

Build Your Own Refinement

S. Reeves D. Streader

Department of Computer Science
University of Waikato

Copenhagen ITU 2009

Outline

- 1 Overview
- 2 Idealised Refinement
- 3 Known Refinement Theories
- 4 Abstract Refinement
BYO Refinement
- 5 Doing better with Galois
- 6 Final
Observations
- 7 Examples
Operations
Protocol stacks
Divergence
Z+B

Overview

Overview

Idealised Refinement

Known Refinement- Theories

Abstract Refinement BYO Refinement

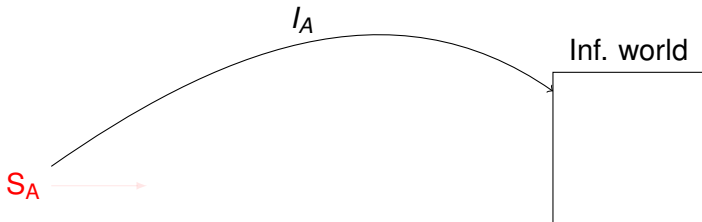
Doing better with Galois

Final Observations

Examples Operations Protocol stacks Divergence Z+B

- Refinement is a step in the development of code from a specification
- Why build your own?
 - What is refinement?
 - What is in the literature?
 - How do we Build Your Own?
 - What can we do even better?
- We introduce some simple ideas
 - 1 Abstract refinement specialising to known theories
 - 2 BYO testing semantics
 - 3 Galois connections as refinement steps

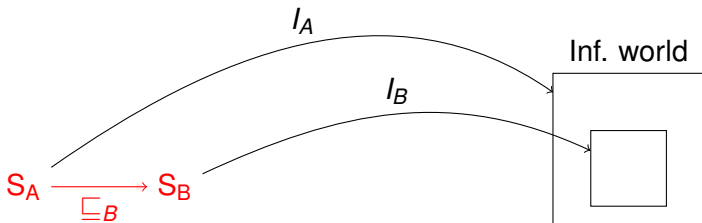
Stepwise Design by Refinement



- Step 1 define specification and interpretation

- Step 2 define $S_A, I_A \sqsubseteq_B S_B$ and I_B
- Step 3 $\sqsubseteq_C S_C$ and I_C
- Refinement = pre-order, \sqsubseteq + guarantee
- Build Your own Refinement Specifications, S_A, S_B and S_C in different theories!

Stepwise Design by Refinement



- Step 1 define specification and interpretation

- Step 2 define S_A, I_A and S_B, I_B

- Step 3 \sqsubseteq_C, S_C and I_C

- Refinement = pre-order, \sqsubseteq + guarantee
- Build Your own Refinement Specifications, S_A, S_B and S_C in different theories!

Stepwise Design by Refinement

Reeves, Streader

Overview

Idealised Refinement

Known Refinement Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

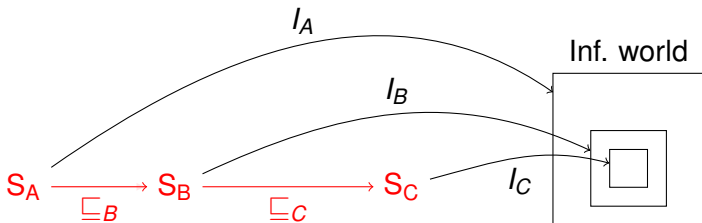
Examples

Operations

Protocol stacks

Divergence

Z+B



- Step 1 define specification and interpretation

- Step 2 define S_A, I_A
 \sqsubseteq_B S_B and I_B

- Step 3 \sqsubseteq_C S_C and I_C

- Refinement = pre-order, \sqsubseteq + guarantee
- Build Your own Refinement Specifications, S_A, S_B and S_C in different theories!

