Algebraic and Coalgebraic methods in software development

Manuel A. Martins [1]

MAP-i, 2017/18

[1] Mathematics Department, Aveiro University, Portugal

Signature morphism

Definition

Let $\Sigma = (S, \Omega)$ and $\Sigma' = (S', \Omega')$ be signatures. A *signature morphism* $\sigma : \Sigma \to \Sigma'$*, is a pair* $\sigma = (\sigma_{sort}, \sigma_{op})$*, where*

- $\sigma_{sorts} : S \to S'$

Signature morphism

Definition

Let $\Sigma = (S, \Omega)$ *and* $\Sigma' = (S', \Omega')$ *be signatures. A* *signature morphism* $\sigma : \Sigma \rightarrow \Sigma'$, *is a pair* $\sigma = (\sigma_{sort}, \sigma_{op})$, *where*

- $\sigma_{sorts} : S \rightarrow S'$ *and*
- $\sigma_{op} : \Omega \rightarrow \Omega'$ *is a family of functions respecting the type of operations symbols in* Ω,

Signature morphism

Definition

Let $\Sigma = (S, \Omega)$ *and* $\Sigma' = (S', \Omega')$ *be signatures. A* *signature morphism* $\sigma : \Sigma \to \Sigma'$, *is a pair* $\sigma = (\sigma_{sort}, \sigma_{op})$, *where*

- $\sigma_{sorts} : S \to S'$ *and*
- $\sigma_{op} : \Omega \to \Omega'$ *is a family of functions respecting the type of operations symbols in* Ω, *that is,* $\sigma_{op} = (\sigma_{\omega,s} : \Omega_{\omega,s} \to \Omega'_{\sigma^*_{sorts}(\omega),\sigma_{sorts}(s)})_{\omega \in S^*, s \in S}$ *(where for* $\omega = s_1 \ldots s_n \in S^*, \sigma^*_{sorts}(\omega) = \sigma_{sorts}(s_1) \ldots \sigma_{sorts}(s_n)$*).*

Signature morphism

Definition

Let $\Sigma = (S, \Omega)$ and $\Sigma' = (S', \Omega')$ be signatures. A signature morphism $\sigma : \Sigma \to \Sigma'$, is a pair $\sigma = (\sigma_{sort}, \sigma_{op})$, where

- $\sigma_{sorts} : S \to S'$ *and*
- $\sigma_{op} : \Omega \to \Omega'$ *is a family of functions respecting the type of operations symbols in* Ω, *that is,* $\sigma_{op} = (\sigma_{\omega,s} : \Omega_{\omega,s} \to \Omega'_{\sigma^*_{sorts}(\omega), \sigma_{sorts}(s)})_{\omega \in S^*, s \in S}$ *(where for* $\omega = s_1 \ldots s_n \in S^*, \sigma^*_{sorts}(\omega) = \sigma_{sorts}(s_1) \ldots \sigma_{sorts}(s_n)$*).*

Renaming, Adding, Identifying

Definition (Reduct Algebra)

Let $\mathbf{A}'$ *be a* Σ'*-algebra, and* $\sigma : \Sigma \to \Sigma'$ *be a signature morphism. The* *σ-reduct* *of* $\mathbf{A}'$ *is the* Σ*-algebra* $\mathbf{A}' \restriction_\sigma$ *defined as follows:*

- *for any* $s \in S$, $(A' \restriction_\sigma)_s = A'_{\sigma(s)}$,

Definition (Reduct Algebra)

Let $\mathbf{A}'$ be a Σ'-algebra, and $\sigma : \Sigma \to \Sigma'$ be a signature morphism. The σ-reduct of $\mathbf{A}'$ is the Σ-algebra $\mathbf{A}' \restriction_\sigma$ defined as follows:

- *for any $s \in S$, $(A' \restriction_\sigma)_s = A'_{\sigma(s)}$, and*
- *for all $f : s_1, \ldots, s_n \to s \in \Sigma$,*

$$f^{\mathbf{A}' \restriction_\sigma} = \sigma op(f)^{\mathbf{A}'}.$$

Definition (Reduct Algebra)

Let $\mathbf{A}'$ be a Σ'-algebra, and $\sigma : \Sigma \to \Sigma'$ be a signature morphism. The *σ-reduct* *of $\mathbf{A}'$ is the Σ-algebra $\mathbf{A}' \restriction_\sigma$ defined as follows:*

- *for any $s \in S$, $(A' \restriction_\sigma)_s = A'_{\sigma(s)}$, and*
- *for all $f : s_1, \ldots, s_n \to s \in \Sigma$,*

$$f^{\mathbf{A}' \restriction_\sigma} = \sigma_{op}(f)^{\mathbf{A}'}.$$

Given a morphism $h' : \mathbf{A}' \to \mathbf{B}'$, the *$\sigma$-reduct de* h' is $h' \restriction_\sigma : A' \restriction_\sigma \to B' \restriction_\sigma$ defined by $(h' \restriction_\sigma)_s = h'_{\sigma(s)}$

Satisfaction lemma

Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and X a set of variables for Σ.
Take $X'_v = \biguplus \{X_s : \sigma_{sorts}(s) = v\}$

Extension to terms

Satisfaction lemma

Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and X a set of variables for Σ.
Take $X'_v = \biguplus \{X_s : \sigma_{sorts}(s) = v\}$

Extension to terms
$\widehat{\sigma} : \mathrm{T}(\Sigma, \mathrm{X}) \to (T(\Sigma', X'))\restriction_\sigma$

(i) If $t = x : s$, then $\widehat{\sigma}(t) = x : \sigma(s)$;

(ii) If $t = c$, then $\widehat{\sigma}(t) = \sigma(c)$;

(iii) If $t = f(t_1, \ldots, t_n)$, with $f : s_1, \ldots, s_n \to s \in \Sigma$, then
$\widehat{\sigma}(t) = \sigma(f)(\widehat{\sigma}(t_0), \ldots, \widehat{\sigma}(t_n))$.

Satisfaction lemma

Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and X a set of variables for Σ.
Take $X'_v = \biguplus\{X_s : \sigma_{sorts}(s) = v\}$

Extension to terms
$\widehat{\sigma} : \mathrm{T}(\Sigma, \mathrm{X}) \to (T(\Sigma', X'))\restriction_\sigma$

(i) If $t = x : s$, then $\widehat{\sigma}(t) = x : \sigma(s)$;

(ii) If $t = c$, then $\widehat{\sigma}(t) = \sigma(c)$;

(iii) If $t = f(t_1, \ldots, t_n)$, with $f : s_1, \ldots, s_n \to s \in \Sigma$, then $\widehat{\sigma}(t) = \sigma(f)(\widehat{\sigma}(t_0), \ldots, \widehat{\sigma}(t_n))$.

And then, in a natural way, to Flas ...

Satisfaction lemma

Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and X a set of variables for Σ.
Take $X'_v = \biguplus\{X_s : \sigma_{sorts}(s) = v\}$

Extension to terms
$\widehat{\sigma} : \mathrm{T}(\Sigma, \mathrm{X}) \to (T(\Sigma', X'))\restriction_\sigma$

(i) If $t = x : s$, then $\widehat{\sigma}(t) = x : \sigma(s)$;

(ii) If $t = c$, then $\widehat{\sigma}(t) = \sigma(c)$;

(iii) If $t = f(t_1, \ldots, t_n)$, with $f : s_1, \ldots, s_n \to s \in \Sigma$, then $\widehat{\sigma}(t) = \sigma(f)(\widehat{\sigma}(t_0), \ldots, \widehat{\sigma}(t_n))$.

And then, in a natural way, to Flas ...

