Model Based Refinement Tools in General and Frog in Particular

R. Banach,
School of Computer Science,
University of Manchester, UK
Contents:

1. Introduction.
2. Model Based Refinement Methods.
3. Configurable Semantics.
5. Frog.
6. Conclusions.
1. Introduction.

- There are a lot of different model based refinement methods.
- The wide choice can put off applications developers ... they have many more things to worry about than minutiae of different development techniques.
- Yet, modern tools bring mainstream-sized problems into the remit of formal development techniques.
1. Introduction.

• There are a lot of different model based refinement methods.

• The wide choice can put off applications developers ... they have many more things to worry about than minutiae of different development techniques.

• Yet, modern tools bring mainstream-sized problems into the remit of formal development techniques.

• Let’s try not to put off the developers ...

• Look at general features of model based refinement methods ... and draw some conclusions for tools.
2. Model Based Refinement Methods.

Model based techniques combine (and differ according to) a number of criteria.
2. Model Based Refinement Methods.

Model based techniques combine (and differ according to) a number of criteria.

- Formal language.
  
  Each technique needs to be precise about linguistic issues.
  
  - writing down the system models
    - states ... (predicates, more or less universal)
    - transitions ... (relations, assignments, predicate transformers, ...
      much more variety)
  
  - building tools
2. Model Based Refinement Methods.

Model based techniques combine (and differ according to) a number of criteria.

- Formal language.
  
  Each technique needs to be precise about linguistic issues.
  - writing down the system models
    - states ... (predicates, more or less universal)
    - transitions ... (relations, assignments, predicate transformers, ... much more variety)
  - building tools

- Granularity and Naming.
  
  Always need to relate concrete to abstract (collections of) steps.
  - (1,1) + name correspondence very common ...
    - convenient for theory ...
    - bad for practice (if hardwired into tools) ...

• Concrete-Abstract Fidelity.

Each technique has a simulation condition.
  — established via the fidelity PO ... usually a sufficient condition ...
  — (almost) universal agreement among techniques
• Concrete-Abstract Fidelity.

Each technique has has a simulation condition.
  — established via the fidelity PO ... usually a sufficient condition ...
  — (almost) universal agreement among techniques

• Notions of Correctness.

  When do abstract steps have corresponding concrete ones?
  — connected with abstract → concrete guarantees ...
  — much variety among techniques
• Concrete-Abstract Fidelity.

  Each technique has has a simulation condition.
  — established via the fidelity PO ... usually a \textit{sufficient condition} ...
  — (almost) universal agreement among techniques

• Notions of Correctness.

  \textit{When} do abstract steps have corresponding concrete ones?
  — connected with \textit{abstract} $\rightarrow$ \textit{concrete} \textit{guarantees} ...
  — much variety among techniques

• Interpretation.

  How do statements in the logic correspond to phenomena in the real world?
  — connected with Notions of Correctness ...
  — usually easy when formulae evaluate to \textbf{TRUE},
     but what about \textbf{FALSE}?
• Trace Inclusion.

Basis of the main diagnostic technique in practice.
— weak trace inclusion: \((\forall \text{ConcTrace} \cdot (\exists \text{AbsTrace} \cdot \text{simulation}))\)
— strong trace inclusion: weak + ... extendable inductively
• Trace Inclusion.
  
  Basis of the main diagnostic technique in practice.
  — weak trace inclusion: \( (\forall ConcTrace \cdot (\exists AbsTrace \cdot simulation) ) \)
  — strong trace inclusion: weak + ... extendable inductively

• Composition.
  
  Big systems are built out of small pieces.
  — do refinement techniques compose?
  — much variety among techniques
• Trace Inclusion.

  Basis of the main diagnostic technique in practice.
  — weak trace inclusion: \((\forall \text{ConcTrace} \quad \exists \text{AbsTrace} \quad \text{simulation})\)
  — strong trace inclusion: weak + ... extendable inductively

• Composition.

  Big systems are built out of small pieces.
  — do refinement techniques compose?
  — much variety among techniques

Even within a fairly closely related set of formalisms such as Z, B, Event-B, ASM, there is a considerable variety of views on these issues ....
• Trace Inclusion.
  