Stepwise Design by Refinement

Reeves, Streader

Overview

Idealised Refinement

Known Refinement Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

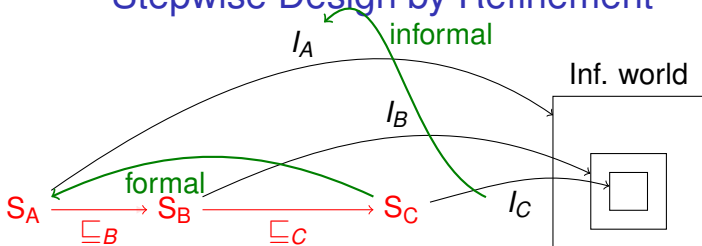
Examples

Operations

Protocol stacks

Divergence

Z+B



- Step 1 define specification and interpretation

S_A, I_A

- Step 2 define $\sqsubseteq_B S_B$ and I_B

- Step 3 $\sqsubseteq_C S_C$ and I_C

- Refinement = pre-order, \sqsubseteq + **guarantee**

- Build Your own Refinement

Specifications, S_A , S_B and S_C in different theories!

Stepwise Design by Refinement

Reeves, Streader

Overview

Idealised Refinement

Known Refinement Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

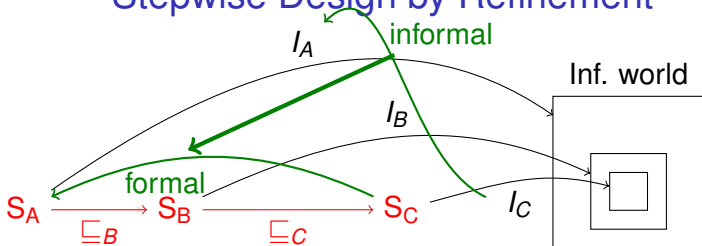
Examples

Operations

Protocol stacks

Divergence

Z+B



- Step 1 define specification and interpretation

S_A, I_A

- Step 2 define $\sqsubseteq_B S_B$ and I_B

$\sqsubseteq_C S_C$ and I_C

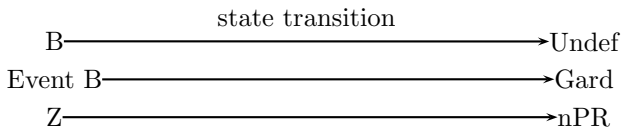
- Step 3

- Refinement = pre-order, \sqsubseteq + **guarantee**

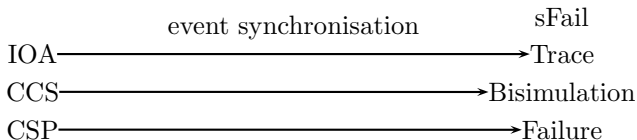
- Build Your own Refinement

Specifications, S_A , S_B and S_C in different theories!

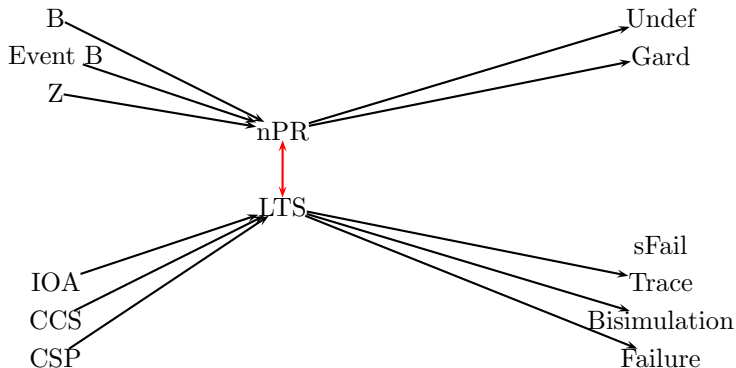
Isolated Refinements Theories



Each theory has fixed semantics and interpretation!



