AN ANALYSIS OF REFINEMENT CALCULI

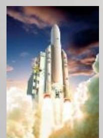
Marie Farrell

Supervisors: Dr Rosemary Monahan & Dr James Power

Principles of Programming Research Group

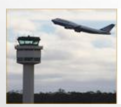
“A LOGICAL FRAMEWORK FOR INTEGRATING SOFTWARE MODELS VIA REFINEMENT”

Last PG Seminar



- ❖ Motivation

- ❖ Formal Methods
- ❖ Event B



- ❖ Refinement
- ❖ Overview of PhD topic
- ❖ Some Maths

Today

- ❖ Overview of PhD topic
- ❖ Refinement Calculi
- ❖ Event B and JML
- ❖ Theoretical Foundations

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 (Sign, ϕ, $\{Cn_\Sigma\}_{\Sigma:Sign}$) consisting of
  1. A category Sign (of signatures)
  2. A functor ϕ:Sign -> Set
  3. A consequence operator Cn_Σ
     - Σ is an object of Sign (i.e. Σ is in the alphabet)
     - Cn_Σ takes a set of axioms $A \subseteq \varphi(\Sigma)$ and gives all properties that can be deduced from A

PROPERTIES OF Π - INSTITUTIONS

(RQ1) $A \subseteq Cn_{\Sigma}(A)$	(Extensiveness)
(RQ2) $Cn_{\Sigma}(Cn_{\Sigma}(A)) = Cn_{\Sigma}(A)$	(Idempotence)
(RQ3) $Cn_{\Sigma}(A) = \bigcup_{B \subseteq A, B\ finite} Cn_{\Sigma}(B)$	(Compactness)
(RQ4) $\varphi(\mu)(Cn_{\Sigma}(A)) \subseteq Cn_{\Sigma'}(\varphi(\mu)(A))$	(Structurality)

EVENT B

- The Event B formal specification language is used in the verification of safety critical systems

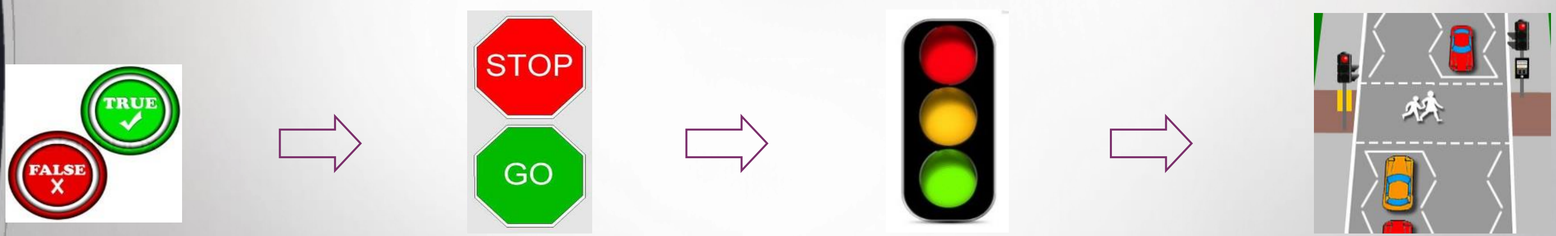


- Event B models are an instance of the specification

- Event B supports refinement

REFINEMENT

- We model systems at different levels of abstraction



- We can map between these levels using refinement
- This process can be mathematically verified

THEORIES OF REFINEMENT

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

GENERAL REFINEMENT

"The abstract entity **A** is refined by the concrete entity **C** if no user of **A** could observe if they were given **C** in its place"

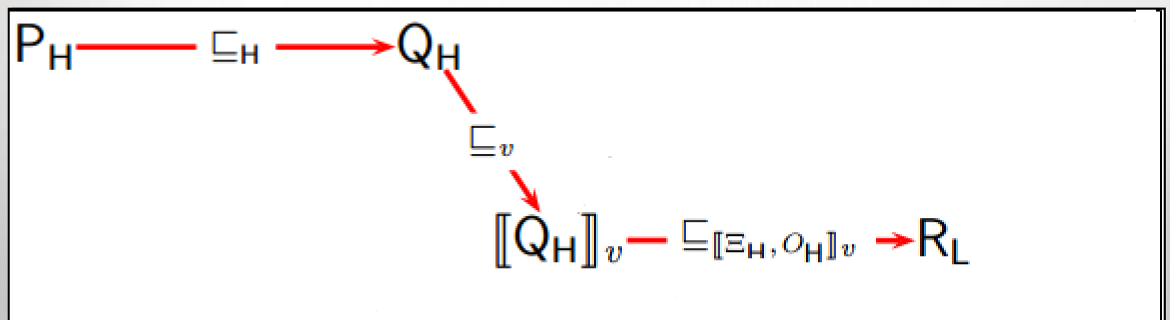
Liskov Substitution

GENERAL REFINEMENT

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

$$\mathsf{A} \sqsubseteq_{\Xi,O} \mathsf{C} \triangleq \forall x \in \Xi . O([\mathsf{C}]_x) \subseteq O([\mathsf{A}]_x)$$

SPECIAL THEORIES



GALOIS CONNECTIONS

- 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
  ctx1
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
```

Gluing Invariant

JAVA MODELLING LANGUAGE (JML)

❖ Specifications are annotations:

```java
/*@ requires array.length>0;
    ensures sorted(array);
 @*/
public int [] sort(int [] array){

    int temp =0;

    for(int j=0;j<array.length-1;j++){

        if(array[j]>array[j+1]){
            temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
        }
    }
    return array;
}
```

```java
/*@ requires a.length>0;
    assignable \nothing;
@*/
public boolean sorted(int [] a)
{
    boolean valid = true;
    for(int i=0;i<a.length-1;i++)
    {
        if(a[i]>a[i+1])
        {
            valid = false;
            break;
        }
    }
    return valid;
}
```

REFINEMENT IN JML

- JML supports refinement as specification inheritance

```
//@ public model non_null String name;
private /*@ non_null @*/ String fullName;
//@ private represents name <- fullName;
```

```java
package org.jmlspecs.samples.jmltutorial;

//@ refine "Person.java";

public class Person {
  private /*@ spec_public non_null @*/
    String name;
  private /*@ spec_public @*/
    int weight;

  /*@ public invariant !name.equals("")
    @         && weight >= 0; @*/

  //@ also
  //@ ensures \result != null;
  public String toString();

  //@ also
  //@ ensures \result == weight;
  public /*@ pure @*/ int getWeight();

  /*@ also
    @ requires kgs >= 0;
    @ requires weight + kgs >= 0;
    @ ensures weight == \old(weight + kgs);
    @*/
  public void addKgs(int kgs);

  /*@ also
    @ requires n != null && !n.equals("");
    @ ensures n.equals(name)
    @   && weight == 0; @*/
  public Person(String n);
}
```

AIM

- ❖ 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

FUTURE WORK

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


Questions?

REDUCING NONDETERMINISM

$$
\begin{array}{l}
if\ a \le b \to a := a - b \\
\square\ b \ge a \to b := b - a \\
\qquad fi
\end{array}
$$

$$\sqsubseteq$$

$$
\begin{array}{l}
if\ a \le b \to a := a - b \\
\square\ a \nleq b \to b := b - a \\
\qquad fi
\end{array}
$$

This one is nondeterministic when a=b

This one is deterministic

Classic example: Converting an NFA to a DFA