

Refinement via A Unified Theory of Testing

S. Reeves D. Streader

Department of Computer Science
University of Waikato

Refine 2008

Outline of Talk

- 1 Motivation
- 2 Tests and Refinement
 - Horizontal + Vertical
 - Parametrised Horizontal
 - Interface types
 - Vertical
- 3 Final
 - Observations
 - Summary
- 4 Comparison with literature
 - Subset Morphism and Ref
 - Divergence
- 5 Example

Motivation

Tests and Refinement

Horizontal + Vertical

Parametrised
Horizontal

Interface types
Vertical

Final

Observations
Summary

Comparison with literature

Subset Morphism
and Ref

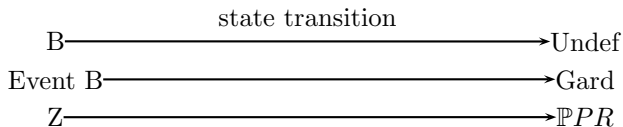
Divergence

Example

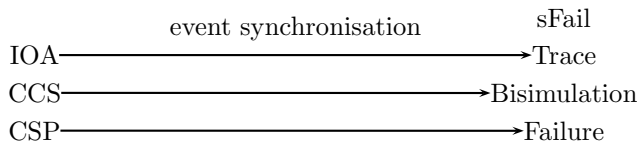
Unification What, How and Why

- What
 - Handshake processes CSP CCS ACP
 - Broadcast processes CSB IOA
 - Abstract data types
 - Operations

Currently



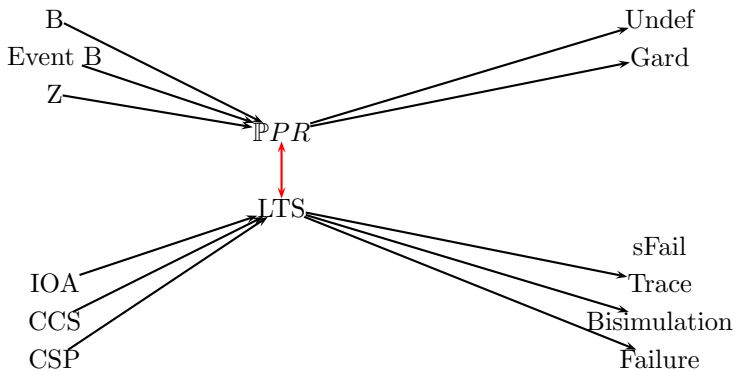
Each special theory has fixed interpretation!



Unification What, How and Why

- How
 - Operational semantics tell us part of the semantic story
 - Refinement can define an interpretation of the operational semantics.
 - One parametrised definition of refinement.
 - Different parameters for different interpretations.
- Why
 - Formalisation is hard so reuse where you can
 - Comparison brings fresh insights
 - Reinterpretation as design decision
 - Mixing event types (interpretations)

Refinement as Interpretation



Motivation

Tests and Refinement

- Horizontal + Vertical
- Parametrised
- Horizontal
- Interface types
- Vertical

Final

- Observations
- Summary

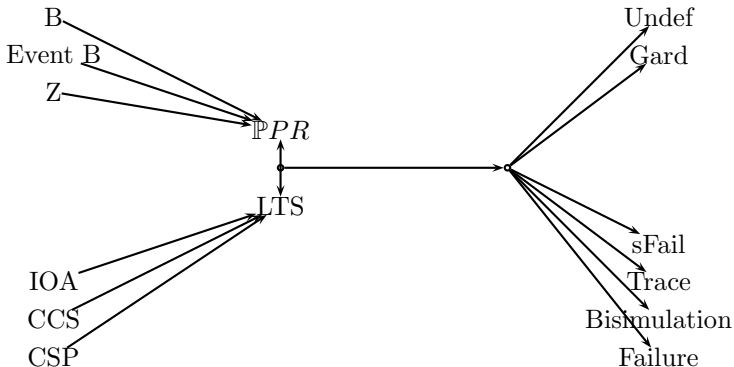
Comparison with literature

- Subset Morphism and Ref
- Divergence

Example

Refinement as Interpretation

+ Operation



Motivation

Tests and Refinement

Horizontal + Vertical
Parametrised
Horizontal
Interface types
Vertical

Final

Observations
Summary

Comparison with literature

Subset Morphism and Ref
Divergence

Example

Why use testing semantics?