  Basis of the main diagnostic technique in practice.
  — weak trace inclusion: \( \forall \text{ConcTrace} \bullet (\exists \text{AbsTrace} \bullet \text{simulation}) \)
  — strong trace inclusion: weak + ... extendable inductively

• Composition.
  
  Big systems are built out of small pieces.
  — do refinement techniques compose?
  — much variety among techniques

Even within a fairly closely related set of formalisms such as Z, B, Event-B, ASM, there is a considerable variety of views on these issues .... and that’s without considering VDM, RAISE, Refinement Calculi, IO-Automata, etc.
• Trace Inclusion.

  Basis of the main diagnostic technique in practice.
  — weak trace inclusion:  \( (\forall \text{ConcTrace} \cdot (\exists \text{AbsTrace} \cdot \text{simulation})) \)
  — strong trace inclusion: weak + ... extendable inductively

• Composition.

  Big systems are built out of small pieces.
  — do refinement techniques compose?
  — much variety among techniques

Even within a fairly closely related set of formalisms such as Z, B, Event-B, ASM, there is a considerable variety of views on these issues .... and that’s without considering VDM, RAISE, Refinement Calculi, IO-Automata, etc.

The main lesson is that these differences amount to design decisions.
3. Configurable Semantics.

Many similarities, yet many detailed differences.

- Could easily imagine many of the detailed design decisions re. semantics of the formalisms being taken in alternative frameworks.
- Lots of such things have been explored in (say) theses etc.
3. Configurable Semantics.

Many similarities, yet many detailed differences.

- Could easily imagine many of the detailed design decisions re. semantics of the formalisms being taken in alternative frameworks.
- Lots of such things have been explored in (say) theses etc.

Semantic alterations adopted in real systems ... driven by applications needs.
- ProB turns original B preconditions into additional guards.
- Domain checking for partial operations in original B strengthened. (Now adopted in Event-B.)
3. Configurable Semantics.

Many similarities, yet many detailed differences.

- Could easily imagine many of the detailed design decisions re. semantics of the formalisms being taken in alternative frameworks.

- Lots of such things have been explored in (say) theses etc.

Semantic alterations adopted in real systems ... driven by applications needs.

- ProB turns original B preconditions into additional guards.

- Domain checking for partial operations in original B strengthened. (Now adopted in Event-B.)

- More recent methodologies/tools are more accommodating of applications needs.

- Intersection of refinement models more important than union.
4. Tools.

Most tools (hitherto) have committed to a single semantics, which has been hardwired. Increasingly, some flexibility is seen as beneficial.
4. Tools.

Most tools (hitherto) have committed to a single semantics, which has been hardwired. Increasingly, some flexibility is seen as beneficial.

Examples:

**Rodin:** A tool for supporting Event-B. Designed on top of Eclipse, a host of plugins manage the Event-B development process. Although Event-B semantics is built-in, some flexibility is visible; eg. ‘new events’ are not forced to be convergent.
4. Tools.

Most tools (hitherto) have committed to a single semantics, which has been hardwired. Increasingly, some flexibility is seen as beneficial.

Examples:

**Rodin:** A tool for supporting Event-B. Designed on top of Eclipse, a host of plugins manage the Event-B development process. Although Event-B semantics is built-in, some flexibility is visible; eg. ‘new events’ are not forced to be convergent.

**Frog:** An experimental tool (Simon Fraser’s Ph.D.) for supporting retrenchment (and its interworking with refinement). Since there are many kinds of refinement and of retrenchment, flexibility is achieved by making POs fully programmable.
4. Tools.

Most tools (hitherto) have committed to a single semantics, which has been hardwired. Increasingly, some flexibility is seen as beneficial.

Examples:

**Rodin**: A tool for supporting Event-B. Designed on top of Eclipse, a host of plugins manage the Event-B development process. Although Event-B semantics is built-in, some flexibility is visible; eg. ‘new events’ are not forced to be convergent.

