


# Free theorems for refinement of behavioral specifications

Marius Petria

University of Edinburgh <sup>1</sup>

Sinaia, March 2008

---

<sup>1</sup>on leave from the Institute of Mathematics of the Romanian Academy, Bucharest  

# The goal of this work

- Ease proofs of behavioral refinements
- Stability, a second order axiom used to capture abstraction barriers, enforces properties not expressible by the behavioral specifications
- Intuition: Analogy between parametricity and stability
- Example: implementation of *replace* for *Containers*

# Behavioral specifications

## Definition (Bidoit and Hennicker's Observational Logic)

observational signature  $(S, F, H, O)$

observational equality  $\approx_{A \subseteq A} A \times A$  - indistinguishability w.r.t.  $O$

observational algebras  $\approx_A$  is a  $F$ -congruence

satisfaction relation  $A \models l = r$  iff  $A_l \approx_A A_r$

## CONTAINER

**sorts** *Container, Nat, Bool*

**ops**

*empty* : *Container*

*insert, remove* : *Nat, Container*  $\rightarrow$  *Container*

*isIn* : *Nat, Container*  $\rightarrow$  *Bool*

**observers** (*isIn*, 2)

**axioms** *remove*(*x*, *insert*(*x*, *c*)) = *remove*(*x*, *c*)

# Stability

## Refinement

Refinement  $SP_{\text{abstract}} \rightsquigarrow SP_{\text{concrete}}$

Constructors  $k : \text{Mod}[SP_{\text{concrete}}] \rightarrow \text{Mod}[SP_{\text{abstract}}]$

Schoett '86 - Stability

Stable constructors take related algebras to related algebras

Reynolds '83 - Parametricity

Parametric elements take related arguments to related results

Definability Respecting abstraction barriers (stability or parametricity) implies definability

# Local constructors in global contexts

A technique for reusing the constructors [Bidoit, Sannella, Tarlecki '06]:

## Constructor translation

$$\begin{array}{ccc} SP_G & \xrightarrow{\iota_G} & SP'_G \\ \phi \uparrow & & \uparrow \phi' \\ SP & \xrightarrow{\iota} & SP' \end{array}$$

**Amalgamation**  $A_G \otimes A'$ , iff  $A_G \upharpoonright_\phi = A' \upharpoonright_\iota$

**Translation** Assuming  $k : \text{Mod}[SP] \rightarrow \text{Mod}[SP']$  persistent and stable, we define  $k_G : \text{Mod}[SP_G] \rightarrow \text{Mod}[SP'_G]$  as  
 $k_G(A_G) := A_G \otimes k(A_G \upharpoonright_\phi)$

**Proofs** What properties do these constructed models have?  
We can use  $SP_G, SP'$ -axioms to prove the  $SP'_G$ .  
**Can we use more assumptions? Maybe stability?**

# Implementing *replace*

## CONTAINERWITHNAME

### sorts

*Container, Nat, String, Bool*

### ops

*empty : String → Container*

*insert, remove : Nat, Container → Container*

*isIn : Nat, Container → Bool*

*name : Container → String*

### observers

*(isIn, 2), (name)*



## CONTAINERWITHNAMEANDREPLACE

### sorts

*Container, Nat, String, Bool*

### ops

*empty : String → Container*

*insert, remove : Nat, Container → Container*

*isIn : Nat, Container → Bool*

*name : Container → String*

*replace : Nat, Nat, Container → Container*

### observers

*(isIn, 2), (name)*

# Implementing *replace*

## CONTAINERWITHNAME

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

### observers

(*isIn*, 2), (*name*)



## CONTAINERWITHNAMEANDREPLACE

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2), (*name*)

## CONTAINERSKELETON

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

### observers

(*isIn*, 2)



## CONTAINERSKELETONWITHREPLACE

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2)

# Implementing *replace*

## CONTAINERWITHNAME

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

### observers

(*isIn*, 2), (*name*)

### axioms

– Invariance axioms

$name(empty(n)) = n$

$name(insert(x, c)) = name(c)$

$name(remove(x, c)) = name(c)$

– Axioms for the reduct

$isIn(x, insert(y, c)) = x = y \vee isIn(x, c)$

$isIn(x, empty(n)) = false$

$isIn(x, remove(y, c)) = x \neq y \wedge isIn(x, c)$



## CONTAINERWITHNAMEANDREPLACE

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2), (*name*)

### axioms

– Theorems for free

$name(replace(x, y, c)) = name(c)$

## CONTAINERSKELETON

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

### observers

(*isIn*, 2)



## CONTAINERSKELETONWITHREPLACE

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2)

### axioms



# Implementing *replace*

## CONTAINERWITHNAME

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

### observers

(*isIn*, 2), (*name*)