- 1 Testing semantics justify the choice of refinement.
 - 155 definitions of refinement for processes with handshake actions.
- 2 By changing the tests we change the interpretation
- 3 Explain properties of formal models
 - singleton failure refinement is not data refinement
 - why divergence masks what we would like to see
 - not modelling the stimulus response nature of event synchronisation **changes determinism**

Horizontal + Vertical

1 General refinement and known examples (2003)

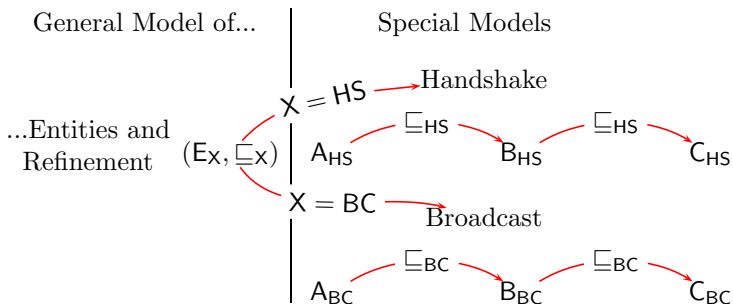
- Refinement is reduction of non-determinism
- Implementer is free to build any refinement of specification

2 Vertical refinement (2007)

- More powerful than horizontal refinement
- Reinterprets specification
- Formalises **client's design decision**

Horizontal + Vertical

- 1 Horizontal refinement and known examples (2003)
- 2 Vertical refinement (2007)

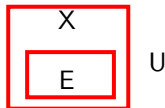


Refinement in General

- The concrete entity C is a refinement of an abstract entity A when no user of A could observe if they were given C in place of A .
- To formalise this we use:
 - Entities A and C from some set \mathbb{E}
 - Contexts $x \in \Xi$ in which we place the entities $[A]_x \in \mathbb{E}$
 - A user formalised by an observation function $O : \mathbb{E} \rightarrow \mathbb{O}$
 - **Two interfaces**
- \mathbb{O} is the set of complete traces

Refinement in General

- The concrete entity C is a refinement of an abstract entity A when no user of A could observe if they were given C in place of A .
- To formalise this we use:
 - Entities A and C from some set \mathbb{E}
 - Contexts $x \in \Xi$ in which we place the entities $[A]_x \in \mathbb{E}$
 - A user formalised by an observation function $O : \mathbb{E} \rightarrow \mathbb{O}$
 - **Two interfaces**



- \mathbb{O} is the set of complete traces

Refinement in General



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

- The denotational semantics of entities is a relation:

$$\llbracket A \rrbracket_{\Xi, O} \triangleq \{(x, o) \mid x \in \Xi, o \in O([A]_x)\}$$

- Refinement is subset or implication in a logical theory:
A theory T is pair $(\mathbb{E}_T, \sqsubseteq_T)$
- For D a deterministic entity $\llbracket D \rrbracket_{\Xi, O}$ is a function
- General Refinement is made concrete by fixing:

$$\mathbb{E}, \Xi \text{ and } O$$

Refinement in General



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

- The denotational semantics of entities is a relation:

$$\llbracket A \rrbracket_{\Xi, O} \triangleq \{(x, o) \mid x \in \Xi, o \in O([A]_x)\}$$

