REFINEMENT AND INSTITUTIONS

Marie Farrell

Supervisors: Dr Rosemary Monahan & Dr James Power

Principles of Programming Research Group
PROBLEM

- Difficult to combine proofs from different systems
PROPOSED SOLUTION

- Provide a theoretical framework for proof sharing
- Mathematically define each formalism
  - Including proof requirements
- Mathematically define how to integrate formalisms
- Reason about systems in the integrated formalism
  - Sharing proof components
HYPOTHESIS

- Institutions can provide this framework
- Each formalism can be defined by an institution
- Institutions can be combined and components can be shared
Π-INSTITUTIONS

A π-institution is a triple \((\text{Sign}, \phi, \{\text{Cn}_\Sigma\}_{\Sigma: \text{Sign}})\) consisting of

1. A category \(\text{Sign}\) (of signatures)
2. A functor \(\phi: \text{Sign} \rightarrow \text{Set}\)
3. A consequence operator \(\text{Cn}_\Sigma\)
   - \(\Sigma\) is an object of \(\text{Sign}\) (i.e. \(\Sigma\) is in the alphabet)
   - \(\text{Cn}_\Sigma\) takes a set of axioms \(A \subseteq \phi(\Sigma)\) and gives all properties that can be deduced from \(A\)
We model systems at different levels of abstraction.

We can map between these levels using refinement.

This process can be mathematically verified.
REDUCING NONDETERMINISM

Classic example: Converting an NFA to a DFA

This one is nondeterministic when \( a=b \)

This one is deterministic

\[
\begin{align*}
\text{if } a \leq b & \rightarrow a := a - b \\
\begin{array}{l}
\text{if } b \geq a \\
\text{if } a \not\leq b
\end{array} & \rightarrow b := b - a \\
\Rightarrow & \\
\text{if } a \leq b & \rightarrow a := a - b \\
\begin{array}{l}
\text{if } a \leq b \\
\text{if } a \not\leq b
\end{array} & \rightarrow b := b - a
\end{align*}
\]
THEORIES OF REFINEMENT

- Carroll Morgan, Ralph Johan Back and Joseph Morris
- Based on Dijkstra’s language of guarded commands and weakest precondition calculus.
MORGAN’S REFINEMENT

- Weakening the precondition
- Strengthening the postcondition
- Introducing local variables
- Renaming local variables
- Introducing logical constants
- Eliminating logical constants
- Expanding the frame

- Introducing skip
- Introducing abort
- Introducing assignment
- Introducing sequential composition
- Introducing alternation
- Introducing iteration
MORGAN’S REFINEMENT

Introducing alternation

\[ w: [\text{pre} \land (\forall i \cdot G_i), \text{post}] \]

\[ \subseteq \]

\[ \text{if } (\Box i \cdot G_i \rightarrow w: [\text{pre} \land G_i, \text{post}]) fi \]
BACK’S REFINEMENT

Figure: The Refinement Calculus Hierarchy
BACK’S REFINEMENT

- Similar rules to Morgan’s refinement calculus
- Example
  - Introduce conditional: \( S \subseteq [g_1 \cup \cdots \cup g_n]; \text{if } g_1 \rightarrow S \square \cdots \square g_n \rightarrow S \)
GENERAL REFINEMENT

- 3 main components:
  1. Set of entities – specifications and implementations
  2. Set of contexts \( \Xi \) – the environment with which the entities interact
  3. A user – observations of a system \( O \)

\[
A \subseteq C \\
\equiv \\
\forall x \in \Xi. \ O([C]_x) \subseteq O([A]_x)
\]
LAYERS OF REFINEMENT
LAYERS OF REFINEMENT

Refinement within and between layers
Mathematically this vertical refinement is a Galois connection between the layers.

Given two posets \((A, \leq_A)\) and \((B, \leq_B)\). A Galois connection between these posets consists of two maps \(f: A \rightarrow B\) and \(g: B \rightarrow A\), such that for all \(a \in A\) and \(b \in B\), we have

- \(a \leq_A f(g(a))\)
- \(f(g(b)) \leq_B b\)
MACHINE
  mac1
VARIABLES
  cars_go
  peds_go
INvariants
  inv1 : cars_go ∈ BOOL
  inv2 : peds_go ∈ BOOL
  inv3 : ¬(peds_go=TRUE ∧ cars_go=TRUE)

EVENTS
  INITIALISATION
    STATUS
    ordinary
  BEGIN
    act1 : cars_go := FALSE
    act2 : peds_go := FALSE
  END

  set_peds_go
    STATUS
    ordinary
  WHEN
    grd1 : cars_go = FALSE
  THEN
    act1 : peds_go := TRUE
  END

  set_peds_stop
    STATUS
    ordinary
  BEGIN
    act1 : peds_go := FALSE

MACHINE
  mac2
REFINES
  mac1
SEES
  cont
VARIABLES
  cars_colour
  peds_colour
INvariants
  inv1 : peds_colour ∈ {red, green}
  inv2 : peds_go = TRUE ⇔ peds_colour = green
  inv3 : cars_colour ∈ {red, green}
  inv4 : cars_go = TRUE ⇔ cars_colour = green

EVENTS
  INITIALISATION
    STATUS
    ordinary
  BEGIN
    act1 : cars_colour := red
    act3 : peds_colour := red
  END

  set_peds_green
    STATUS
    ordinary
  REFINES
    set_peds_go
  WHEN
    grd1 : cars_colour = red
  THEN
    act1 : peds_colour := green
  END

  set_peds_red
    STATUS
    ordinary
  REFINES
    set_peds_stop
  BEGIN
    act1 : peds_colour := red
  END
REFINEMENT IN JML

```java
package jmlpractice;

public class refineExamples{
    //@ public model non_null String name;
    private /*@ non_null@*/ String fullName;
    //@ private represents name = fullName;
}

public int second, minute, hour;
//@ public model long time;
//@ private represents time = second + minute*60 + hour*60*60;
```
REFINEMENT IN JML

Specification Inheritance

extends
FUTURE WORK

1. Specify a $\pi$-institution for refinement in at least two formalisms
2. Complete refinement case studies in both formalisms
3. Use $\pi$-institutions to combine proofs in these formalisms