**Frog**: An experimental tool (Simon Fraser’s Ph.D.) for supporting retrenchment (and its interworking with refinement). Since there are many kinds of refinement and of retrenchment, flexibility is achieved by making POs fully programmable.

**KIV**: A theorem prover aimed at program verification, so very flexible. The user can decide what the refinement theory is, and set up and prove the relevant theorems (though this puts more responsibility on the user). Used with great success in the Mondex challenge.
4. Tools.

Most tools (hitherto) have committed to a single semantics, which has been hardwired. Increasingly, some flexibility is seen as beneficial.

Examples:

**Rodin:** A tool for supporting Event-B. Designed on top of Eclipse, a host of plugins manage the Event-B development process. Although Event-B semantics is built-in, some flexibility is visible; e.g. ‘new events’ are not forced to be convergent.

**Frog:** An experimental tool (Simon Fraser’s Ph.D.) for supporting retrenchment (and its interworking with refinement). Since there are many kinds of refinement and of retrenchment, flexibility is achieved by making POs fully programmable.

**KIV:** A theorem prover aimed at program verification, so very flexible. The user can decide what the refinement theory is, and set up and prove the relevant theorems (though this puts more responsibility on the user). Used with great success in the Mondex challenge.
5. Frog

Frog grew out of the desire to have a flexible tool for experimenting with retrenchment. Experimenting with retrenchment means working with refinement too.
5. Frog

Frog grew out of the desire to have a flexible tool for experimenting with retrenchment. Experimenting with retrenchment means working with refinement too.

Lots of different notions of refinement (as we saw) ... so lots of retrenchments too? Preferably not ... strong desire to avoid retrenchment balkanisation.
5. Frog

Frog grew out of the desire to have a flexible tool for experimenting with retrenchment. Experimenting with retrenchment means working with refinement too.

Lots of different notions of refinement (as we saw) ... so lots of retrenchments too? Preferably not ... strong desire to avoid retrenchment balkanisation.

Observation: when it comes down to actual example applications, all kinds of refinement notions generate pretty similar looking VCs. So:

1. Design notions of retrenchment round POs/VCs.
2. To cover all the bases, need flexibility re. the form of these POs/VCs.
Typical POs

A typical refinement PO:

\[
G(u, v) \land (i = j) \land Op_C(v, j, v', p) \Rightarrow \\
(\exists u', o \cdot Op_A(u, i, u', o) \land G(u', v') \land (o = p))
\]

A typical retrenchment PO:

\[
G(u, v) \land P_{Op}(i, j, u, v) \land Op_C(v, j, v', p) \Rightarrow \\
(\exists u', o \cdot Op_A(u, i, u', o) \land ((G(u', v') \land Op(p, o, u', v', u, v, i, j)) \\
\lor Op(u', v', o, p; u, v, i, j)))
\]

Built out of relations for:
- defining operations
- defining various kinds of relationship between data spaces etc.
Configurability

The Frog Tool makes configurable most aspects of a posit-and-prove based verification tool ... 

• what a machine (transition system) consists of,
• what constitutes correctness for a machine,
• what relationships between machines are permitted,
• what a relationship between machines consists of,
• what constitutes correctness for a relationship between machines.

Certain things are hardwired nevertheless ...

• mathematical innards expressed in Z,
• operation-level correspondence via name identity.
Example

MACHINE myNumberMachine
TYPE simpleMachine
SECTION standard_toolkit
STATE
  a : IN
INITIALISATION
  | a = 0
OPERATION increment ≡
POST
  | a' = a + 1
END increment
END myNumberMachine
Frog-CCL config file for *simpleMachine* (part)

```ccl
DEFINE MACHINE simpleMachine
CLAUSES
( NAME = state, LEVEL = MACHINE,
  REQUIREMENT = OPTIONAL, CONTENT = SCHEMA_TEXT,
  RELATION = <state> ),
( NAME = initialisation, LEVEL = MACHINE,
  REQUIREMENT = OPTIONAL, CONTENT = SCHEMA_TEXT,
  RELATION = <state> ),

... ... ... ... ... ...

( CONSTRUCT_LEVEL,
  ( #u @ u : state & u : initialisation ) ),

... ... ... ... ... ...

( OPERATION_LEVEL,
  ( !u,i @ u : state & i : operation.inputs
    & <u,i> : operation.pre
    => ( #u',o @ u' : state & o : operation.outputs
      & <u,u',i,o> : operation.post ) )

END
```
Processing the Initialisation

\[ \vdash ? \exists u \cdot u \in \text{state} \land u \in \text{initialisation} \]