Satisfaction Lemma

Let Σ, Σ' be signatures, $\mathbf{A}'$ be a Σ'-algebra and ϕ be a Σ-equation. Then,

$$\mathbf{A}' \models \sigma(\phi) \text{ iff } \mathbf{A}' \restriction_\sigma \models \phi.$$

Satisfaction lemma

Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and X a set of variables for Σ.
Take $X'_v = \biguplus\{X_s : \sigma_{sorts}(s) = v\}$

Extension to terms
$\widehat{\sigma} : \mathrm{T}(\Sigma, \mathrm{X}) \to (T(\Sigma', X'))\restriction_\sigma$

(i) If $t = x : s$, then $\widehat{\sigma}(t) = x : \sigma(s)$;
(ii) If $t = c$, then $\widehat{\sigma}(t) = \sigma(c)$;
(iii) If $t = f(t_1, \ldots, t_n)$, with $f : s_1, \ldots, s_n \to s \in \Sigma$, then $\widehat{\sigma}(t) = \sigma(f)(\widehat{\sigma}(t_0), \ldots, \widehat{\sigma}(t_n))$.

And then, in a natural way, to Flas ...

Satisfaction Lemma

Let Σ, Σ' be signatures, $\mathbf{A}'$ be a Σ'-algebra and ϕ be a Σ-equation. Then,

$$\mathbf{A}' \models \sigma(\phi) \text{ iff } \mathbf{A}' \restriction_\sigma \models \phi.$$

Corollary

$$\Phi \models_\Sigma t_1 \approx t_2 \quad \Rightarrow \quad \sigma(\Phi) \models_{\Sigma'} \sigma(t_1 \approx t_2).$$

Satisfaction lemma

Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and X a set of variables for Σ.
Take $X'_v = \biguplus\{X_s : \sigma_{sorts}(s) = v\}$

Extension to terms
$\widehat{\sigma} : \mathrm{T}(\Sigma, \mathrm{X}) \to (T(\Sigma', X'))\restriction_\sigma$

(i) If $t = x : s$, then $\widehat{\sigma}(t) = x : \sigma(s)$;
(ii) If $t = c$, then $\widehat{\sigma}(t) = \sigma(c)$;
(iii) If $t = f(t_1, \ldots, t_n)$, with $f : s_1, \ldots, s_n \to s \in \Sigma$, then $\widehat{\sigma}(t) = \sigma(f)(\widehat{\sigma}(t_0), \ldots, \widehat{\sigma}(t_n))$.

And then, in a natural way, to Flas ...

Satisfaction Lemma

Let Σ, Σ' be signatures, $\mathbf{A}'$ be a Σ'-algebra and ϕ be a Σ-equation. Then,

$$\mathbf{A}' \models \sigma(\phi) \text{ iff } \mathbf{A}' \restriction_\sigma \models \phi.$$

Corollary

$$\Phi \models_\Sigma t_1 \approx t_2 \quad \Rightarrow \quad \sigma(\Phi) \models_{\Sigma'} \sigma(t_1 \approx t_2).$$

- When the implication "$\Leftarrow$" also holds, the morphism is called conservative.

Translations

Syntax

$$\begin{tikzcd} \Sigma \arrow[d, "\sigma"'] & T(\Sigma, X) \arrow[d, "\widehat{\sigma}"] \\ \Sigma' & T(\Sigma', X') \end{tikzcd}$$

Translations

Syntax

$$\begin{tikzcd} \Sigma \arrow[d, "\sigma"'] & T(\Sigma, X) \arrow[d, "\widehat{\sigma}"] \\ \Sigma' & T(\Sigma', X') \end{tikzcd}$$

Semantics

$$\begin{tikzcd} \Sigma - algebras & \Sigma - homomorphisms \\ \Sigma' - algebras \arrow[u, "\restriction_\sigma"] & \Sigma' - homomorphisms \arrow[u, "\restriction_\sigma"] \end{tikzcd}$$

Structured specifications

We follow Sannella and Tarlecki [ST88], by assuming that the software systems, described by (algebraic) specifications, are adequately represented as models of an appropriated underlying logic. Therefore, a specification describes a signature and a class the models over this signature - *the models of the specification*.

Structured specifications

We follow Sannella and Tarlecki [ST88], by assuming that the software systems, described by (algebraic) specifications, are adequately represented as models of an appropriated underlying logic. Therefore, a specification describes a signature and a class the models over this signature - *the models of the specification*.

Definition

A specification SP is a pair $\langle \Sigma, K \rangle$, *where* Σ *is a signature and* K *is a class of* Σ*-algebra. We will represent* Σ *by* $Sig(SP)$ *and* K *by* $\mathrm{Mod}(SP)$ *- the class of models of SP.*

Structured specifications

We follow Sannella and Tarlecki [ST88], by assuming that the software systems, described by (algebraic) specifications, are adequately represented as models of an appropriated underlying logic. Therefore, a specification describes a signature and a class the models over this signature - *the models of the specification*.

Definition

A specification SP is a pair $\langle \Sigma, K \rangle$, where Σ is a signature and K is a class of Σ-algebra. We will represent Σ by $Sig(SP)$ and K by $\mathrm{Mod}(SP)$ - the class of models of SP.

Structured Specifications, Why?

Structured specifications

We follow Sannella and Tarlecki [ST88], by assuming that the software systems, described by (algebraic) specifications, are adequately represented as models of an appropriated underlying logic. Therefore, a specification describes a signature and a class the models over this signature - *the models of the specification*.

Definition

A specification SP is a pair $\langle \Sigma, K \rangle$, where Σ is a signature and K is a class of Σ-algebra. We will represent Σ by $Sig(SP)$ and K by $\mathrm{Mod}(SP)$ - the class of models of SP.

Structured Specifications, Why?

When we deal with real complex systems, it is worth to systematize the algebraic **programme development**. It is in this way that Structured Specifications appear based in the compositional principle.

Structured specifications

We follow Sannella and Tarlecki [ST88], by assuming that the software systems, described by (algebraic) specifications, are adequately represented as models of an appropriated underlying logic. Therefore, a specification describes a signature and a class the models over this signature - *the models of the specification*.

Definition

A specification SP is a pair $\langle \Sigma, K \rangle$*, where* Σ *is a signature and* K *is a class of* Σ*-algebra. We will represent* Σ *by* $Sig(SP)$ *and* K *by* $\mathrm{Mod}(SP)$ *- the class of models of SP.*

Structured Specifications, Why?

When we deal with real complex systems, it is worth to systematize the algebraic **programme development**. It is in this way that Structured Specifications appear based in the compositional principle.

We build more complex specification from simpler ones following the modular development of programmes.

Basic operators

1. **flat specifications** - to define specifications by the class of models of a set of axioms Φ over a signature Σ;

Basic operators

1. **flat specifications** - to define specifications by the class of models of a set of axioms Φ over a signature Σ;
2. **union** - to define a specification from the union of two given specifications over a same signature.

Basic operators

1. **flat specifications** - to define specifications by the class of models of a set of axioms Φ over a signature Σ;
2. **union** - to define a specification from the union of two given specifications over a same signature.
3. **translate** - to define a specification over a signature Σ' from a specification over another specification over a signature Σ using a signature morphism $\sigma : \Sigma \to \Sigma'$.

Basic operators

1. **flat specifications** - to define specifications by the class of models of a set of axioms Φ over a signature Σ;
2. **union** - to define a specification from the union of two given specifications over a same signature.
3. **translate** - to define a specification over a signature Σ' from a specification over another specification over a signature Σ using a signature morphism $\sigma : \Sigma \to \Sigma'$.
4. **derive (or Hidding)** - to define a specification over a signature Σ from a specification over another specification over a signature Σ' using a signature morphism $\sigma : \Sigma \to \Sigma'$, by considering the reducts.

Basic operators

- **flat**
  - Syntax:

    $< .,. >$: $\mathit{Sig}, \mathit{Sentences} \to \mathit{Spec}$
  - Semantics: Σ a signature and Φ a set of sentences over Σ.

    $\mathit{Sig}(<\Sigma, \Phi>) = \Sigma$

    $\mathrm{Mod}(<\Sigma, \Phi>) =_{\mathit{def}} \{\mathbf{A} \in \mathit{Alg}(\Sigma) | \mathbf{A} \models \Phi\}$

Basic operators

- **flat**
  - Syntax:

    $< .,. >$: *Sig*, *Sentences* $\to$ *Spec*
  - Semantics: Σ a signature and Φ a set of sentences over Σ.