Common Semantics



Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

Examples

Operations

Protocol stacks

Divergence

Z+B

Parameterised Theory

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

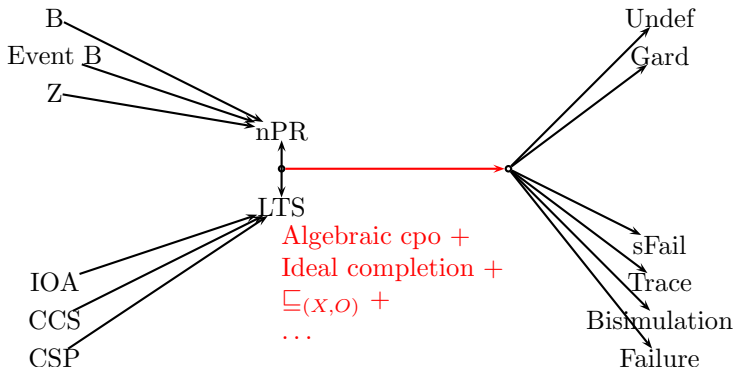
Examples

Operations

Protocol stacks

Divergence

Z+B



Unified Theory of Testing

- 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**

E	X	U
Motor	Car	Driver
Motor	Car	Passenger
ADT	Prog	User
• Method	ADT	Prog

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

Examples

Operations

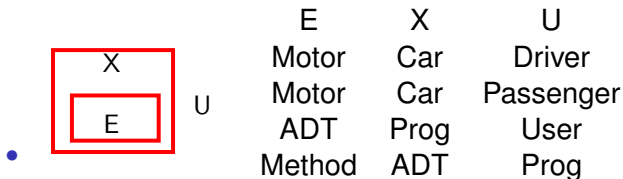
Protocol stacks

Divergence

Z+B

Unified Theory of Testing

- 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**



Three semantics, refinements

- specification as a contract: if $X \in \Xi$ and only \mathbb{O} observed then $O([E]_X)$ defines what can be observed
 - refinement as satisfies contract (subset of observations)

$$A \sqsubseteq_{\Xi, \mathbb{O}} C \triangleq \forall X \in \Xi. O([C]_X) \subseteq O([A]_X)$$

- specification as proposition (relation)

$$A_{\Xi, \mathbb{O}}(X, o) \triangleq X \in \Xi \wedge o \in O([A]_X)$$

- refinement as implication (subset)

$$A \sqsubseteq_{\rightarrow, \Xi, \mathbb{O}} C \triangleq C_{\Xi, \mathbb{O}} \rightarrow A_{\Xi, \mathbb{O}}$$

- We refer to $\Xi \times \mathbb{O}$ as the *Frame* of the theory

Three semantics, refinements

Overview

Idealised
Refinement

Known
Refinement-
Theories

Abstract
Refinement

BYO Refinement

Doing better
with Galois

Final

Observations

Examples

Operations

Protocol stacks

Divergence

Z+B

- 4 • specification as set of implementations

$$\llbracket A \rrbracket_{I, \Xi, O} \triangleq \{ I \mid \text{Det}_{\Xi, O}(I) \wedge I_{\Xi, O} \rightarrow A_{\Xi, O} \}$$

- refinement as subset

$$A \sqsubseteq_{I, \Xi, O} C \triangleq \llbracket C \rrbracket_{I, \Xi, O} \subseteq \llbracket A \rrbracket_{I, \Xi, O}$$

semantics can be built from refinement

- The three refinement relations are equivalent

$$A \sqsubseteq_{I, \Xi, O} C \quad \Leftrightarrow \quad A \sqsubseteq_{\rightarrow, \Xi, O} C \quad \Leftrightarrow \quad A \sqsubseteq_{\Xi, O} C$$

Interface types

- **Transactional interface**
 - Observation occurs at initialisation and at finalisation if termination is successful.
 - Examples - a method of an object and passenger in car
- **Interactive interface**
 - Observation can occur at many points throughout the execution
 - Observations can be made prior to termination and even prior to non-termination
 - Examples - a coffee machine and the driver of a car

Build Your Own Refinement

- Which formal model corresponds closely to engineering intuition "this satisfies the specification"?
- From an informal notion of test
 - 1 How is the entity to be used?
 - 2 What can be observed in an execution?
- Build a formal testing semantics
 - 1 The entities are we considering - \mathbb{E}
 - 2 The contexts in which the entity can be placed - Ξ
 - 3 The set of observations we can make - \mathbb{O}
 - 4 The observation mapping $O([_]_)$

