Free theorems for refinement of behavioral specifications

Marius Petria

University of Edinburgh [1]

IFIP WG1.3 2008

[1] 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: implemention of *replace* for *Containers*

Behavioral specifications

Definition (Bidoit and Hennicker's Observational Logic)

observational signature (S, F, H, O)

observational equality $\approx_A \subseteq A \times A$ - 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

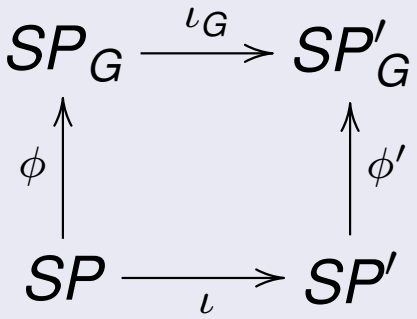
Parametric elements take related arguments to related results

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



Amalgamation $A_G \otimes A'$, iff $A_G{\restriction_\phi} = A'{\restriction_\iota}$

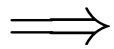
Translation Assuming $k : \mathrm{Mod}[SP] \to \mathrm{Mod}[SP']$ persistent and stable, we define $k_G : \mathrm{Mod}[SP_G] \to \mathrm{Mod}[SP'_G]$ as $k_G(A_G) := A_G \otimes k(A_G{\restriction_\phi})$

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



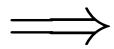
ContainerWithNameAndReplace

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*)

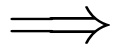


ContainerWithNameAndReplace

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
$isIn(x, insert(y, c)) = x = y \vee isIn(x, c)$
$isIn(x, empty(n)) = false$
$isIn(x, remove(y, c)) = x \neq y \wedge isIn(x, c)$

$\Longrightarrow$

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)

$\Longrightarrow$

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
$isIn(x, insert(y, c)) = x = y \vee isIn(x, c)$
$isIn(x, empty(n)) = false$
$isIn(x, remove(y, c)) = x \neq y \wedge isIn(x, c)$

$\Longrightarrow$

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)

$\Longrightarrow$

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
$name(empty(n)) = n$
$name(insert(x, c)) = name(c)$
$name(remove(x, c)) = name(c)$
– Axioms for the reduct
$isIn(x, insert(y, c)) = x = y \vee isIn(x, c)$
$isIn(x, empty(n)) = false$
$isIn(x, remove(y, c)) = x \neq y \wedge isIn(x, c)$

$\Longrightarrow$

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)

$\Longrightarrow$

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
$isIn(x, insert(y, c)) = x = y \vee isIn(x, c)$
$isIn(x, empty(n)) = false$
$isIn(x, remove(y, c)) = x \neq y \wedge isIn(x, c)$

$\Longrightarrow$

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

ContainerSkeleton

sorts
Container, *Nat*, *Bool*
ops
insert, *remove* : *Nat*, *Container* → *Container*
isIn : *Nat*, *Container* → *Bool*
observers
(*isIn*, 2)

$\Longrightarrow$

ContainerSkeletonWithReplace

sorts
Container, *Nat*, *Bool*
ops
insert, *remove* : *Nat*, *Container* → *Container*
isIn : *Nat*, *Container* → *Bool*
replace : *Nat*, *Nat*, *Container* → *Container*
observers
(*isIn*, 2)
axioms
$replace(x, y, c) = insert(x, remove(y, c))$

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

Strachey '67 Ad hoc polymorphism vs parametric polymorphism

Reynolds - Parametricity

Parametric elements take related arguments to related results

Definability Parametricity implies definability

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