- Refinement is subset or implication in a logical theory:
A theory T is pair $(\mathbb{E}_T, \sqsubseteq_T)$
- For D a deterministic entity $\llbracket D \rrbracket_{\Xi, O}$ is a function
- General Refinement is made concrete by fixing:

$$\mathbb{E}, \Xi \text{ and } O$$

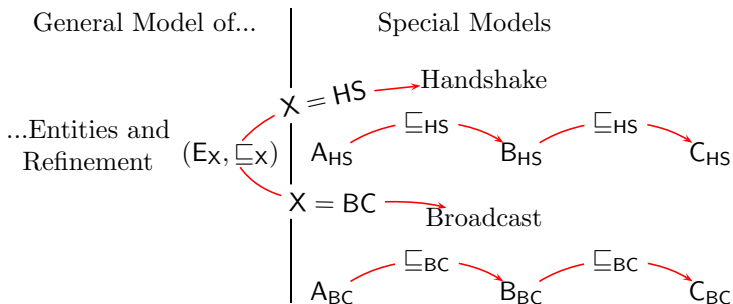
Interface types

- **Transactional interface**
 - Observation occurs at initialisation and at finalisation if termination is successful.
 - If termination is unsuccessful then all that can be “observed” is that the entity fails to terminate.
 - Example - a method of an object
- **Interactive interface**
 - Observation can occur at many points throughout the execution.
 - Observations can be made prior to termination and even prior to non-termination.
 - An example of an interactive entity is a coffee machine.

Horizontal + Vertical

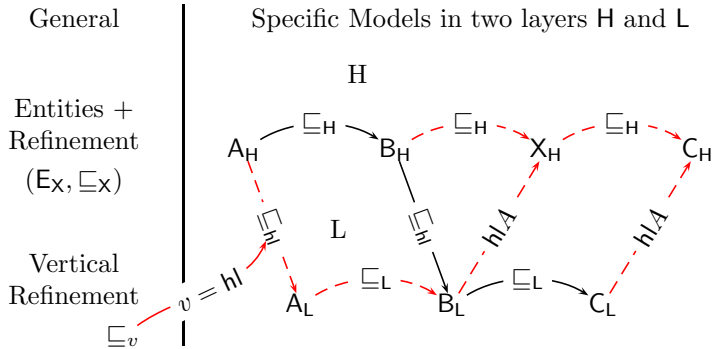
1 Horizontal refinement (2003)

2 Vertical refinement (2007)



Horizontal + Vertical

- 1 Horizontal refinement (2003)
- 2 **Vertical refinement (2007)**



Theory morphism

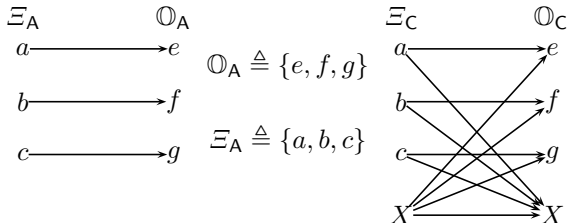
- Different kinds of entities have different theories
- Galois connections interpret one theory in another and vice versa
- A Galois connection is a pair of functions: $\llbracket _ \rrbracket_V^{HL}$ one from the high- to low-level theory the other vA^{HL} from the low- to high-level theory.

$$\llbracket X_H \rrbracket_V^{HL} \sqsubseteq_{\Xi_L, O_L} Y_L \Leftrightarrow X_H \sqsubseteq_{\Xi_H, O_H} vA^{HL}(Y_L)$$

- We refer to our Galois connections as vertical refinements as they provide a **guarantee** of the behaviour of the high-level entities in term of the behaviour of the low-level entity and vice versa.

Subset morphism

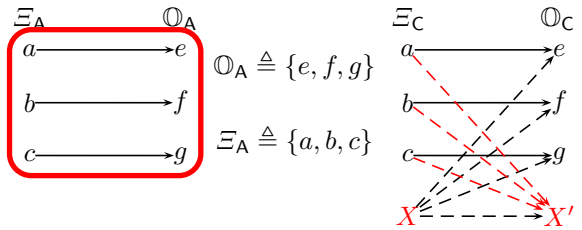
- Subset theories are related by a Galois connection
- If $\Xi_A \times \mathbb{O}_A \subseteq \Xi_C \times \mathbb{O}_C$ then the embedding and projection functions are a Galois connection.
- Example $A \sqsubseteq_{sub} C$



- Undefined outside of frame
- X is an unconsidered context and X' an unconsidered observation.