Next - Doing Better

Build Your Own Refinement

Overview

Idealised
Refinement

Known
Refinement-
Theories

Abstract
Refinement

BYO Refinement

Doing better
with Galois

Final

Observations

Examples

Operations

Protocol stacks

Divergence

Z+B

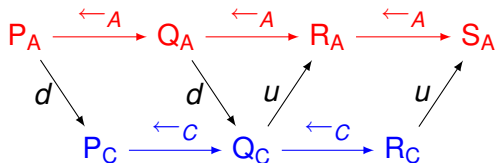
- Which formal model corresponds closely to engineering intuition "this satisfies the specification"?
- From an informal notion of test
 - 1 How is the entity to be used?
 - 2 What can be observed in an execution?
- Build a formal testing semantics
 - 1 The entities are we considering - \mathbb{E}
 - 2 The contexts in which the entity can be placed - Ξ
 - 3 The set of observations we can make - \mathbb{O}
 - 4 The observation mapping $O([_]_)$

•

Next - Doing Better

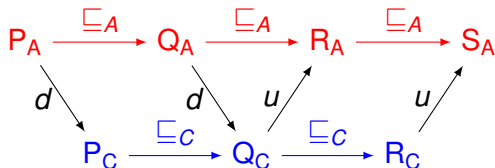
Galois Connections

- A weak form of isomorphism between logical theories
- defined by a pair of mappings (d, u)



Galois Connections

- A weak form of isomorphism between logical theories
- defined by a pair of mappings (d, u)



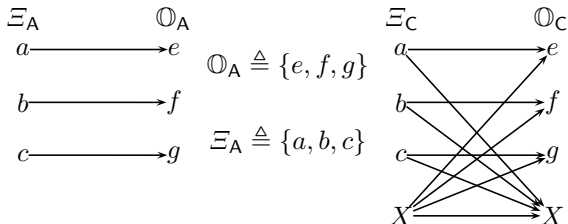
- Galois Steps UTT+GC+UTT
 - 1 both preserves and reflects refinement
 - 2 very strict step - may be regarded as a refinement.

$$d(P_A) \sqsubseteq_C R_C$$

$$P_A \sqsubseteq_A u(R_C)$$

Subset morphism

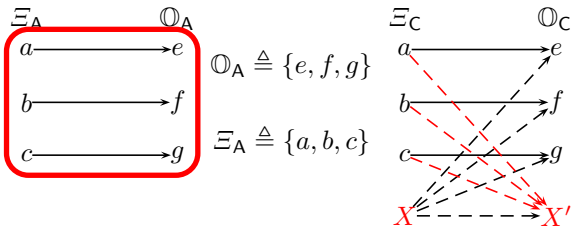
- Subset theories are related by a Galois connection
- If $\Xi_A \times \mathbb{O}_A \subseteq \Xi_C \times \mathbb{O}_C$ then the embedding and projection functions are a Galois connection
- Example $A \sqsubseteq_{sub} C$



- Silent 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 $A \sqsubseteq_{sub} C$



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

Summary

Overview

Idealised
Refinement

Known
Refinement-
Theories

Abstract
Refinement
BYO Refinement

Doing better
with Galois

Final
Observations

Examples
Operations
Protocol stacks
Divergence
Z+B

- UTT factors out a common theory
- UTT as Build Your Own Refinement
- Example refinement theories
 - Handshake processes
 - Broadcast processes
 - ADT
 - Operations
- Better by Galois
 - layered development
 - reinterpretation
 - silent outside of frame
 - some retrenchments

Observations

Overview

Idealised
Refinement

Known
Refinement-
Theories

Abstract
Refinement
BYO Refinement

Doing better
with Galois

Final
Observations

Examples
Operations
Protocol stacks
Divergence
Z+B

- Singleton failure semantics is different from data refinement because each uses a different style of context/user 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

Formality, Flexibility or Both

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

Examples

Operations

Protocol stacks

Divergence

Z+B

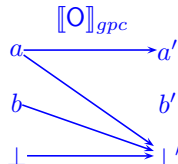
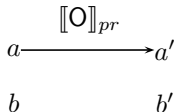
- Formal semantics fix part of the meaning
- More formality; Less flexibility
- Partial relations have many valid interpretations

$$\begin{array}{ccc} & \llbracket O \rrbracket_{pr} & \\ a & \longrightarrow & a' \\ b & & b' \end{array}$$

- From state a O terminates and terminates in a'
- From state $a +$ undefined outside of precondition O terminates and terminates in a'