    $Sig(< \Sigma, \Phi >) = \Sigma$

    $\mathrm{Mod}(< \Sigma, \Phi >) =_{def} \{\mathbf{A} \in Alg(\Sigma) | \mathbf{A} \models \Phi\}$

- **union**

Let SP_1 e SP_2 be specifications over a same signature Σ:

  - Syntax:

    $< . \cup . >$: *Spec*, *Spec* $\to$ *Spec*
  - Sematics:

    $Sig(SP1 \cup SP2) = Sig(SP_1) = Sig(SP_2)$

    $\mathrm{Mod}(SP1 \cup SP2) =_{def} \mathrm{Mod}(SP_1) \cap \mathrm{Mod}(SP_2)$

Basic operators

- **translate**
  - Syntax:

    translate . **by** . : *Spec*, *morph* $\to$ *Spec*
  - Semantics: let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and *SP* a specification with $Sig(SP) = \Sigma$.

    $Sig(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \Sigma'$

    $\mathrm{Mod}(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \{\mathbf{A}' \in \mathrm{Alg}(\Sigma') | \mathbf{A}' \restriction_\sigma \in \mathrm{Mod}(SP)\}$.

Basic operators

- **translate**
  - Syntax:

    translate . **by** . : *Spec*, *morph* $\to$ *Spec*
  - Semantics: let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and *SP* a specification with $Sig(SP) = \Sigma$.

    $Sig(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \Sigma'$

    $\mathrm{Mod}(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \{\mathbf{A}' \in \mathrm{Alg}(\Sigma') | \mathbf{A}' \restriction_\sigma \in \mathrm{Mod}(SP)\}$.
- **derive** (or **Hiding**)
  - Syntax:

    derive from . **by** . : *Spec*, *morph* $\to$ *Spec*
  - Semantics: Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism morfismo and *SP* a specification with $Sig(SP) = \Sigma'$.

    $Sig(\textbf{derive from } SP \textbf{ by } \sigma) =_{def} \Sigma$

    $\mathrm{Mod}(\textbf{derive from } SP \textbf{ by } \sigma) =_{def} \{\mathbf{A}' \restriction_\sigma \in \mathrm{Alg}(\Sigma) | \mathbf{A}' \in \mathrm{Mod}(SP)\}$.

- **translate**
  - Syntax:

    translate . **by** . : *Spec*, *morph* $\rightarrow$ *Spec*
  - Semantics: let $\sigma : \Sigma \rightarrow \Sigma'$ be a signature morphism and *SP* a specification with $Sig(SP) = \Sigma$.

    $Sig(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \Sigma'$

    $\mathrm{Mod}(\textbf{translate } SP \textbf{ by } \sigma) =_{def} \{\mathbf{A}' \in \mathrm{Alg}(\Sigma') | \mathbf{A}' \restriction_\sigma \in \mathrm{Mod}(SP)\}.$
- **derive** (or **Hiding**)
  - Syntax:

    derive from . **by** . : *Spec*, *morph* $\rightarrow$ *Spec*
  - Semantics: Let $\sigma : \Sigma \rightarrow \Sigma'$ be a signature morphism morfismo and *SP* a specification with $Sig(SP) = \Sigma'$.

    $Sig(\textbf{derive from } SP \textbf{ by } \sigma) =_{def} \Sigma$

    $\mathrm{Mod}(\textbf{derive from } SP \textbf{ by } \sigma) =_{def} \{\mathbf{A}' \restriction_\sigma \in \mathrm{Alg}(\Sigma) | \mathbf{A}' \in \mathrm{Mod}(SP)\}.$

A **structured specification** is a specification *SP* obtained by a finite number of applications o these 4 operators.

Equational case

Not all algebraic specification (classe of algebras) can be axiomatized by a set of equations.

So,

Fact

Not all specifications are flat specifications

Equational case

Not all algebraic specification (classe of algebras) can be axiomatized by a set of equations.

So,

Fact

Not all specifications are flat specifications

Birkhoff's theorem

A specification is flat iff the class of algebras is closed by subalgebras, homomorphic images and products.

Equational case

Not all algebraic specification (classe of algebras) can be axiomatized by a set of equations.

So,

Fact

Not all specifications are flat specifications

Birkhoff's theorem

A specification is flat iff the class of algebras is closed by subalgebras, homomorphic images and products.

Even with first-order formulas it is impossible!

More useful operators

- **enrich**: To add new sorts, new axioms and new operation symbols
Let $\Sigma = (S, \Omega)$, $\Sigma' = (S \cup S', \Omega \cup \Omega')$ and $\iota : \Sigma \hookrightarrow \Sigma'$ the inclusion morphism.

$$\textbf{enrich } SP \textbf{ by sorts } S' \textbf{ opns } F' \textbf{ axioms } \Phi' = (\textbf{translate } SP \textbf{ by } \iota) \cup \langle \Sigma', \Phi' \rangle$$

More useful operators

- **enrich**: To add new sorts, new axioms and new operation symbols

Let $\Sigma = (S, \Omega)$, $\Sigma' = (S \cup S', \Omega \cup \Omega')$ and $\iota : \Sigma \hookrightarrow \Sigma'$ the inclusion morphism.

$$\textbf{enrich } SP \textbf{ by sorts } S' \textbf{ opns } F' \textbf{ axioms } \Phi' = (\textbf{translate } SP \textbf{ by } \iota) \cup \langle \Sigma', \Phi' \rangle$$

- **export**: A particular case of **derive**, the morphism is the inclusion, i.e., let $\iota : \Sigma \hookrightarrow \Sigma'$:

$$\textbf{export } \Sigma' \textbf{ from } SP = \textbf{ derive from } SP \textbf{ by } \iota.$$

Reach operator

- A *reachability constraint of* Σ is a pair $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$ s.t. $F_{\mathcal{R}} \subseteq \Omega$ and $S_{\mathcal{R}} = \{s \in S |$ existe um $f \in (F_{\mathcal{R}})_{ws}\}$. ▶ An $s \in S_{\mathcal{R}}$ is called a *constrained sort* and a symbol $f \in F_{\mathcal{R}}$ *a constructor*.

Reach operator

- A *reachability constraint of* Σ is a pair $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$ s.t. $F_{\mathcal{R}} \subseteq \Omega$ and $S_{\mathcal{R}} = \{s \in S |$ existe um $f \in (F_{\mathcal{R}})_{ws}\}$. ▶ An $s \in S_{\mathcal{R}}$ is called a *constrained sort* and a symbol $f \in F_{\mathcal{R}}$ *a constructor*.

- A *constructor term* is a $t \in T(\Sigma', X')_s$, where $\Sigma' = \langle S, F_{\mathcal{R}} \rangle$, $X' = X_s$ if $s \in S \setminus S_{\mathcal{R}}$, and $X'_s = \emptyset$ if $s \in S_{\mathcal{R}}$.

Reach operator

- A *reachability constraint of* Σ is a pair $\mathcal{R} = \langle S_\mathcal{R}, F_\mathcal{R}\rangle$ s.t. $F_\mathcal{R} \subseteq \Omega$ and $S_\mathcal{R} = \{s \in S |$ existe um $f \in (F_\mathcal{R})_{ws}\}$. ▶ An $s \in S_\mathcal{R}$ is called a *constrained sort* and a symbol $f \in F_\mathcal{R}$ *a constructor.*
- A *constructor term* is a $t \in T(\Sigma', X')_s$, where $\Sigma' = \langle S, F_\mathcal{R}\rangle$, $X' = X_s$ if $s \in S \setminus S_\mathcal{R}$, and $X'_s = \emptyset$ if $s \in S_\mathcal{R}$.

A Σ-algebra $\mathbf{A}$, satisfies a *reachability constraint* $\mathcal{R} = \langle S_\mathcal{R}, F_\mathcal{R}\rangle$, $\mathbf{A} \models \mathcal{R}$, if for all $s \in S$ and every $a \in A_s$, there exists a constructor term t and an evaluation $\alpha : X' \to A$ s.t. $\alpha(t) = a$.

