Refinement Patterns

A. Iliasov

Newcastle University, Newcastle Upon Tyne, England

Plan

- Motivation and Proposal
- Refinement Patterns
- Pattern Correctness
- Example - Triple Modular Redundancy
- TMR Correctness
- Patterns Plugin

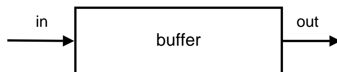
Motivation (1)

- Use of formal methods requires specific expertise
- Full-scale formal modelling is expensive
- Formal development is not a linear process - mistakes are made and refinements steps have to be discarded
- Design mistakes are still a possibility - a perfectly valid model may have serious design flaws
- Interactive proofs are time-consuming

Motivation (2)

- There is a large number of well-studied design patterns
- In a large-scale development there may be important reoccuring patterns
- Very similar refinement steps are often repeated during a development
- Modelling expertise cannot be easily transferred among designers
- Proofs made for a model cannot be reused in another model

Motivation (3) - Communication Buffer



- May have slightly different versions - types, size restrictions
- Basic features - reader blocks when there are no messages, write blocks when buffer is full
- Additional features - lost, duplicates, noise
- How long does it take to specify it in B?
- How many times this has been done?

Proposal (1)

- Language for describing refinement steps as model transformations: input is an abstract model, output is a concrete model
- Method and tool for applying transformations to Event-B specification
- Method to prove for a given transformation that produced models is always a correct refinement of the respective abstract model
- Mechanism to capture refinement transformation given an abstract machine and its refinement
- An internet library of reusable model transformation

Describing Refinement

Definition

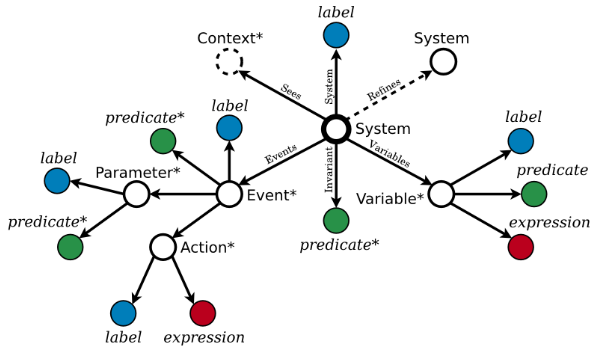
Refinement. The fact that S_1 is refined by S_2 is written as $S_1 \sqsubseteq S_2$, relation $\sqsubseteq$ is reflexive (a specification is a valid refinement of itself), transitive and antisymmetric $(S_1 \sqsubseteq S_2 \land S_1 \sqsubseteq S_2 \Rightarrow S_1 = S_2)$.

Describing Refinement

Definition

Refinement pattern (or pattern). Let S be the universe of specifications. Function $p : S_1 \longrightarrow S_2$ where $S_1 \subseteq S$ and $S_2 \subseteq S$ such that $\forall s \cdot (s \in S_1 \Rightarrow s \sqsubseteq p(s))$ is called *refinement pattern*.

Event-B



Notation - Pattern

pattern declarataion
pattern [s]
  [parameters $u_1, \dots u_l$]
  [requirements r]
  [$r_1 \dots r_n$]

- Pattern parameters are elements of an input specification
- Requirements express element types and restrictions

Notation - Variable declaration

variable [v] [**for** p]
[**label** `varname`]
invariant T
[**action** I]

- A variable is defined by supplying its typing invariant
- System variables also require and initialisation action
- Optional label is variable name variable in output model
- **for** $\cdot$ defines rule aplication scope

Notation - New guards and invariants

invariant $[i]$ [**for** p]
  expression I
or
invariant I [**for** p]
	guard $[g]$ [**for** p]
	  expression G
	or
	guard G [**for** p]

Notation - Action declaration

action [a] [**for** p]
  label l
  style s
  expression e
or
action l s e [**for** p]