Formality, Flexibility or Both

- Formal semantics fixe part of the meaning
- More formality; Less flexibility
- Partial relations have many valid interpretations



- From state a If O terminates it terminates in a'
- From state $a +$ undefined outside of precondition O terminates and terminates in a'

Formality, Flexibility or Both

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

Examples

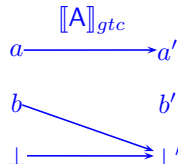
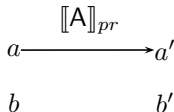
Operations

Protocol stacks

Divergence

Z+B

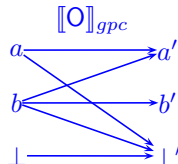
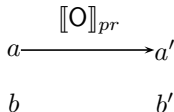
- Formal semantics fix part of the meaning
- More formality; Less flexibility
- Partial relations have many valid interpretations



- From state a \perp terminates and terminates in a'
- From state $a + \text{undefined}$ outside of precondition \perp terminates and terminates in a'

Formality, Flexibility or Both

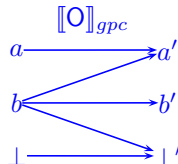
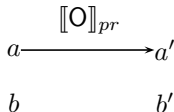
- Formal semantics fix part of the meaning
- More formality; Less flexibility
- Partial relations have many valid interpretations



- From state a O terminates and terminates in a'
- From state a + undefined outside of precondition
If O terminates it terminates in a'

Formality, Flexibility or Both

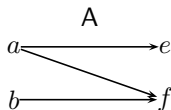
- Formal semantics fixe part of the meaning
- More formality; Less flexibility
- Partial relations have many valid interpretations



- From state a O terminates and terminates in a'
- From state a + undefined outside of precondition O terminates and terminates in a'

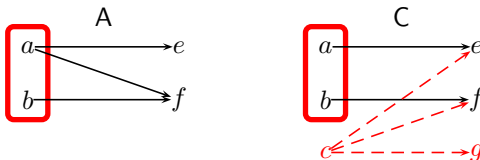
What is Refinement?

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



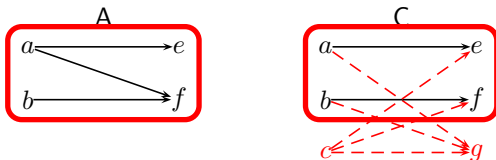
What is Refinement?

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

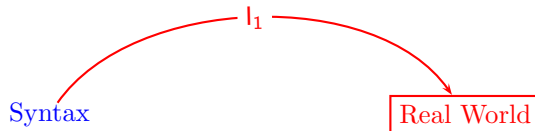


What is Refinement?

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



From formal Specification to informal world



- To be of **use** a formal statement must be **interpreted**
- Any set **Act** and **binary operator** is a valid interpretation (in the range of I_1)
- Semantics are designed to be closer to the real world

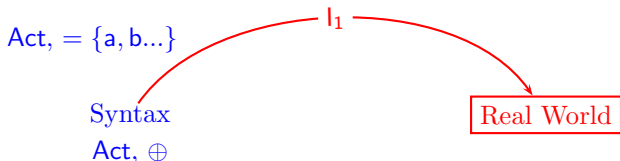
$$[[a \oplus b]] \triangleq [[a]][\oplus][[b]] \quad [\oplus] \triangleq \cup \quad \forall a \in Act. [[a]] \subseteq S \times S$$

- The meaning given to a term is part **formal** part **informal**
- Semantics restricts the interpretations of terms:

$I_2[[t]]$ is a **subset** of $I_1 t$

$$[[a \oplus a]] = [[a]]$$

From formal Specification to informal world



- To be of **use** a formal statement must be **interpreted**
- Any set **Act** and **binary operator** is a valid interpretation (in the range of I_1)

- Semantics are designed to be closer to the real world

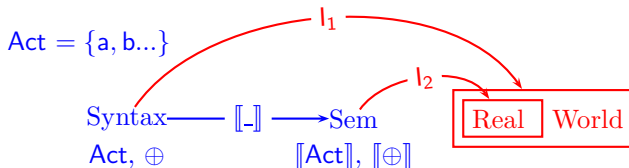
$$\llbracket a \oplus b \rrbracket \triangleq \llbracket a \rrbracket \llbracket \oplus \rrbracket \llbracket b \rrbracket \quad \llbracket \oplus \rrbracket \triangleq \cup \quad \forall a \in \text{Act}. \llbracket a \rrbracket \subseteq S \times S$$