Subset morphism

- Subset theories are related by a Galois connection
- If $\Xi_A \times \mathbb{O}_A \subseteq \Xi_C \times \mathbb{O}_C$ then the embedding and projection functions are a Galois connection.
- Example $A \sqsubseteq_{sub} C$



- Undefined outside of **frame**
- **X** is an unconsidered context and **X'** an unconsidered observation.

Insights

- Singleton failure semantics is different from data refinement because each uses a different style of context user interface interface.
- Divergence only masks what you want to see when the context user interface is transactional not when it is interactive.
- Dropping the stimulus-response nature of event synchronisation changes how determinism is modelled

Summary

- General refinement factors out a common theory
- Special concrete examples include
 - Handshake processes
 - Broadcast processes
 - ADT
 - Operations
- Vertical refinement allows
 - layered development
 - reinterpretation
 - undefined outside of frame
 - some retrenchments

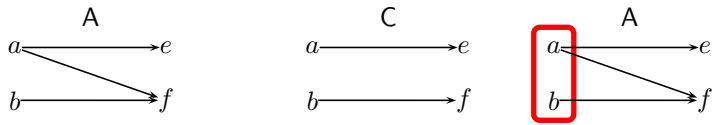
Refinement is?

- A **strict** view of reduction of non-determinism.
- Undefined outside of **domain**
The abstract specification does not consider behaviour outside of the domain **c**
- Undefined outside of **frame**
In addition the abstract specification does not consider when new observations **g** can be made.
- Note the introduction of non-determinism



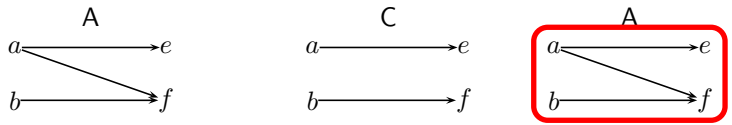
Refinement is?

- A **strict** view of reduction of non-determinism.
- Undefined outside of **domain**
The abstract specification does not consider behaviour outside of the domain **c**
- Undefined outside of **frame**
In addition the abstract specification does not consider when new observations **g** can be made.
- Note the introduction of non-determinism



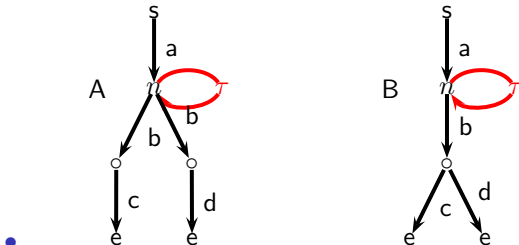
Refinement is?

- A **strict** view of reduction of non-determinism.
- Undefined outside of **domain**
The abstract specification does not consider behaviour outside of the domain **c**
- Undefined outside of **frame**
In addition the abstract specification does not consider when new observations **g** can be made.
- Note the introduction of non-determinism



Divergence

- The Chaotic interpretation of divergence
“typically masks much L behaviour we should really
want to see” Roscoe



- I-I** tests are less masking semantics than **I-T** tests
- $A \sqsubseteq_{II} B$ and $B \not\sqsubseteq_{II} A$ but $A =_{IT} B$
- Consider the test **abc**