Reach operator

- A *reachability constraint of* Σ is a pair $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$ s.t. $F_{\mathcal{R}} \subseteq \Omega$ and $S_{\mathcal{R}} = \{s \in S |$ existe um $f \in (F_{\mathcal{R}})_{ws}\}$. ▶ An $s \in S_{\mathcal{R}}$ is called a *constrained sort* and a symbol $f \in F_{\mathcal{R}}$ *a constructor.*

- A *constructor term* is a $t \in T(\Sigma', X')_s$, where $\Sigma' = \langle S, F_{\mathcal{R}} \rangle$, $X' = X_s$ if $s \in S \setminus S_{\mathcal{R}}$, and $X'_s = \emptyset$ if $s \in S_{\mathcal{R}}$.

A Σ-algebra $\mathbf{A}$, satisfies a *reachability constraint* $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$, $\mathbf{A} \models \mathcal{R}$, if for all $s \in S$ and every $a \in A_s$, there exists a constructor term t and an evaluation $\alpha : X' \to A$ s.t. $\alpha(t) = a$.

Theorem

Let $\mathbf{A}$ *be a* Σ*-algebra and* $\mathcal{R}$ *a reachability constraint over* Σ*. TFAE*

1. $\mathbf{A} \models \mathcal{R}$
2. *for every* $s \in S$*, and any* $a \in A_s$ *there exists a constructor term* t *of sort* S *such that* $\mathbf{A}, \alpha \models \exists Var(t).x = t$*, where* $x \in X_s$*,* $x \notin Var(t)$ *and* $\alpha : X \to A$ *an evaluation such that* $\alpha(x) = a$*.*

Reach operator

- A *reachability constraint of* Σ is a pair $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$ s.t. $F_{\mathcal{R}} \subseteq \Omega$ and $S_{\mathcal{R}} = \{s \in S |$ existe um $f \in (F_{\mathcal{R}})_{ws}\}$. ▶ An $s \in S_{\mathcal{R}}$ is called a *constrained sort* and a symbol $f \in F_{\mathcal{R}}$ *a constructor*.

- A *constructor term* is a $t \in T(\Sigma', X')_s$, where $\Sigma' = \langle S, F_{\mathcal{R}} \rangle$, $X' = X_s$ if $s \in S \setminus S_{\mathcal{R}}$, and $X'_s = \emptyset$ if $s \in S_{\mathcal{R}}$.

A Σ-algebra $\mathbf{A}$, satisfies a *reachability constraint* $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$, $\mathbf{A} \models \mathcal{R}$, if for all $s \in S$ and every $a \in A_s$, there exists a constructor term t and an evaluation $\alpha : X' \to A$ s.t. $\alpha(t) = a$.

Theorem

Let $\mathbf{A}$ *be a* Σ*-algebra and* $\mathcal{R}$ *a reachability constraint over* Σ*. TFAE*

1. $\mathbf{A} \models \mathcal{R}$
2. *for every* $s \in S$*, and any* $a \in A_s$ *there exists a constructor term* t *of sort* S *such that* $\mathbf{A}, \alpha \models \exists Var(t).x = t$*, where* $x \in X_s$*,* $x \notin Var(t)$ *and* $\alpha : X \to A$ *an evaluation such that* $\alpha(x) = a$*.*
3. *For all* $s \in S$*,*
$$\mathbf{A} \models (\forall x : s) \bigvee_{t \in (T_{\mathcal{R}})_s} \exists Var(t)\, x \approx t.$$

Reach operator

reach

- Syntax: **reach** **with** . : *Spec*, *Opns* $\rightarrow$ *Spec*
- Semantics:

  Let $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$ a *reachability constraint* over $Sig(SP)$

  $Sig(\textbf{reach } SP \textbf{ with } F_{\mathcal{R}}) = Sig(SP)$

  $\mathrm{Mod}(\textbf{ reach } SP \textbf{ with } F_{\mathcal{R}}) = \{\mathbf{A} \in \mathrm{Mod}(SP) | \mathbf{A} \models \mathcal{R}\}$

Reach operator

reach

- Syntax: **reach** **with** . : *Spec*, *Opns* $\rightarrow$ *Spec*
- Semantics:

  Let $\mathcal{R} = \langle S_{\mathcal{R}}, F_{\mathcal{R}} \rangle$ a *reachability constraint* over $Sig(SP)$

  $Sig(\textbf{reach}\ SP\ \textbf{with}\ F_{\mathcal{R}}) = Sig(SP)$

  $\mathrm{Mod}(\ \textbf{reach}\ SP\ \textbf{with}\ F_{\mathcal{R}}) = \{\mathbf{A} \in \mathrm{Mod}(SP) | \mathbf{A} \models \mathcal{R}\}$

Example

$INTZERO =$ **reach** INT **with**
$F_{\mathcal{R}} =$
$0 : \rightarrow\ int;$
$s,p:\ int \rightarrow int;$

Examples [ST88]

$$
\begin{array}{lll}
BOOL = & \textbf{sorts} & \mathit{bool} \\
 & \textbf{opns} & \mathit{true} : \mathit{bool} \\
 & & \mathit{false} : \mathit{bool} \\
 & \textbf{axioms} & \mathit{true} \neq \mathit{false} \\
 & & \forall x{:}\mathit{bool}.\, x = \mathit{true} \vee x = \mathit{false}
\end{array}
$$

Examples [ST88]

$$
\begin{array}{lll}
BOOL = & \textbf{sorts} & bool \\
 & \textbf{opns} & true : bool \\
 & & false : bool \\
 & \textbf{axioms} & true \neq false \\
 & & \forall x{:}bool.\, x = true \vee x = false
\end{array}
$$

$$
\begin{array}{lll}
INT = & \multicolumn{2}{l}{\textbf{enrich } BOOL \textbf{ by}} \\
 & \textbf{sorts} & int \\
 & \textbf{opns} & 0 : int \\
 & & succ : int \rightarrow int \\
 & & pred : int \rightarrow int \\
 & \textbf{axioms} & \ldots \text{induction scheme for } int \ldots \\
 & & \forall x{:}int.\, pred(x) \neq x \wedge succ(x) \neq x \\
 & & \forall x{:}int.\, pred(succ(x)) = x \wedge succ(pred(x)) = x
\end{array}
$$

Examples [ST88]

$$
\begin{array}{lll}
BOOL = & \textbf{sorts} & bool \\
& \textbf{opns} & true : bool \\
& & false : bool \\
& \textbf{axioms} & true \neq false \\
& & \forall x{:}bool.\, x = true \vee x = false
\end{array}
$$

$$
\begin{array}{lll}
INT = & \multicolumn{2}{l}{\textbf{enrich } BOOL \textbf{ by}} \\
& \textbf{sorts} & int \\
& \textbf{opns} & 0 : int \\
& & succ : int \to int \\
& & pred : int \to int \\
& \textbf{axioms} & \ldots \text{induction scheme for } int \ldots \\
& & \forall x{:}int.\, pred(x) \neq x \wedge succ(x) \neq x \\
& & \forall x{:}int.\, pred(succ(x)) = x \wedge succ(pred(x)) = x
\end{array}
$$

$$
\begin{array}{lll}
INTORD = & \multicolumn{2}{l}{\textbf{enrich } INT \textbf{ by}} \\
& \textbf{opns} & po : int \times int \to bool \\
& \textbf{axioms} & \forall x{:}int.\, po(x,x) = true \\
& & \forall x, y{:}int.\, po(x,y) = true \wedge po(y,x) = true \Longrightarrow x = y \\
& & \forall x, y, z{:}int.\, po(x,y) = true \wedge po(y,z) = true \Longrightarrow po(x,z) = true
\end{array}
$$