- The meaning given to a term is part **formal** part **informal**
- Semantics restricts the interpretations of terms:

$\llbracket t \rrbracket$ is a **subset** of $I_1 t$

$$\llbracket a \oplus a \rrbracket = \llbracket a \rrbracket$$

From formal Specification to informal world



- To be of **use** a formal statement must be **interpreted**
- Any set **Act** and **binary operator** is a valid interpretation (in the range of I_1)
- Semantics are designed to be closer to the real world

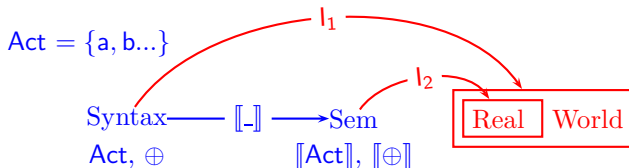
$$[[a \oplus b]] \triangleq [[a]][\oplus][[b]] \quad [\oplus] \triangleq \cup \quad \forall a \in \text{Act}. [[a]] \subseteq \mathcal{S} \times \mathcal{S}$$

- The meaning given to a term is part **formal** part **informal**
- Semantics restricts the interpretations of terms:

$I_2[t]$ is a **subset** of $I_1 t$

$$[[a \oplus a]] = [[a]]$$

From formal Specification to informal world



- To be of **use** a formal statement must be **interpreted**
- Any set **Act** and **binary operator** is a valid interpretation (in the range of I_1)
- Semantics are designed to be closer to the real world

$$[[a \oplus b]] \triangleq [[a]][\oplus][[b]] \quad [\oplus] \triangleq \cup \quad \forall a \in \text{Act}. [[a]] \subseteq \mathcal{S} \times \mathcal{S}$$

- The meaning given to a term is part **formal** part **informal**
- Semantics restricts the interpretations of terms:

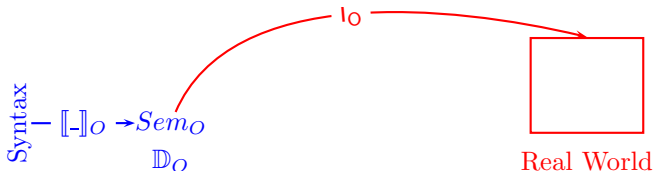
$I_2[[t]]$ is a **subset** of $I_1 t$

$$[[a \oplus a]] = [[a]]$$

Stepwise development

- Step One define the Syntax
- Step Two define a Semantics, \mathbb{D}_O and mapping $\llbracket - \rrbracket_O$
 - $\llbracket - \rrbracket_O$ defines only part of the semantic story
 - Closer to our intuitions
 - Syntax is open to fewer interpretations
- Step Three repeat step Two
- Examples Z & B theory + practice

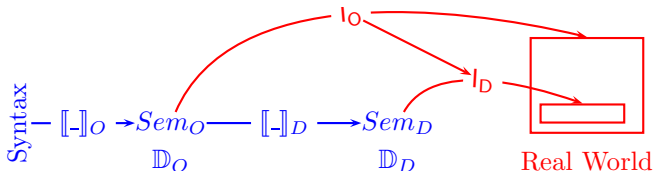
\mathbb{D}_O	partial relations	Correct
\mathbb{D}_D	lifted total relations	Correct + Error



Stepwise development

- Step One define the Syntax
- Step Two define a Semantics, \mathbb{D}_O and mapping $\llbracket - \rrbracket_O$
 - $\llbracket - \rrbracket_O$ defines only part of the semantic story
 - Closer to our intuitions
 - Syntax is open to fewer interpretations
- Step Three repeat step Two
- Examples Z & B theory + practice