*state* is a schema whose signature is given by

\{ state\_schema \cdot \theta \text{ state\_schema} \}

\text{state\_schema} == [ a : \mathbb{N} ]

\{ a : \mathbb{N} \cdot a \}

\[ \vdash ? \exists u \cdot u \in \{ a : \mathbb{N} \cdot a \} \land u \in \text{initialisation} \]
Processing the Initialisation

\[ \vdash \exists u \cdot u \in \text{state} \land u \in \text{initialisation} \]

state is a schema whose signature is given by

\[ \{ \text{state}_\text{schema} \cdot \emptyset \text{state}_\text{schema} \} \]

\[ \text{state}_\text{schema} == [ a : \mathbb{IN} ] \]

\[ \{ a : \mathbb{IN} \cdot a \} \]

\[ \vdash \exists u \cdot u \in \{ a : \mathbb{IN} \cdot a \} \land u \in \text{initialisation} \]

initialisation is a schema whose signature is given by

\[ \{ \text{initialisation}_\text{schema} \cdot \emptyset \text{state}_\text{schema} \} \]

\[ \text{initialisation}_\text{schema} == [ \text{state}_\text{schema} \mid a = 0 ] == [ a : \mathbb{IN} \mid a = 0 ] \]

\[ \{ a : \mathbb{IN} \mid a = 0 \} \]

\[ \vdash \exists u \cdot u \in \{ a : \mathbb{IN} \} \land u \in \{ a : \mathbb{IN} \mid a = 0 \} \]
Processing an Operation

\[ !u, i @ u : \text{state} & i : \text{operation.inputs} \\
& <u, i> : \text{operation.pre} \\
\Rightarrow ( #u', o @ u' : \text{state} & o : \text{operation.outputs} \\
& <u, u', i, o> : \text{operation.post} ) \]

becomes

\[ \vdash \forall u, i \cdot u \in \text{state} & i \in \text{operation.inputs} \\
& \langle u, i \rangle \in \text{operation.pre} \\
\Rightarrow ( \exists u', o \cdot u' \in \text{state} & o \in \text{operation.outputs} \\
& \langle u, u', i, o \rangle \in \text{operation.post} ) \]

becomes

\[ \vdash \forall u \cdot u \in \{ a : \mathbb{N} \} \\
\Rightarrow ( \exists u' \cdot u' \in \{ a : \mathbb{N} \} \\
& \langle u, u' \rangle \in \{ a : \mathbb{N}; a' : \mathbb{N} | a' = a + 1 \} ) \]
Frog-CCL config file for the *retrenchment* relationship (part)

```plaintext
DEFINE RELATIONSHIP retrenchment
    CLAUSES
    ...
    ( OPERATION_LEVEL,
        ( !u,v,v',i,j,p
          @ <v,v',j,p> : TO_MACHINE(operation.post)
          & <u,v> : ramifications.retrieve
          & <u,v,i,j> : ramifications.within
          & <u,i> : FROM_MACHINE(operation.pre)
        ) => ( #u',o @ <u,u',i,o> : FROM_MACHINE(operation.post)
          & ( ( <u',v'> : ramifications.retrieve
             & <u,v,u',v',i,j,o,p> :
             ramifications.output )
             | <u,v,u',v',i,j,o,p> :
             ramifications.concedes )
           )
      ))
```
6. Conclusions.

We’ve briefly reviewed model based refinement.

Commonalities and differences; many design decisions, not ‘laws of nature’. Differences certainly inhibit tool interworking when too much is hardwired.

Increasing flexibility is in evidence, and is to be encouraged.

There is a spectrum of approaches toward tool flexibility.

Aligns with the Evidential Tool Bus idea from SRI.

 Aligns with the Verification Grand Challenge.