EXAMPLE [ST88]

$$
\begin{array}{lll}
\mathit{INTLIST} = & \multicolumn{2}{l}{\textbf{enrich } \mathit{INTORD} \textbf{ by}} \\
& \textbf{sorts} & \mathit{list} \\
& \textbf{opns} & \mathit{nil} : \mathit{list} \\
& & \mathit{cons} : \mathit{int} \times \mathit{list} \to \mathit{list} \\
& & \mathit{head} : \mathit{list} \to \mathit{int} \\
& & \mathit{tail} : \mathit{list} \to \mathit{list} \\
& & \mathit{append} : \mathit{list} \times \mathit{list} \to \mathit{list} \\
& & \mathit{is\_in} : \mathit{int} \times \mathit{list} \to \mathit{bool} \\
& \textbf{axioms} & \ldots \text{induction scheme for } \mathit{list} \ldots \\
& & \forall x{:}\mathit{int}.\, \forall l{:}\mathit{list}.\ \mathit{cons}(x,l) \neq l \\
& & \forall x{:}\mathit{int}.\, \forall l{:}\mathit{list}.\ \mathit{head}(\mathit{cons}(x,l)) = x \\
& & \forall x{:}\mathit{int}.\, \forall l{:}\mathit{list}.\ \mathit{tail}(\mathit{cons}(x,l)) = l \\
& & \forall l{:}\mathit{list}.\ \mathit{append}(\mathit{nil},l) = l \\
& & \forall x{:}\mathit{int}.\, \forall l,l'{:}\mathit{list}.\ \mathit{append}(\mathit{cons}(x,l),l') = \mathit{cons}(x,\mathit{append}(l,l')) \\
& & \forall x{:}\mathit{int}.\ \mathit{is\_in}(x,\mathit{nil}) = \mathit{false} \\
& & \forall x,y{:}\mathit{int}.\, \forall l{:}\mathit{list}.\ \mathit{is\_in}(x,\mathit{cons}(y,l)) = \mathit{true} \iff \\
& & \qquad\qquad (x = y \vee \mathit{is\_in}(x,l) = \mathit{true})
\end{array}
$$

Calculus for Structured specifications

$$\frac{\varphi \in \Phi}{\langle \Sigma, \Phi \rangle \vdash \varphi}$$

$$\frac{SP_1 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi}$$

$$\frac{SP_2 \vdash \varphi}{SP_1 \cup SP_2 \vdash \varphi}$$

$$\frac{SP \vdash \varphi}{\textbf{translate } SP \textbf{ by } \sigma \vdash \sigma(\varphi)}$$

$$\frac{SP' \vdash \sigma(\varphi)}{\textbf{derive from } SP' \textbf{ by } \sigma \vdash \varphi}$$

$$\frac{SP \vdash \varphi_1 \ldots SP \vdash \varphi_n \text{ e } \{\varphi_1, \ldots, \varphi_n\} \vdash \varphi}{SP \vdash \varphi}$$

Completeness

$$SP \models \varphi \quad \textit{iff} \quad SP \vdash \varphi.$$

Completeness

$$SP \models \varphi \quad \mathit{iff} \quad SP \vdash \varphi.$$

"$\Leftarrow$" - if $\vdash_\Sigma$ is sound.

Completeness

$$SP \models \varphi \quad \textit{iff} \quad SP \vdash \varphi.$$

"$\Leftarrow$" - if $\vdash_\Sigma$ is sound.

"$\Rightarrow$" - if the underlying logic (institution) has pushouts, amalgamation property and $\vdash_\Sigma$ é complete for the logic semantics.

Completeness

$$SP \models \varphi \quad \textit{iff} \quad SP \vdash \varphi.$$

"$\Leftarrow$" - if $\vdash_\Sigma$ is sound.

"$\Rightarrow$" - if the underlying logic (institution) has pushouts, amalgamation property and $\vdash_\Sigma$ é complete for the logic semantics.

A more abstract treatment using INSTITUTIONS.

Stepwise refinement process

The *stepwise refinement process* is the systematic process by which, from a specification SP_0 we successively build more restrictive specifications by introducing new requirements:

$$SP_0 \rightsquigarrow SP_1 \rightsquigarrow SP_2 \rightsquigarrow \cdots \rightsquigarrow SP_{n-1} \rightsquigarrow SP_n,$$

where for all $1 \leq i \leq n$, $SP_{i-1} \rightsquigarrow SP_i$ is a refinement.

The software development - the stepwise refinement methodology

Definition (Refinement)

Let SP and SP' be specifications. SP' is a refinement of SP if:

- $Sig(SP) = Sig(SP')$;
- $\mathrm{Mod}(SP') \subseteq \mathrm{Mod}(SP)$;

We write $SP \rightsquigarrow SP'$ when SP' is a refinement of SP.

The software development - the stepwise refinement methodology

Definition (Refinement)

Let SP and SP' be specifications. SP' is a refinement of SP if:

- $Sig(SP) = Sig(SP')$;
- $\mathrm{Mod}(SP') \subseteq \mathrm{Mod}(SP)$;

We write $SP \rightsquigarrow SP'$ when SP' is a refinement of SP.

Definition (σ-refinement)

Let SP and SP' be algebraic specifications and $\sigma : \mathrm{Sig}(SP) \to \mathrm{Sig}(SP')$. SP' is a σ-refinement of SP, in symbols $SP \rightsquigarrow_\sigma SP'$, if:

- $\mathrm{Mod}(SP') \restriction_\sigma \subseteq \mathrm{Mod}(SP)$,

where $\mathrm{Mod}(SP') \restriction_\sigma = \{\mathbf{A}' \restriction_\sigma | \mathbf{A}' \in \mathrm{Mod}(SP')\}$.

Compositionality

Vertical composition

$SP \rightsquigarrow_{\sigma} SP' \rightsquigarrow_{\phi} SP''$
$\mathrm{Mod}(SP'') \restriction_{\phi \circ \sigma} \subseteq \mathrm{Mod}(SP') \restriction_{\sigma} \subseteq \mathrm{Mod}(SP)$

Compositionality

Vertical composition

$SP \rightsquigarrow_{\sigma} SP' \rightsquigarrow_{\phi} SP''$
$\mathrm{Mod}(SP'') \restriction_{\phi \circ \sigma} \subseteq \mathrm{Mod}(SP') \restriction_{\sigma} \subseteq \mathrm{Mod}(SP)$

Stepwise Refinement Process:

$$SP_0 \rightsquigarrow_{\sigma_0} SP_1 \rightsquigarrow_{\sigma_1} SP_2 \rightsquigarrow_{\sigma_2} \ldots \rightsquigarrow_{\sigma_{n-2}} SP_{n-1} \rightsquigarrow_{\sigma_{n-1}} SP_n.$$

Compositionality

Vertical composition

$SP \rightsquigarrow_{\sigma} SP' \rightsquigarrow_{\phi} SP''$
$\mathrm{Mod}(SP'') \restriction_{\phi \circ \sigma} \subseteq \mathrm{Mod}(SP') \restriction_{\sigma} \subseteq \mathrm{Mod}(SP)$

Stepwise Refinement Process:

$$SP_0 \rightsquigarrow_{\sigma_0} SP_1 \rightsquigarrow_{\sigma_1} SP_2 \rightsquigarrow_{\sigma_2} \ldots \rightsquigarrow_{\sigma_{n-2}} SP_{n-1} \rightsquigarrow_{\sigma_{n-1}} SP_n.$$

Horizontal composition

$$\frac{SP_1 \rightsquigarrow SP'_1, \ldots, SP_n \rightsquigarrow SP'_n}{op(SP_1, \ldots, SP_n) \rightsquigarrow op(SP'_1, \ldots, SP'_n)}$$

Compositionality

Vertical composition

$SP \rightsquigarrow_\sigma SP' \rightsquigarrow_\phi SP''$
$\mathrm{Mod}(SP'') \restriction_{\phi\circ\sigma} \subseteq \mathrm{Mod}(SP') \restriction_\sigma \subseteq \mathrm{Mod}(SP)$

Stepwise Refinement Process:

$$SP_0 \rightsquigarrow_{\sigma_0} SP_1 \rightsquigarrow_{\sigma_1} SP_2 \rightsquigarrow_{\sigma_2} \dots \rightsquigarrow_{\sigma_{n-2}} SP_{n-1} \rightsquigarrow_{\sigma_{n-1}} SP_n.$$

Horizontal composition

$$\frac{SP_1 \rightsquigarrow SP'_1, \dots, SP_n \rightsquigarrow SP'_n}{op(SP_1, \dots, SP_n) \rightsquigarrow op(SP'_1, \dots, SP'_n)}$$

Horizontal composition - not so easy!