- Action is defined by supplied a variable to be updated, a style and an expression
- Style is $:=$, $:\in$ or $:\mid$

Notation - Event declaration

event $[e]$
  [**label** `eventname`]
  [**refines** `abstractevent`]
  $[v_1 \dots v_n]$
  $[g_1 \dots g_k]$
  $[a_1 \dots a_m]$

Notation - Expressions

- Event-B expressions are objects
- . (dot) operator refernces object properties

$$\mathit{var}.\mathrm{type}$$

$$a = b \quad a \neq b \quad a \in b \quad \texttt{left}\ e\ \texttt{right}$$

$$a + b \quad a - b \quad a * b \quad a/b$$

Simple Example

pattern *addinc*
parameters e
req_*typing* $e \in Events$
variable v
  invariant $v \in \mathbb{Z}$
  action $v :\in 0 \ldots 5$
action $v := v + 1$ **for** e
guard $v < 10$ **for** e

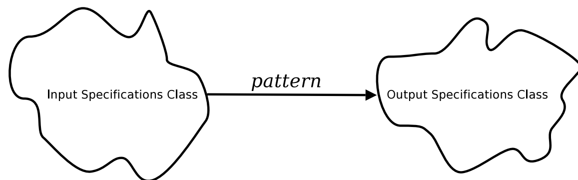
Simple Example - Abstract Specification

machine *ex*
variables *s*
invariant $s \in \mathbb{N}$
initialisation $s := 0$
events
  $a =$ **begin** $s := s + 1$ **end**;
  $b =$ **begin** $s := s + 2$ **end**
end

Simple Example - Result of the Pattern Application

refinement *ex_addinc*
refines a
variables s, q
invariant $s \in \mathbb{N}, q \in \mathbb{Z}$
initialisation $s := 0 \parallel$
$q :\in 0 \ldots 5$
events
$a =$ **when** $q < 10$ **then**
$s := s + 1 \parallel$
$q := q + 1$
end;
$b =$ **begin** $s := s + 2$ **end**
end

Pattern Correctness



- A pattern is *correct* if for any input specification its produces a correct refinement of the specification. A correct refinement is understood as well-formed specification that is a refinement of its abstract specification.
- Event-B well-formedness and refinement conditions as the basis for formulating pattern correctness conditions
- A pattern is proved to be correct for the whole class of matching input specifications.

Proof Obligations

PAT_REQ_FIS	$\exists p \cdot RQ(p)$
PAT_FIS_INI	$\exists w' \cdot S(w, w')$
PAT_INV_INI	$S(w, w') \Rightarrow J(v, w')$
PAT_REF_FIS	$I(v) \wedge J(v, w) \wedge H(w) \Rightarrow \exists w' \cdot S(w, w')$
PAT_REF_GRD	$I(v) \wedge J(v, w) \wedge H(w) \Rightarrow G(v)$
PAT_REF_INV	$I(v) \wedge J(v, w) \wedge H(w) \wedge S(w, w') \Rightarrow$ $\exists v' \cdot (R(v, v') \wedge J(v', w'))$
PAT_REF_DLK$_i$	$G_i \Rightarrow \bigvee H_j(w)$
PAT_NEW_INV	$I(v) \wedge J(v, w) \wedge H(w) \wedge S(w, w') \Rightarrow J(v, w')$
PAT_NEW_DIV	$I(v) \wedge J(v, w) \wedge H(w) \wedge S(w, w') \Rightarrow$ $V(w) \in \mathbb{N} \wedge V(w') < V(w)$

Pattern Correctness

- Proofs are done for general case; actual expression of invariant, guards and before-after predicates may be unknown
- H, J, S and V correspond to the known objects produced by a pattern, they are substituted with concrete expressions
- I, G and R are unknown predicates, they can be restricted by pattern requirements

Checking *addinc* correctness (1)

