Automatic Guidance for Refinement based Formal Methods

Maria Teresa Llano[1]
Andrew Ireland[1] Gudmund Grov[2]

[1]School of Mathematical and Computer Sciences
Heriot-Watt University

[2]School of Informatics
University of Edinburgh

AFM'10, July, 2010

Motivation

- The rigour of formal methods brings a lot of benefits to the development of systems.
- However, it is still not widely used outside specific domains.
- A major problem is the need for expertise in:
    - Formal modelling.
    - Theorem proving.
    - Understanding the close relationship between proof and modelling.

Goal: To abstract away from the complexities of proof obligations, providing high-level modelling guidance.

Here, with a focus on refinement.

Proof planning

Proof planning uses proof patterns to automate the search for proofs.

- Automatic proof failure analysis and patching.
- Reusability of proof strategies.

Reasoned Modelling

An approach that provides high-level modelling guidance by combining proof and modelling patterns.

- Currently the ideas of reasoned modelling are being developed in Event-B.
- We are working in *Refinement Plans*, a type of reasoned modelling method that focus on refinement.

Event-B

- A formalism that supports modelling and reasoning of discrete event systems.
- Uses a *posit-and-prove* style of modelling.
- Promotes the evolution of models through *refinement*.
- Uses mathematical proof to verify the consistency between refinement levels.

Event-B Example: A logging system

Records whether a resource is allocated or unallocated.

Context

Sets *RESOURCES*

Machine

Variables: *resources*, *allocated*, *unallocated*

Invariants: $resources \in \mathbb{P}(RESOURCES)$

$allocated \subseteq resources$

$unallocated \subseteq resources$

$allocated \cap unallocated = \varnothing$

$allocated \cup unallocated = resources$

Events ...

Event $unallocate \widehat{=}$

any r

where $r \in allocated$

then $allocated = allocated \setminus \{r\}$

$unallocated = unallocated \cup \{r\}$

end

Refinement in Event-B

Handle the complexity of large systems through the use of abstraction.

Refinement in Event-B:

- Existing events can be split, merged or modified.
- New events can be added.
- Variables can be added and/or removed.
- Gluing invariants: relate the state of the abstract model with the state of the concrete model.

All refinement steps are verified through proof obligations.

Refinement Plans

- Classify common patterns of refinement at the level of models.
- Combine modelling and proof knowledge.
- Detect partial matches of known patterns of refinement in a development.
- Provide user guidance in terms of modelling decisions.

Roles of refinement plans

- **Correcting refinements:** modifications to flawed refinements.
- **Layering refinements:** introduction of intermediate layers of abstraction in complex refinements.
- **Abstracting refinements:** reduction of the initial complexity of a development.
- **Suggesting refinements:** suggesting alternative refinement steps.
- **Increasing proof automation:** associated proof patterns.

Structure of refinement plans

$$
\begin{aligned}
\textbf{refinement plan} &= \textbf{refinement method}\\
&+ \textbf{proof methods}\\
&+ \textbf{critics}
\end{aligned}
$$

Refinement method

- Describes a common pattern of refinement (abstract model, concrete model and gluing invariant).
- Represented with declarative preconditions.
- These preconditions identify partial or complete instances of the pattern within a user's refinement.

Refinement method example: ***sets-to-function***

An instance of this pattern requires the:

- abstract model to contain a partition of sets (many variables),
- concrete model to replace the partition with a function (one variable)

Abstract model	**Concrete model**
$state1 \subseteq Elements$	$Status = \{STATE1, STATE2\}$
$state2 \subseteq Elements$	$fStatus \in Elements \rightarrow Status$
$state1 \cap state2 = \varnothing$	$\underline{state1 = fStatus^{-1}[\{STATE1\}]}$
$state1 \cup state2 = Elements$	$\underline{state2 = fStatus^{-1}[\{STATE2\}]}$

Proof methods[1]

- Reasoning patterns associated to a refinement method.
- Proof methods are used when a user's refinement fully matches with a pattern but it has a set of unproven POs.
- The use of proof methods and the analysis of partial success at the level of proof planning represents future work.