Horizontal composition

Theorem

Let $\Sigma \subseteq \Sigma'$ *and suppose* $SP_0 \rightsquigarrow_\iota SP'_0$ *and* $SP_1 \rightsquigarrow_\iota SP'_1$*, and* $\phi : \Sigma' \to \Sigma''$ *a signature morphisms. Then*

1. $SP_0 \cup SP_1 \rightsquigarrow_\iota SP'_0 \cup SP'_1$;
2. $\textbf{translate } SP_0 \textbf{ by } \phi\restriction_\Sigma \rightsquigarrow_\iota \textbf{translate } SP'_0 \textbf{ by } \phi$;

Limitations of the classical approach

spec SPEC1 =
sorts
s;
ops
$f : s \to s$;
Ax + Ir

$$t \approx t; \quad \frac{t \approx t'}{t' \approx t} \quad \frac{t \approx t', t' \approx t''}{t \approx t''}; \quad \frac{t \approx t'}{f(t) \approx f(t')}$$

spec SPEC2 =
sorts
s;
ops
$ok :\to s, f : s \to s, test : s \times s \to s$;
Ax + Ir

$$test(t,t) \approx ok; \quad \frac{test(t,t') \approx ok}{test(t',t) \approx ok}; \quad \frac{test(t,t') \approx ok, test(t',t'') \approx ok}{test(t,t'') \approx ok}; \quad \frac{test(t,t') \approx ok}{test(f(t),f(t')) \approx ok};$$

Limitations of the classical approach

spec $\mathrm{SPEC1} =$
sorts
s;
ops
$f : s \to s$;
Ax + Ir
$t \approx t$;
$\dfrac{t \approx t'}{t' \approx t}$
$\dfrac{t \approx t', t' \approx t''}{t \approx t''}$;
$\dfrac{t \approx t'}{f(t) \approx f(t')}$

spec $\mathrm{SPEC2} =$
sorts
s;
ops
$ok : \to s, f : s \to s, test : s \times s \to s$;
Ax + Ir
$test(t, t) \approx ok$;
$\dfrac{test(t, t') \approx ok}{test(t', t) \approx ok}$;
$\dfrac{test(t, t') \approx ok, test(t', t'') \approx ok}{test(t, t'') \approx ok}$;
$\dfrac{test(t, t') \approx ok}{test(f(t), f(t')) \approx ok}$;

- Naturally, $\mathrm{SPEC1} \models \varphi \approx \varphi'$ iff $\mathrm{SPEC2} \models test(\varphi, \varphi') \approx ok$

Limitations of the classical approach

spec $\mathrm{SPEC1} =$
sorts
s;
ops
$f : s \to s$;
Ax + Ir

$$t \approx t; \quad \frac{t \approx t'}{t' \approx t} \quad \frac{t \approx t', t' \approx t''}{t \approx t''}; \quad \frac{t \approx t'}{f(t) \approx f(t')}$$

spec $\mathrm{SPEC2} =$
sorts
s;
ops
$ok : \to s, f : s \to s, test : s \times s \to s$;
Ax + Ir

$$test(t,t) \approx ok; \quad \frac{test(t,t') \approx ok}{test(t',t) \approx ok}; \quad \frac{test(t,t') \approx ok, test(t',t'') \approx ok}{test(t,t'') \approx ok}; \quad \frac{test(t,t') \approx ok}{test(f(t),f(t')) \approx ok};$$

- Naturally, $\mathrm{SPEC1} \models \varphi \approx \varphi'$ iff $\mathrm{SPEC2} \models test(\varphi, \varphi') \approx ok$
- However, $\iota : Sig(\mathrm{SPEC1}) \to Sig(\mathrm{SPEC2})$ is the unique morphism definable between the specifications of SPEC1 and SPEC2.

Motivations

Refinement based on signature morphisms

- a formula is mapped into another one;
- formula structure is preserved;

Motivations

Refinement based on signature morphisms

- a formula is mapped into another one;
- formula structure is preserved;

Thus, it is difficult to deal with some specification transformations such as data encapsulation, decomposition of operations in atomic transactions, ... which are useful in practice.

Motivations

Refinement based on signature morphisms

- a formula is mapped into another one;
- formula structure is preserved;

Thus, it is difficult to deal with some specification transformations such as data encapsulation, decomposition of operations in atomic transactions, ... which are useful in practice.

The strategy

- Introduce a formalization of the refinement where the translation of specifications is witnessed by a suitable kind of multifunctions;

Motivations

Refinement based on signature morphisms

- a formula is mapped into another one;
- formula structure is preserved;

Thus, it is difficult to deal with some specification transformations such as data encapsulation, decomposition of operations in atomic transactions, ... which are useful in practice.

The strategy

- Introduce a formalization of the refinement where the translation of specifications is witnessed by a suitable kind of multifunctions;
- Generalize this approach by allowing translations between specifications expressed in logics with different dimensions;

Interpretations within algebraic specification

Refinement by interpretations

A translation $\tau : \mathrm{Eq}(\Sigma) \to \mathcal{P}(\mathrm{Eq}(\Sigma'))$ *interprets* SP if there is a specification SP' over Σ' such that:

- for all $t \approx t' \in \mathrm{Eq}(\mathrm{Sig}(SP)), SP \models t \approx t'$ iff $SP' \models \tau(t \approx t')$

Interpretations within algebraic specification

Refinement by interpretations

A translation $\tau : \mathrm{Eq}(\Sigma) \to \mathcal{P}(\mathrm{Eq}(\Sigma'))$ *interprets* SP if there is a specification SP' over Σ' such that:

- for all $t \approx t' \in \mathrm{Eq}(\mathrm{Sig}(SP)), SP \models t \approx t'$ iff $SP' \models \tau(t \approx t')$

A mathematical example

The self translation $\tau(t \approx t') = \{\neg\neg t \approx \neg\neg t'\}$ interprets the specification $\mathbb{BA}$ (boolean algebras) in the specification $\mathbb{HA}$ (Heyting algebras).

Interpretations within algebraic specification

Refinement by interpretations

A translation $\tau : \mathrm{Eq}(\Sigma) \to \mathcal{P}(\mathrm{Eq}(\Sigma'))$ *interprets* SP if there is a specification SP' over Σ' such that:

- for all $t \approx t' \in \mathrm{Eq}(\mathrm{Sig}(SP)), SP \models t \approx t'$ iff $SP' \models \tau(t \approx t')$

A mathematical example

The self translation $\tau(t \approx t') = \{\neg\neg t \approx \neg\neg t'\}$ interprets the specification $\mathbb{BA}$ (boolean algebras) in the specification $\mathbb{HA}$ (Heyting algebras).

Definition

SP' is a refinement by the interpretation τ of SP if

- *τ interprets SP and*
- *for all $t \approx t' \in \mathrm{Eq}(\mathrm{Sig}(SP)), SP \models t \approx t'$ implies $SP' \models \tau(t \approx t')$*

Ex. BAMS: replacing operations by atomic transactions

Σ_1 :

sorts

$Ac; Int;$

ops

$bal : Ac \to Int;$

$cred, deb : Ac \times Int \to Ac$

spec BAMS = enrich EQ_{Σ_1}
and INT with

axioms

$bal(cred(x, n)) \approx bal(x) + n;$

$bal(deb(x, n)) \approx bal(x) + (-n).$

Ex. BAMS: replacing operations by atomic transactions

Σ_1 :

sorts

$Ac; Int;$

ops

$bal : Ac \to Int;$

$cred, deb : Ac \times Int \to Ac$

spec BAMS = enrich EQ_{Σ_1} and INT with

axioms

$bal(cred(x, n)) \approx bal(x) + n;$

$bal(deb(x, n)) \approx bal(x) + (-n).$

Σ_2 :

sorts

$Ac; Int;$

ops

...