PAT_FIS_INI	$\exists v' \cdot (v' \in \mathbb{Z} \land v' = 0)$
PAT_INV_INI	$v' = 0 \Rightarrow v' \in \mathbb{Z} \land I$
PAT_REF_FIS	$I \land v \in \mathbb{Z} \land v < 10 \Rightarrow$
	$\exists v' \cdot (v' \in \mathbb{Z} \land v' = v + 1)$
PAT_REF_GRD	$I \land v \in \mathbb{Z} \land G \land v < 10 \Rightarrow G$
PAT_REF_INV	$I \land v \in \mathbb{Z} \land v < 10 \land v' = v + 1 \Rightarrow v' \in \mathbb{Z}$

Checking *addinc* correctness (2)

Taking variant $V = 10 - v$ we prove the non-divergence of the transformed event e:

PAT_NEW_DIV $\quad I \wedge v \in \mathbb{Z} \wedge v < 10 \wedge v' = v + 1 \Rightarrow (10 - v) \in \mathbb{N} \wedge (10 - v) > (10 - v')$

Checking *addinc* correctness (3)

The relative deadlock freeness condition:

$$\text{PAT_REF_DLK} \quad I \wedge v \in \mathbb{Z} \wedge G_i \Rightarrow (v < 10 \wedge G_i) \vee \bigvee_{j \neq i} H_j(w)$$

- It cannot be discharged since we cannot demonstrate that $I \wedge G_i \Rightarrow \bigvee_{j \neq i} H_j(w)$.
- The pattern is **not** correct!

Fixing *addinc*

1. Strengthen the pattern requirements

$$\text{req_}evt\ e_2 \in \textbf{Events}$$
$$\text{req_}dlk\ e_2.\mathrm{guard} \Rightarrow e_1.\mathrm{guard}$$

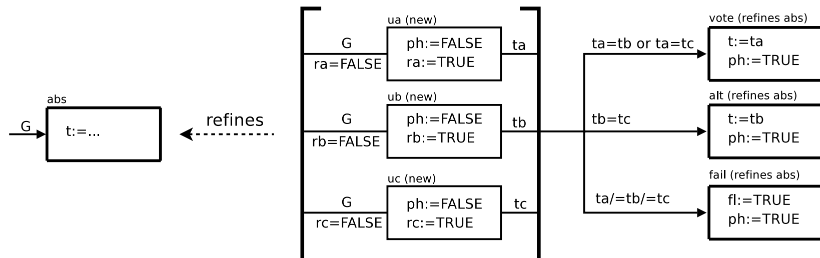
where $e.\mathrm{guard}$ is conjunction of guards of event e. In the latter case we add an empty event with implicit truth guard:

2. Add new event

$$\textbf{event}\ e_2$$

Then we have $I \wedge G_i \Rightarrow G_k \bigvee_{j \neq i \wedge j \neq k} H_j$ and also $G_i \Rightarrow G_k$ either due to the new requirements or because of the new event e_2.

Triple-Modular Redundancy Pattern



Triple-Modular Redundancy Pattern (1)

pattern *tmr*
parameters s, u, areq_*typing* $s \in \textbf{Variable} \wedge u \in \textbf{Event} \wedge a \in \textbf{Action}$
req_*action* $a \in u.\mathrm{actions} \wedge a.\mathrm{style} \neq (:=)$
req_*var* $s = a.\mathrm{variable}$

Triple-Modular Redundancy Pattern (2)

New sensors are modelled by adding new variables.

variable *sa*
  label $\mathtt{sa}$
  invariant $sa \in s.\mathrm{type}$
  action
    label *sa*
    style $s.\mathrm{init}.\mathrm{style}$
    expression $s.\mathrm{init}.\mathrm{expression}$

Sensor variables *sb* and *sc* are defined in the same manner.