First specify correct behaviour later specify errors

$$\text{Set}_C = \text{Set}_A + \text{errors}$$

Correct Behaviour

Set_A

$$\text{State}_A = s : \mathbb{PN}$$

$$\text{Put}_A(n?)$$

If $n? \notin s$ then

$$s' = s \cup \{n?\}$$

$$\text{Get}_A(n!)$$

If $n! \in s$ then

$$s' = s \setminus \{n!\}$$

$$\text{State}_C = t : \mathbb{PN} \cup \{X\}$$

$$\text{Put}_C(n?)$$

If $t \neq X \wedge \#t < 3 \wedge n? \notin s$ then

$$s' = s \cup \{n?\} \text{ elseif}$$

$t \neq X \wedge n? \notin s \wedge \#t = 3$ then

$$t = X$$

$$\text{Get}_C(n!)$$

If $t \neq X \wedge n! \in s$ then

$$s' = s \setminus \{n!\}$$

$$\text{Reset}_C$$

If $t = X$ then $t' = \emptyset$

First specify correct behaviour later specify errors

$$\text{Set}_C = \text{Set}_A + \text{errors}$$

Correct Behaviour

Set_A

$$\text{State}_A = s : \mathbb{PN}$$

$\text{Put}_A(n?)$

If $n? \notin s$ then

$$s' = s \cup \{n?\}$$

$\text{Get}_A(n!)$

If $n! \in s$ then

$$s' = s \setminus \{n!\}$$

$$\text{State}_C = t : \mathbb{PN} \cup \{X\}$$

$\text{Put}_C(n?)$

If $t \neq X \wedge \#t < 3 \wedge n? \notin s$ then

$$s' = s \cup \{n?\} \text{ elseif}$$

$t \neq X \wedge n? \notin s \wedge \#t = 3$ then

$$t = X$$

$\text{Get}_C(n!)$

If $t \neq X \wedge n! \in s$ then

$$s' = s \setminus \{n!\}$$

Reset_C

If $t = X$ then $t' = \emptyset$

First specify correct behaviour later **specify errors**

$$\text{Set}_C = \text{Set}_A + \text{errors}$$

Correct Behaviour

Set_A

$$\text{State}_A = s : \mathbb{PN}$$

$\text{Put}_A(n?)$

If $n? \notin s$ then

$$s' = s \cup \{n?\}$$

$\text{Get}_A(n!)$

If $n! \in s$ then

$$s' = s \setminus \{n!\}$$

$$\text{State}_C = t : \mathbb{PN} \cup \{X\}$$

$\text{Put}_C(n?)$

If $t \neq X \wedge \#t < 3 \wedge n? \notin s$ then

$s' = s \cup \{n?\}$ **elseif**

$t \neq X \wedge n? \notin s \wedge \#t = 3$ then

$t = X$

$\text{Get}_C(n!)$

If $t \neq X \wedge n! \in s$ then

$s' = s \setminus \{n!\}$

Reset_C

If $t = X$ then $t' = \emptyset$

Is $\text{Set}_A \sqsubseteq \text{Set}_C$?

- In most formalisms NO!
 - In most formalisms Set_A , not Set_C , allows Put^∞
- If we want to develop software like this how are we interpreting the specification?
- What is Set_A formally specifying?
 - We interpret Set_A as only guaranteeing behaviour when no errors occur. Thus when:
 1. state remains in State_A and
 2. only operations in Set_A are called/observed.
- Flexible refinement will formalise such development steps

Is $\text{Set}_A \sqsubseteq \text{Set}_C$?

- In most formalisms NO!
 - In most formalisms Set_A , not Set_C , allows Put^∞
- If we want to develop software like this how are we interpreting the specification?
- What is Set_A formally specifying?
 - We interpret Set_A as only guaranteeing behaviour when no errors occur. Thus when:
 1. state remains in State_A and
 2. only operations in Set_A are called/observed.
- Flexible refinement will formalise such development steps

Is $\text{Set}_A \sqsubseteq \text{Set}_C$?

- In most formalisms NO!
 - In most formalisms Set_A , not Set_C , allows Put^∞
- If we want to develop software like this how are we interpreting the specification?
- What is Set_A formally specifying?
 - We interpret Set_A as only guaranteeing behaviour when no errors occur. Thus when:
 1. state remains in State_A and
 2. only operations in Set_A are called/observed.
- Flexible refinement will formalise such development steps

Is $\text{Set}_A \sqsubseteq \text{Set}_C$?

