PROBLEM

- Different formalisms do not integrate well e.g. Event B only models the specification and its proofs are not easily transferable to other formalisms.
PROPOSED SOLUTION

- Establish a theoretical framework within which refinement steps, and their associated proof obligations, can be shared between different formalisms
- Hypothesis: the theory of institutions can provide this framework and, we will construct an institution based specification of the Event B formalism
REFINEMENT

- In software engineering it is common to model systems at different levels of abstraction

- We can map between these different levels of abstraction in a verifiable way through a process known as refinement
Classic example: Converting an NFA to a DFA
THEORIES OF REFINEMENT

- Main theories developed by Carroll Morgan, Ralph Johan Back and Joseph Morris
- All three are based on Dijkstra’s language of guarded commands and weakest precondition calculus.
- Morgan takes a very program oriented view whereas Back appears to be much more theoretical with foundations in lattice and category theory. Morris extended Back’s work with prescriptions.
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 REFERENCE

Introducing alternation

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

Relations Category

State Transformer Category

Predicate Transformer Category

Predicate Category

Category of Truth Values
BACK’S REFINEMENT

- Similar rules to Morgan’s refinement calculus
- Example
  - Introduce conditional:

\[ S \subseteq [g_1 \cup \ldots \cup g_n]; \text{if } g_1 \rightarrow S \cup \ldots \cup g_n \rightarrow S \text{ fi}. \]
MORRIS REFINEMENT

- Extended Back’s calculus with prescriptions
- A prescription $P\parallel Q$ specifies a mechanism that when executed in a state satisfying $P$ will terminate in a state satisfying $Q$
  - $P$ and $Q$ are predicates

\[
P\parallel Q \subseteq s_1 \subseteq s_2 \subseteq \cdots \subseteq s\]
Morris Refinement

Given $P || Q$ there are 6 ways of choosing $s$ such that

$$P || Q \sqsubseteq s$$

1. Skip
2. Assignment
3. Prescription
4. If statement
5. Composition
6. Block

$P || Q \sqsubseteq R || S; T || U$ if $[P \Rightarrow R], [S \Rightarrow T]$ and $[U \Rightarrow Q]$
"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"
3 main components:
1. Set of entities – specifications and implementations
2. Set of contexts – the environment with which the entities interact
3. A user formalised by defining the set of observations that can be made when an entity is executed in a given context

Example: an entity as a motor, a context as the car in which the motor runs and the user as the driver of the car

\[ A \subseteq_{\Xi, O} C \triangleq \forall x \in \Xi. O([C]_x) \subseteq O([A]_x) \]
We can view each special model of refinement as a layer in the grand scheme of things each encompassing a set of entities and a refinement relation.

This allows us to interpret high level entities as low level entities using a semantic mapping, however, these low level entities cannot interact with the high level ones so the contexts must also be refined.
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

- \(a \leq_A f(g(a))\)
- \(f(g(b)) \leq_B b\)
Π-INSTITUTIONS

- Alternative to institution – replacing the notions of model and satisfaction by Tarski’s consequence operator

- Definition:
  - A π-institution is a triple \((\text{Sign}, \phi, \{Cn_\Sigma\}_{\Sigma \in \text{Sign}})\) consisting of
    1. A category \(\text{Sign}\) (of signatures)
    2. A functor \(\phi: \text{Sign} \to \text{Set}\) (set of formulae over each signature)
    3. For each object \(\Sigma\) of \(\text{Sign}\), a consequence operator \(Cn_\Sigma\) defined in the power set of \(\phi(\Sigma)\) satisfying for each \(A, B \subseteq \phi(\Sigma)\) and \(\mu: \Sigma \to \Sigma\)
      - (RQ1) \(A \subseteq Cn_\Sigma(A)\) (Extensiveness)
      - (RQ2) \(Cn_\Sigma(Cn_\Sigma(A)) = Cn_\Sigma(A)\) (Idempotence)
      - (RQ3) \(Cn_\Sigma(A) = \bigcup_{B \subseteq A, \text{finite}} Cn_\Sigma(B)\) (Compactness)
      - (RQ4) \(\phi(\mu)(Cn_\Sigma(A)) \subseteq Cn_{\Sigma'}(\phi(\mu)(A))\) (Structurality)
Axiom 1:

\[ |S| \leq \aleph_0 \]

Axiom 2:

If \( X \subseteq S \), then \( X \subseteq Cn(X) \subseteq S \)

Axiom 3:

If \( X \subseteq S \), then \( Cn(Cn(X)) = Cn(X) \)

Axiom 4:

If \( X \subseteq S \), then \( Cn(X) = \sum_{Y \subseteq X \text{ and } |Y| < \aleph_0} Cn(Y) \)

Axiom 5:

\( \exists x \in S \) such that \( Cn(\{x\}) = S \)
$f : A \rightarrow B$

$f(x) = Cn(x)$

$g : B \rightarrow A$

$g(x) = \bigcap_{Y \subseteq S \text{ and } X \subseteq Cn(Y)} Y$

Both posets are ordered by set theoretic inclusion
The Event B formal specification language is used in the verification of safety critical systems.

Event B models are an instance of the specification.
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
cntl
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
JML

- JML = Java Modelling Language
- 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;
}

/*@ 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

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

- Establish a theoretical framework within which refinement steps, and their associated proof obligations, can be shared between different formalisms
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