Triple-Modular Redundancy Pattern (3)

Each sensor reports its status using a boolean flag variable

variable *ra*
  label `sa_on`
  invariant $ra \in \mathbb{B}$
  action $ra := FALSE$

Triple-Modular Redundancy Pattern (4)

There is a delay caused by collecting values from individual sensors and voting on them

variable *ph*
  label $s_\mathtt{ph}$
  invariant $ph \in \mathbb{B}$
  action $ph := FALSE$

Triple-Modular Redundancy Pattern (5)

Invariants:

$$
\begin{aligned}
J_1 &: \textbf{invariant } (ph = TRUE \land (sa = sb \lor sa = sc)) \Rightarrow s = sa \\
J_2 &: \textbf{invariant } (ph = TRUE \land sb = sc) \Rightarrow s = sb \\
J_3 &: \textbf{invariant } (ph = TRUE \land sa \neq sb \land sb \neq sc \land sa \neq sc) \Rightarrow \\
&\qquad fl = TRUE
\end{aligned}
$$

Triple-Modular Redundancy Pattern (6)

Values for the new sensors variables are generated by new events.

event *ua*
  label *u*_gena
  guard $ra = FALSE$
  guard u.guard
  action
    label *sa*
    style a.style
    expression a.expression
  action $ra := TRUE$
  action $ph := FALSE$

Triple-Modular Redundancy Pattern (7)

The abstract event providing sensor readings is refined using these rules

H_1 : **guard** $ra = TRUE \wedge rb = TRUE \wedge rc = TRUE$ **for** u

guard $sa = sb \vee sa = sc$ **for** u

style := **for** a
expression sa **for** a

action $ra := FALSE$ **for** u
action $rb := FALSE$ **for** u
action $rc := FALSE$ **for** u
action $ph := TRUE$ **for** u

Triple-Modular Redundancy Pattern (8)

Alternate output

event *alt*
  refines *u*
  label *u*_`alt`
  guard $sb = sc$
  action $s := sb$
  ...

Triple-Modular Redundancy Pattern (9)

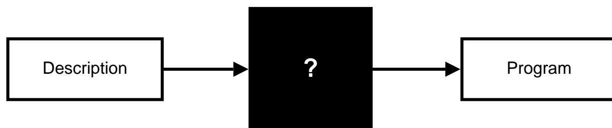
Sensor disagreement

event *fail*
  refines *u*
  label *u*_`fail`
  guard $sa \neq sb \wedge sb \neq sc \wedge sa \neq sc$
  action $fl := TRUE$
  ...

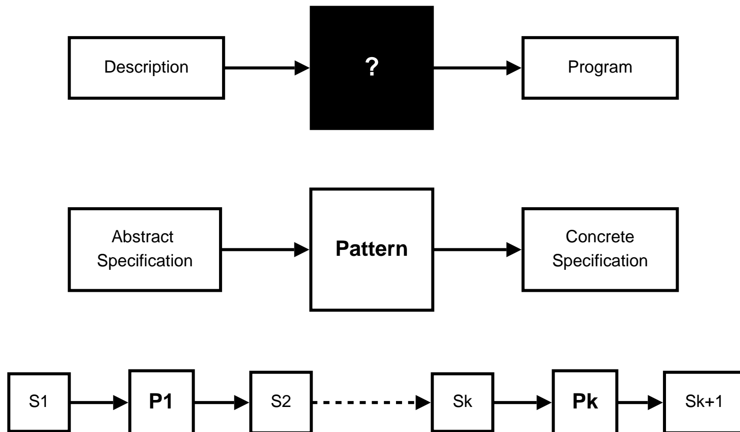
The *Finer* plugin

- Implemented
  - Intergration with the platform
  - Simple integrity checks for patterns
  - Pattern application
  - Access to the online pattern library
  - Capture of refinement
  - *Input in XML notation*
- To do
  - Generation of proof obligations
  - Integration with the platform prover
  - User-friendly pattern editing dialogs
  - Access to online pattern library
  - Pattern search
  - Documentation

Global Picture



Global Picture



Thanks

Questions?