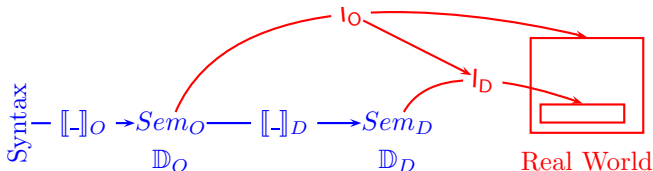
\mathbb{D}_O	partial relations	Correct
\mathbb{D}_D	lifted total relations	Correct + Error



Stepwise development

- Step One define the Syntax
- Step Two define a Semantics, \mathbb{D}_O and mapping $\llbracket - \rrbracket_O$
 - $\llbracket - \rrbracket_O$ defines only part of the semantic story
 - Closer to our intuitions
 - Syntax is open to fewer interpretations
- Step Three repeat step Two
- Examples Z & B theory + practice

\mathbb{D}_O	partial relations	Correct
\mathbb{D}_D	lifted total relations	Correct + Error



Protocol stacks

Overview

Idealised
Refinement

Known
Refinement-
Theories

Abstract
Refinement
BYO Refinement

Doing better
with Galois

Final
Observations

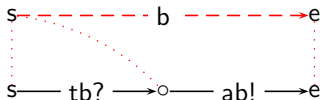
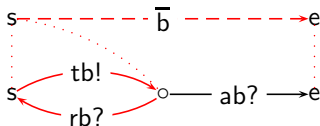
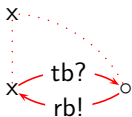
Examples
Operations
Protocol stacks
Divergence
Z+B

- Each layer in a protocol stack is implemented on the layer beneath
- Both valid entities and contexts are defined
- A broadcast layer has unblockable send events
- This can be formalised in two ways
 - BC_{CSP} with processes that always listen and CSP style parallel composition
 - BC_V with a definition of parallel composition to formalise non-blocking send
- There is a Galois connection between the two
- but not between BC_{CSP} and CSP style processes

On a Broadcast Layer

- Can we implement processes with handshake style events?
- Using broadcast send $b!$ and broadcast listen $b?$

$b \notin Ready(x)$



Active actions and determinism

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement

BYO Refinement

Doing better with Galois

Final

Observations

Examples

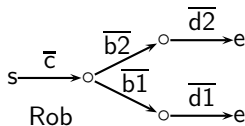
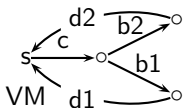
Operations

Protocol stacks

Divergence

Z+B

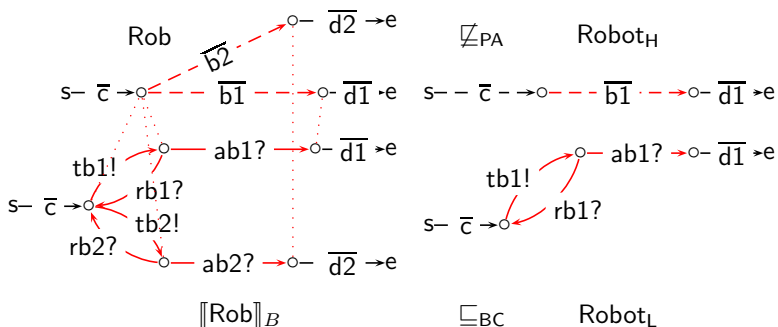
- Are active actions the same as passive actions?



- Is Rob deterministic?

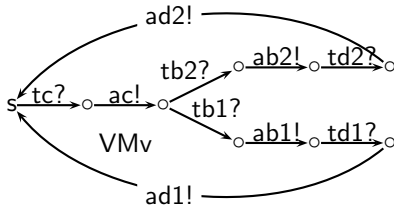
Robot Problems

- We have a problem with branching active actions



Mixing action types

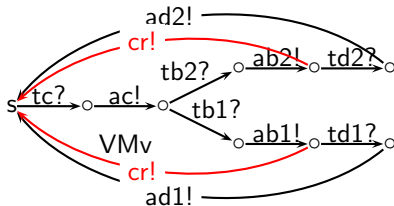
- Vending machine on broadcast layer



- Error action **coin return** can not be blocked!
- Simplify
- Refiine for two cups

Mixing action types

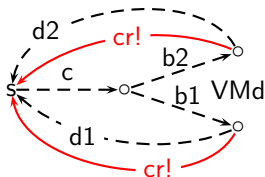
- Vending machine on broadcast layer



- Error action **coin return** can not be blocked!
- Simplify
- Refiine for two cups