$val : Ac \to Ac$

spec $\mathrm{BAMS2}$ = enrich EQ_{Σ_2} and INT with

axioms

$bal(val(cred(x, n)) \approx bal(x) + n;$

$bal(val(deb(x, n)) \approx bal(x) + (-n).$

Ex. BAMS: replacing operations by atomic transactions

Σ_1 :
sorts
$Ac; Int;$
ops
$bal : Ac \to Int;$
$cred, deb : Ac \times Int \to Ac$

spec BAMS = enrich EQ_{Σ_1} and INT with
axioms
$bal(cred(x, n)) \approx bal(x) + n;$
$bal(deb(x, n)) \approx bal(x) + (-n).$

Σ_2 :
sorts
$Ac; Int;$
ops
...
$val : Ac \to Ac$

spec $\mathrm{BAMS2}$ = enrich EQ_{Σ_2} and INT with
axioms
$bal(val(cred(x, n)) \approx bal(x) + n;$
$bal(val(deb(x, n)) \approx bal(x) + (-n).$

$\tau : \mathrm{Eq}(\Sigma_1) \to \mathcal{P}(\mathrm{Eq}(\Sigma_2)) = \{\langle op(x), y\rangle \to \{\langle val(op(x)), y\rangle\} | op \in \{cred, deb\}\}$

Ex. NatBool: encapsulating sorts

- **Spec Nat=** enrich $EQ_{\Sigma_{Nat}}$ by

ops $s : nat \to nat$;

IR $\frac{s(x) \approx s(y)}{x \approx y}$

- **Spec NatEq=** enrich BOOL by

sorts nat;

ops $s : nat \to nat; eq : nat, nat \to bool$;

axioms $eq(x, x) \approx true$

IR $\frac{eq(x, y) \approx true}{eq(y, x) \approx true}$; $\frac{eq(x, y) \approx true\ ,\ eq(y, z) \approx true}{eq(x, z) \approx true}$;

$\frac{eq(x, y) \approx true}{eq(s(x), s(y)) \approx true}$; $\frac{eq(s(x), s(y)) \approx true}{eq(x, y) \approx true}$.

Ex. NatBool: encapsulating sorts

- Spec Nat= enrich $EQ_{\Sigma_{Nat}}$ by

ops $s : nat \to nat;$

IR $\dfrac{s(x) \approx s(y)}{x \approx y}$

- Spec NatEq= enrich BOOL by

sorts $nat;$

ops $s : nat \to nat; eq : nat, nat \to bool;$

axioms $eq(x,x) \approx true$

IR $\dfrac{eq(x,y) \approx true}{eq(y,x) \approx true}; \quad \dfrac{eq(x,y) \approx true\ ,\ eq(y,z) \approx true}{eq(x,z) \approx true};$

$\dfrac{eq(x,y) \approx true}{eq(s(x),s(y)) \approx true}; \quad \dfrac{eq(s(x),s(y)) \approx true}{eq(x,y) \approx true}.$

Taking $\tau(x : nat \approx y : nat) = \{eq(x : nat, y : nat) \approx true\}$, we have

$$\text{Nat} \to_\tau \text{NatEq}$$

k-logics

Goal

Provide a suitable context to deal simultaneously with different specification logics as, assertional, equational, modal, ...

- Let Σ be a signature and Va a set of variables for Σ. The set of terms in the variables Va over Σ is denoted by $\mathrm{Fm}_\Sigma(\mathrm{Va})$.

Definition

A k-logic is a pair $\mathcal{L} = \langle \Sigma, \vdash_\mathcal{L} \rangle$, where Σ is a signature and $\vdash_\mathcal{L} \subseteq \mathcal{P}(\mathrm{Fm}^k_\Sigma(\mathrm{Va})) \times \mathrm{Fm}^k_\Sigma(\mathrm{Va})$ a relation such for all $\Gamma \cup \Delta \cup \{\bar{\gamma}, \bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$:

(i) $\Gamma \vdash_\mathcal{L} \bar{\gamma}$ *for each* $\bar{\gamma} \in \Gamma$;

(ii) *if* $\Gamma \vdash_\mathcal{L} \bar{\varphi}$, *and* $\Delta \vdash_\mathcal{L} \bar{\gamma}$ *for each* $\bar{\gamma} \in \Gamma$, *then* $\Delta \vdash_\mathcal{L} \bar{\varphi}$;

(iii) *if* $\Gamma \vdash_\mathcal{L} \bar{\varphi}$, *then* $\sigma(\Gamma) \vdash_\mathcal{L} \sigma(\bar{\varphi})$ *for every substitution* σ.

Semantics

A pair $\mathcal{A} = \langle \mathbf{A}, F \rangle$ is a *k-data structure over* Σ if

- $\mathbf{A}$ is a Σ-algebra over Σ
- F is a subset of A^k.

Semantic consequence

$\Gamma \models_{\mathcal{A}} \bar{\varphi}$ if for any assignment $h : \mathrm{Va} \to A$, $h(\Gamma) \subseteq F$ implies $h(\bar{\varphi}) \in F$.

Familiar examples

1-data structures: models of CPC, e.g. $\mathcal{A} = \langle \mathbf{A}, F \rangle$ over a sentential language with A a Boolean algebra and $F = \{\top\}$;

2-data structures: models of the (free) equational logic over Σ, e.g. $\mathcal{A} = \langle \mathbf{A}, F \rangle$ over a multi-sorted signature with $F = \ id_A$;

Translating k-logics

Definition ((k, m)-translation from Σ to Σ')

$$\tau : \mathrm{Fm}^{k}_{\Sigma}(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^{m}_{\Sigma'}(\mathrm{Va}))$$

Translating k-logics

Definition ((k, m)-translation from Σ to Σ')

$$\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$$

Definition (Interpretation)

τ interprets $\mathcal{L}$ if there is a m-logic $\mathcal{L}'$ over Σ' such that, for any $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$,

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \textit{ iff } \tau(\Gamma) \vdash_{\mathcal{L}'} \tau(\bar{\varphi}).$$

A paradigmatic example

$$\rho(\langle p, q\rangle) = \{\langle p \to q\rangle, \langle q \to p\rangle\}$$

$$\begin{tikzcd}
\color{blue}{\mathrm{BA}} \arrow[r, bend left] & \color{blue}{\mathrm{CPC}} \arrow[l, bend left]
\end{tikzcd}$$

$$\tau(\langle p\rangle) = \{\langle p, \top\rangle\}$$

τ-model class

Definition (τ-model)

Let $\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$ *and* $\mathcal{L}$ *over* Σ*. An I-data structure* $\mathcal{A}$ *is a* *τ-model of $\mathcal{L}$* *if for any* $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$,

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \text{ implies } \tau(\Gamma) \models_{\mathcal{A}} \tau(\bar{\varphi}).$$

$\mathrm{Mod}^\tau(\mathcal{L})$ *denotes the class of all τ-model of $\mathcal{L}$.*

τ-model class

Definition (τ-model)

Let $\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$ *and* $\mathcal{L}$ *over* Σ. *An I-data structure* $\mathcal{A}$ *is a* *τ-model of* $\mathcal{L}$ *if for any* $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$,

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \text{ implies } \tau(\Gamma) \models_{\mathcal{A}} \tau(\bar{\varphi}).$$

$\mathrm{Mod}^\tau(\mathcal{L})$ *denotes the class of all* τ*-model of* $\mathcal{L}$.

Theorem

If τ *interprets* $\mathcal{L}$ *then* $\models_{\mathrm{Mod}^\tau(\mathcal{L})}$ *is the largest* τ*-interpretation of* $\mathcal{L}$.