### axioms

– Invariance axioms

$name(empty(n)) = n$

$name(insert(x, c)) = name(c)$

$name(remove(x, c)) = name(c)$

– Axioms for the reduct

$isIn(x, insert(y, c)) = x = y \vee isIn(x, c)$

$isIn(x, empty(n)) = false$

$isIn(x, remove(y, c)) = x \neq y \wedge isIn(x, c)$



## CONTAINERWITHNAMEANDREPLACE

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2), (*name*)

### axioms

– Theorems for free

$name(replace(x, y, c)) = name(c)$

## CONTAINERSKELETON

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

### observers

(*isIn*, 2)



## CONTAINERSKELETONWITHREPLACE

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2)

### axioms

$replace(x, y, c) = insert(x, remove(y, c))$

# Stability implies definability

Assumption - Stability:  $k$  is stable

Conclusion - Definability:  $replace(m, n, a) = t[m, n, a]$ , with  $t$  a  $\{insert, remove, a, b\}$ -term

Assumption - Invariance:  $insert$  and  $remove$  do not change the name

Conclusion - Invariance:  $replace$  does not change the name

# Implementing *replace*

## CONTAINERWITHNAME

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

### observers

(*isIn*, 2), (*name*)

### axioms

– Invariance axioms

$\text{name}(\text{empty}(n)) = n$

$\text{name}(\text{insert}(x, c)) = \text{name}(c)$

$\text{name}(\text{remove}(x, c)) = \text{name}(c)$

– Axioms for the reduct

$\text{isIn}(x, \text{insert}(y, c)) = x = y \vee \text{isIn}(x, c)$

$\text{isIn}(x, \text{empty}(n)) = \text{false}$

$\text{isIn}(x, \text{remove}(y, c)) = x \neq y \wedge \text{isIn}(x, c)$



## CONTAINERWITHNAMEANDREPLACE

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2), (*name*)

### axioms

– Theorems for free

$\text{name}(\text{replace}(x, y, c)) = \text{name}(c)$

## CONTAINERSKELETON

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

### observers

(*isIn*, 2)



## CONTAINERSKELETONWITHREPLACE

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2)

### axioms

# Implementing *replace*

## CONTAINERWITHNAME

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

### observers

(*isIn*, 2), (*name*)

### axioms

– Invariance axioms

$\text{name}(\text{empty}(n)) = n$

$\text{name}(\text{insert}(x, c)) = \text{name}(c)$

$\text{name}(\text{remove}(x, c)) = \text{name}(c)$

– Axioms for the reduct

$\text{isIn}(x, \text{insert}(y, c)) = x = y \vee \text{isIn}(x, c)$

$\text{isIn}(x, \text{empty}(n)) = \text{false}$

$\text{isIn}(x, \text{remove}(y, c)) = x \neq y \wedge \text{isIn}(x, c)$



## CONTAINERWITHNAMEANDREPLACE

### sorts

*Container*, *Nat*, *String*, *Bool*

### ops

*empty* : *String* → *Container*

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*name* : *Container* → *String*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2), (*name*)

### axioms

– Theorems for free

$\text{name}(\text{replace}(x, y, c)) = \text{name}(c)$

$\text{replace}(x, y, c) = \text{insert}(x, \text{remove}(y, c))$

## CONTAINERSKELETON

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

### observers

(*isIn*, 2)



## CONTAINERSKELETONWITHREPLACE

### sorts

*Container*, *Nat*, *Bool*

### ops

*insert*, *remove* : *Nat*, *Container* → *Container*

*isIn* : *Nat*, *Container* → *Bool*

*replace* : *Nat*, *Nat*, *Container* → *Container*

### observers

(*isIn*, 2)

### axioms

$\text{replace}(x, y, c) = \text{insert}(x, \text{remove}(y, c))$

# Conclusions and future work

- We can obtain some “theorems for free” [Wadler '89] based on types:  
 $union : Container, Container \rightarrow Container$  acts like a projection w.r.t. *name* observation
- More types! Definability for implementations of higher order functions:  
 $transform : (Container \rightarrow Container), Container \rightarrow Container$   
 $transform(f)(c) =_{name} f^n(c)$

Thank You!

# Parametricity

## Second order lambda calculus

Reynolds '74, Girard '72 System F: quantification over types

Types and terms

- polymorphic identity:  
 $(\lambda X. \lambda x : X. x) : (\forall X. X \rightarrow X)$
- polymorphic projection:  
 $(\lambda X. \lambda x, y : X. x) : (\forall X. X \times X \rightarrow X)$

Strachey '67 Ad hoc polymorphism vs parametric polymorphism

Reynolds - Parametricity

Parametric elements take related arguments to related results

Definability Parametricity implies definability

- $f : (\forall X. X \rightarrow X)$  implies  $f$  is identity
- $f : (\forall X. X \times X \rightarrow X)$  implies  $f$  is a projection