- In most formalisms NO!
 - In most formalisms Set_A , not Set_C , allows Put^∞
- If we want to develop software like this how are we interpreting the specification?
- What is Set_A formally specifying?
 - We interpret Set_A as only guaranteeing behaviour when no errors occur. Thus when:
 1. state remains in State_A and
 2. only operations in Set_A are called/observed.
- Flexible refinement will formalise such development steps

From Set_A to Set_C in three steps

- Step one: state-based subset morphism $\sqsubseteq_{sub}^{\{X\}}$
- Step two: general refinement
- Step three: event-based subset morphism $\sqsubseteq_{sub}^{\{\text{Reset}\}}$

$$\text{Set}_A \sqsubseteq_{sub}^{\{X\}} \text{Set}_X \sqsubseteq \text{Set}_B \sqsubseteq_{sub}^{\{\text{Reset}\}} \text{Set}_C$$

Step 1 from Set_A to Set_X $\{X\}$ is outside the frame

Set_A

Set_X

$\text{State}_A = s : \mathbb{PN}$

$\text{Put}_A(n?)$

If $n? \notin s$ then

$s' = s \cup \{n?\}$

$\text{Get}_A(n!)$

If $n! \in s$ then

$s' = s \setminus \{n!\}$

$\text{State}_X = s : \mathbb{PN} \cup \{X\}$

$\text{Put}_X(n?)$

If $s \neq X \wedge n? \notin s$ then

$s' = s \cup \{n?\} \vee s' = X$

$\text{Get}_X(n!)$

If $s \neq X \wedge n! \in s$ then

$s' = s \setminus \{n!\} \vee s' = X$

Motivation

Tests and
Refinement

Horizontal + Vertical

Parametrised
Horizontal

Interface types
Vertical

Final

Observations
Summary

Comparison
with literature

Subset Morphism
and Ref
Divergence

Example

Step 2 from Set_X to Set_B General Refinement

Set_B

Set_X

$\text{State}_X = s : \mathbb{PN} \cup \{X\}$

$\text{Put}_X(n?)$

If $s \neq X \wedge n? \notin s$ then

$s' = s \cup \{n?\} \vee s' = X$

$\text{Get}_X(n!)$

If $s \neq X \wedge n! \in s$ then

$s' = s \setminus \{n!\} \vee s' = X$

$\text{State}_B = t : \mathbb{PN} \cup \{X\}$

$\text{Put}_B(n?)$

If $t \neq X \wedge n? \notin t \wedge \#t < 3$ then

$t' = t \cup \{n?\}$ **elseif**

$t \neq X \wedge n? \notin t \wedge \#t = 3$ then

$t' = X$

$\text{Get}_B(n!)$

If $t \neq X \wedge n! \in t$ then

$t' = t \setminus \{n!\}$

Motivation

Tests and
Refinement

Horizontal + Vertical

Parametrised

Horizontal

Interface types

Vertical

Final

Observations

Summary

Comparison
with literature

Subset Morphism

and Ref

Divergence

Example

Step 3 from Set_B to Set_C **Reset is outside the frame**

$$\text{Set}_C = \text{Set}_B + \text{Reset}_C$$

$$\text{State}_C = t : \mathbb{PN} \cup \{X\}$$

$$\text{Put}_C(n?)$$

If $t \neq X \wedge \#t < 3 \wedge n? \notin s$ then

$$s' = s \cup \{n?\} \text{ elseif}$$

$t \neq X \wedge n? \notin s \wedge \#t = 3$ then

$$t = X$$

$$\text{Get}_C(n!)$$

If $t \neq X \wedge n! \in s$ then

$$s' = s \setminus \{n!\}$$

Reset_C

If $t = X$ then $t' = \emptyset$