Mixing action types

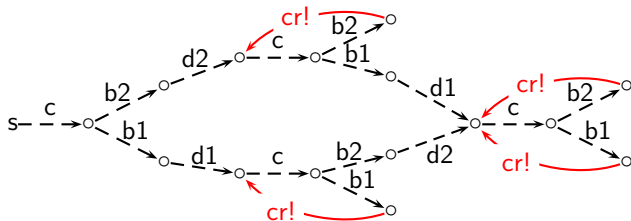
- Vending machine on broadcast layer



- Error action **coin return** can not be blocked!
- Simplify
- Refiine for two cups

Mixing action types

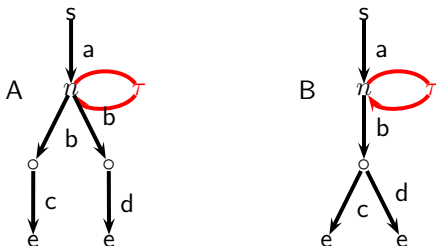
- Vending machine on broadcast layer



- Error action **coin return** can not be blocked!
- Simplify
- Refiine for two cups

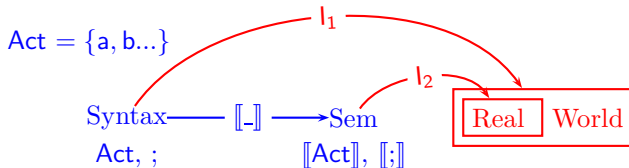
Divergence

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



- I-I** tests are a 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**

B = Z + Formal methodology.



- Both Z and (Event) B use similar semantics But!

$$[a;b] \triangleq [a][;][b] \quad [;] \triangleq \circ \quad \forall a \in \mathbf{Act}. [a] \subseteq \mathcal{S} \times \mathcal{S}$$

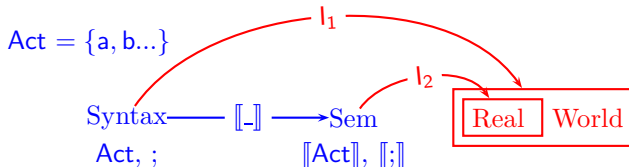
- Engineers must take account of the informal

$$I_1(a;b) \text{ is equal to } I_1(a)I_1(;)I_1(b)$$

$I_1(;)$ is the sequential composition of operations

- Z Spivey care needed with sequential composition.

B = Z + Formal methodology.



- Both Z and (Event) B use similar semantics But!

$$[a;b] \triangleq [a][;][b] \quad [;] \triangleq \circ \quad \forall a \in Act. [a] \subseteq S \times S$$

- Engineers must take account of the **informal**

$I_1(a;b)$ is equal to $I_1(a)I_1(;)I_1(b)$

$I_1(;)$ is the sequential composition of operations

- Z Spivey **care needed** with sequential composition.

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

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

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

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

$t \neq X \wedge n? \notin s \wedge \#t = 3$ 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 $t' = \emptyset$

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

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement
BYO Refinement

Doing better with Galois

Final Observations

Examples

Operations
Protocol stacks
Divergence

Z+B

- 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

Overview

Idealised
Refinement

Known
Refinement-
Theories

Abstract
Refinement
BYO Refinement

Doing better
with Galois

Final
Observations

Examples
Operations
Protocol stacks
Divergence
Z+B

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

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

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

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

Step 2 from Set_X to Set_B

General Refinement

Overview

Idealised Refinement

Known Refinement-Theories

Abstract Refinement
BYO Refinement

Doing better with Galois

Final Observations

Examples

Operations
Protocol stacks
Divergence
Z+B

Set_X

$State_X = s : \mathbb{PN} \cup \{X\}$

$Put_X(n?)$

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

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

$Get_X(n!)$

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

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

Set_B

$State_B = t : \mathbb{PN} \cup \{X\}$

$Put_B(n?)$

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

$t' = X$

$Get_B(n!)$

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

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

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

$s' = s \cup \{n?\}$ 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 $t' = \emptyset$