinement

MACHINE
M2
REFINES *M1*
SEES
C0
VARIABLES
$g,\ r,\ s,\ d$
INVARIANT
$s \in 1..fsize + 1$
$d \in DATA$
$s = r + 1 \ \Rightarrow d = f(r)$
$s \in r..r + 1$
VARIANT
$fsize + 1 - s$
...

File transfer protocol: notes

- A new counter, s, is introduced for the sender site
- Only the sender counter s and one data item are sent over the channel
- The counter value on the receiver site, r, is sent back as an acknowledgement
- We immediately prove that the new *send* event is convergent with the provided variant

File transfer protocol: second refinement (cont.)

INITIALISATION

$g \;:=\; \varnothing$

$r, s \;:=\; 1, 1$

$d \;:\in\; DATA$

EVENTS

$receive \;=$

refines *receive*

WHEN $s = r + 1$

THEN

$g(r) \;:=\; d$

$r \;:=\; r + 1$

END

...

File transfer protocol: second refinement (cont.)

$final \;=$
refines $final$
WHEN $r = fsize + 1$
THEN
$skip$
END

$send \;=$
status $convergent$
WHEN $s = r$
$s \neq fsize + 1$
THEN
$d \;:=\; f(s)$
$s \;:=\; s + 1$
END

File transfer protocol: third refinement

- It is not necessary to transmit the entire counters s and r on the data and acknowledgement channels
- The reasons: the counters only tested for equality, the only changes are increments by 1, the difference between s and r is at most 1
- Thus, the tests can be performed only on *parities* (last bits) of these counters

File transfer protocol: extended context

CONTEXT

C1

EXTENDS C0

CONSTANTS

parity

AXIOMS

$parity \in \mathbb{N}1 \rightarrow \{0, 1\}$

$parity(1) = 1$

$\forall x.\ x \in \mathbb{N}1 \ \Rightarrow\ parity(x + 1) = 1 - parity(x)$

END

File transfer protocol: third refinement

MACHINE
M3
REFINES *M2*
SEES
C1
VARIABLES
g, r, s, d, p, q
INVARIANT
$p \in \{0, 1\}$
$q \in \{0, 1\}$
$p = parity(s)$
$q = parity(r)$
$p = q \Rightarrow s = r$
$p \neq q \Rightarrow s = r + 1$
...

File transfer protocol: third refinement (cont.)

INITIALISATION

$g \ := \ \varnothing$

$r, s \ := \ 1, 1$

$d \ :\in \ DATA$

$p, q \ := \ 1, 1$

EVENTS

$receive \ =$

refines *receive*

WHEN $p \neq q$

THEN

$g(r) \ := \ d$

$r \ := \ r + 1$

$q \ := \ 1 - q$

END

...

File transfer protocol: third refinement (cont.)

$$
\begin{array}{l}
\mathit{final} \;\; = \\
\quad \textbf{refines } \mathit{final} \\
\quad \textbf{WHEN } r = \mathit{fsize} + 1 \\
\quad \textbf{THEN} \\
\qquad \mathit{skip} \\
\quad \textbf{END} \\
\mathit{send} \;\; = \\
\quad \textbf{refines } \mathit{send} \\
\quad \textbf{status } \mathit{convergent} \\
\quad \textbf{WHEN } p = q \\
\qquad\qquad\;\; s \neq \mathit{fsize} + 1 \\
\quad \textbf{THEN} \\
\qquad d \;:=\; f(s) \\
\qquad s \;:=\; s + 1 \\
\qquad p \;:=\; 1 - p \\
\quad \textbf{END}
\end{array}
$$

File transfer protocol: notes

- The counter parity values are stored in the new variables p and q
- The checks in event guards are rewritten in terms of parities
- The parity and counter values are related in additional invariants (needed to actually prove a refinement step, especially GRD POs)

Homework: flight ticket system (requirements)

1. The purpose of the system is facilitate booking tickets to available flights between cities
2. Each flight is associated with a pair of cities: the departure city and the destination city
3. The departure and destination cities for the same flight should be different
4. The same flight might occur on different dates (days)
5. The information about specific dates when the flight occurs is constant and is available in the system
6. There is a maximal number of plane seats, which can be different for each flight

Flight ticket system: requirements (cont.)

7. The system should keep track of a number of available tickets for a particular flight and a date
8. The system should provide an operation for booking (if possible) for a number of tickets for a particular flight on a particular date
9. The system should have an operation for cancelling of a number of tickets for a particular flight on a particular date
10. The system also keeps the information about the prices for a particular flight on specific dates
11. An additional booking operation should allow to book, if possible, two connecting flights (i.e., with the same intermediate city) for a pair of given cities, a specific date, and the maximal price of two flights
12. If there several pairs of possible connecting flights satisfying the criteria, the actual connecting flights are chosen non-deterministically