[1] Taken from Computational Logic [Bundy, 1991]

Critics

- Exception handling mechanism.
- If a partial instance of a refinement method is found, critics are applied.
- Represented with declarative preconditions.
- All preconditions must succeed for a critic to be applicable.
- Modelling guidance to overcome the failure is automatic generated (e.g. change guard/action/(gluing) invariants).
- The decision of applying/choosing guidance is left to the user.

Example: Logging system applied to the sets-to-function refinement method

Precondition: There must exists a set of gluing invariants with the pattern:

$$\textbf{stateVariable} = \textbf{concreteFunction}^{-1}[\{\textbf{stateConstant}\}]$$

Abstract model	**Concrete model**
$allocated \subseteq Resources$	$Status = \{ALLOCATED, UNALLOCATED\}$
$unallocated \subseteq Resources$	$rStatus \in Resources \rightarrow Status$
$unallocated \cap unallocated = \varnothing$	
$unallocated \cup unallocated = Resources$	*Missing gluing invariants*

There exists a partial instance of the refinement method!

Gluing_Invariant_Speculation critic – Key ideas

1. There exists a failed guard strengthening PO in the concrete model with the form:

$$\exists failed\_po \in \{\langle \_, \_, \_, PO\rangle \in POs \mid failed\_proof(PO)\}.$$
$$failed\_po = \langle M, E, \_/GRD, (\Delta, \underbrace{stateFunction(x) = Y}_{concrete\ guard} \vdash \underbrace{x \in \{z\}}_{abstract\ guard})\rangle$$

2. That by adding the gluing invariant pattern to the set of hypotheses the failed PO is provable:

$$provable\ (\Delta, stateFunction(x) = Y, \underline{\{z\} = stateFunction^{-1}[\{Y\}]} \ \vdash \ x \in \{z\})$$

Instantiation of the critic: logging example

1. Failed POs with the form "$\Delta, stateFunction(x) = Y \vdash x \in \{z\}$": ✔

$$..., rStatus(r) = ALLOCATED \vdash r \in \{allocated\}$$
$$..., rStatus(r) = UNALLOCATED \vdash r \in \{unallocated\}$$

2. The addition of the gluing invariant discharges the POs: ✔

$$provable\ (..., rStatus(r) = ALLOCATED, \underline{\{allocated\} = rStatus^{-1}[\{ALLOCATED\}]} \vdash r \in \{allocated\})$$

(A similar instantiation is given for state unallocated)

All preconditions succeed, then the guidance is the addition of the gluing invariants in the concrete model.

The REMO[2] tool

[2] The REMO acronym follows from REasoned MOdelling

Ongoing and future work

- Development of more refinement plans and their evaluation through case studies.
- Explore the role of refinement plans for:
    - Guiding users in their initial choice of refinement.
    - Suggesting intermediate refinement steps.
- Tool implementation: *REMO*.
- Development of the proof planning mechanism to exploit proof methods.

Conclusions

- Refinement plans aim at providing modelling guidance by automatically analysing specifications that lie just outside a known pattern of refinement.
- While the analysis of failure and generation of guidance is automatic, the decision as to whether or not to take the guidance on offer will be left to the user.
- We believe that this approach will enable us to turn low-level proof-failures into high-level modelling guidance.

Refinement plan and critic schemas

PLAN(*Name*)
  INPUTS:
    PO_SET $\{POs\}$
    MODELS $\{AM, CM\}$
  REFINEMENT METHOD:
    1. *Precondition*
    ....
    I. *Precondition*
  PROOF METHODS:
    1. *Proof_Method*
    ....
    J. *Proof_Method*
  CRITICS:
    1. *Critic_Name*
    ....
    K. *Critic_Name*

CRITIC(*Name*)
  INPUTS:
    PO_SET $\{POs\}$
    MODELS $\{AM, CM\}$
    R_INSTANCES $\{Instances\}$
    P_INSTANCES $\{Instances\}$
  PRECONDITIONS:
    1. *Precondition*
    ....
    L. *Precondition*
  OUTPUTS:
    PATCH *patch description*
    GUIDE *guidance description*