τ-model class

Definition (τ-model)

Let $\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$ *and* $\mathcal{L}$ *over* Σ*. An I-data structure* $\mathcal{A}$ *is a* *τ-model of $\mathcal{L}$* *if for any* $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$*,*

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \textit{ implies } \tau(\Gamma) \models_{\mathcal{A}} \tau(\bar{\varphi}).$$

$\mathrm{Mod}^\tau(\mathcal{L})$ *denotes the class of all τ-model of $\mathcal{L}$.*

Theorem

If τ *interprets* $\mathcal{L}$ *then* $\models_{\mathrm{Mod}^\tau(\mathcal{L})}$ *is the largest τ-interpretation of $\mathcal{L}$.*

Theorem

Let τ *be a translation that commutes with substitutions. Then if* $\vdash_{\mathcal{L}}$ *is axiomatized by* Φ *then* $\models_{\mathrm{Mod}^\tau(\mathcal{L})}$ *is axiomatized by* $\tau(\Phi)$*.*

(Generalized) refinements by translation

Definition (Refinement via interpretation)

Let $\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$ *be an interpretation of* $\mathcal{L}$*.* $\mathcal{L} \rightharpoonup_\tau \mathcal{L}'$*, if for any* $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$,

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \;\Rightarrow\; \tau(\Gamma) \vdash_{\mathcal{L}'} \tau(\bar{\varphi}).$$

(Generalized) refinements by translation

Definition (Refinement via interpretation)

Let $\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$ *be an interpretation of* $\mathcal{L}$. $\mathcal{L} \rightharpoondown_\tau \mathcal{L}'$, *if for any* $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$,

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \ \Rightarrow \ \tau(\Gamma) \vdash_{\mathcal{L}'} \tau(\bar{\varphi}).$$

Example

I *-*

$$\mathrm{CPC} \rightharpoondown_{id} \mathrm{K}$$

since K is obtained from CPC *by adding* $\Box$ *to the signature, the axiom* $\Box(p \to q) \to (\Box p \to \Box q)$ *and the inference rule* $\dfrac{p}{\Box p}$,

(Generalized) refinements by translation

Definition (Refinement via interpretation)

Let $\tau : \mathrm{Fm}^k_\Sigma(\mathrm{Va}) \to \mathcal{P}(\mathrm{Fm}^m_{\Sigma'}(\mathrm{Va}))$ *be an interpretation of* $\mathcal{L}$*.* $\mathcal{L} \rightharpoonup_\tau \mathcal{L}'$*, if for any* $\Gamma \cup \{\bar{\varphi}\} \subseteq \mathrm{Fm}^k_\Sigma(\mathrm{Va})$,

$$\Gamma \vdash_{\mathcal{L}} \bar{\varphi} \;\Rightarrow\; \tau(\Gamma) \vdash_{\mathcal{L}'} \tau(\bar{\varphi}).$$

Example

I -

$$\mathrm{CPC} \rightharpoonup_{id} \mathrm{K}$$

since K *is obtained from* CPC *by adding* $\Box$ *to the signature, the axiom* $\Box(p \to q) \to (\Box p \to \Box q)$ *and the inference rule* $\dfrac{p}{\Box p}$,

II-

$$\mathrm{CPC} \rightharpoonup_\tau \mathbb{HA} \rightharpoonup_\rho \mathrm{IPC}$$

where $\tau(p) = \{\langle \neg\neg p, \top \rangle\}$ *and* $\rho(\langle p, q \rangle) = \{p \to q, q \to p\}$.

Behavioral specification

Principle

The satisfaction of the requirements does not need to be strict, and may be checked up to a behavioral relation.

Behavioral specification

Principle

The satisfaction of the requirements does not need to be strict, and may be checked up to a behavioral relation.

Context

In the observational approach/modern algebraic specification of abstract data types are split in two types of data representation: the representation types for internal data (data hiding) and the types of representation of the actual data, i.e. the data that we have access direct (visible or observable data).

Behavioral specification

Principle

The satisfaction of the requirements does not need to be strict, and may be checked up to a behavioral relation.

Context

In the observational approach/modern algebraic specification of abstract data types are split in two types of data representation: the representation types for internal data (data hiding) and the types of representation of the actual data, i.e. the data that we have access direct (visible or observable data). The types of hidden data representation are used to represent encapsulated data, which the user has access only via procedures (ie, complex operations) with visible output and taking such data as input.

Behavioral specification

Principle

The satisfaction of the requirements does not need to be strict, and may be checked up to a behavioral relation.

Context

In the observational approach/modern algebraic specification of abstract data types are split in two types of data representation: the representation types for internal data (data hiding) and the types of representation of the actual data, i.e. the data that we have access direct (visible or observable data). The types of hidden data representation are used to represent encapsulated data, which the user has access only via procedures (ie, complex operations) with visible output and taking such data as input.

- Data encapsulation is very important, for security reasons AND to allow effective and fast software updates.

Observational signature

Let $\Sigma = \langle S, \Omega \rangle$ and $\mathrm{Obs} \subseteq S$, *the observational signature Σ w.r.t Obs* is the pair $\langle \Sigma, \mathrm{Obs} \rangle$. The sorts Obs are called *observable sorts*.

Observational signature

Let $\Sigma = \langle S, \Omega \rangle$ and $\mathrm{Obs} \subseteq S$, *the observational signature* Σ *w.r.t* Obs is the pair $\langle \Sigma, \mathrm{Obs} \rangle$. The sorts Obs are called *observable sorts*.

Example

Automata. *The input and the output sorts (In and out) are considered the observable sorts and the state sort* Z *as hidden.*

Observational signature

Let $\Sigma = \langle S, \Omega \rangle$ and $\mathrm{Obs} \subseteq S$, *the observational signature* Σ *w.r.t* Obs is the pair $\langle \Sigma, \mathrm{Obs} \rangle$. The sorts Obs are called *observable sorts*.

Example

Automata. *The input and the output sorts (In and out) are considered the observable sorts and the state sort Z as hidden.*

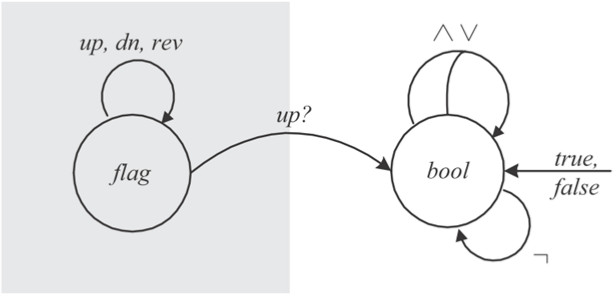
Example

```
Gen
    elt;
    cell;
Obs
    elt;
Op
    put: elt,cell -> cell;
    get:cell -> elt;
Ax

get(put(e,c))=e;
```

Example

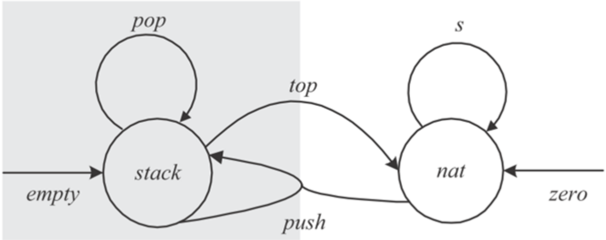
```
Spec FLAGS = enrich BA by
Gen
    flag;
Obs
    bool;
Op
    up: flag -> flag;
    dn: flag -> flag;
    rev: flag -> flag;
    up?: flag -> bool;
```

```
Ax

up?(up(x))=true;
up?(dn(x))=false;
up?(rev(x))=¬(up?(x));
```



Example

```
Spec STACK = = enrich Nat by
Gen
    stack;
Obs
    nat;
Op
    push:nat,stack -> stack;
    pop:stack -> stack;
    top:stack -> nat;
Ax

pop(push(x,s))=s;
top(push(x,s))=x;
```



Observational equality

Definition (Contexts and Observable Contexts)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $X = (X_s)_{s \in S}$ *a family of infinite countable sets of variables (pairwise disjoint) and* $Z = \langle \{z_s\} \rangle_{s \in S}$ *an* S*-singular family of sets (pairwise disjoint) of different variables from the variables in* X*. pausa An s-context over* Σ *is a term* $c \in T(\Sigma, X \cup \{z_s\})_{s'}$*, for some* $s' \in S$*, with at least one occurrence of the variable* z_s*.*

Observational equality

Definition (Contexts and Observable Contexts)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ be an observational signature, $X = (X_s)_{s \in S}$ a family of infinite countable sets of variables (pairwise disjoint) and $Z = \langle \{z_s\} \rangle_{s \in S}$ an S-singular family of sets (pairwise disjoint) of different variables from the variables in X. pausa An s-context over Σ is a term $c \in T(\Sigma, X \cup \{z_s\})_{s'}$, for some $s' \in S$, with at least one occurrence of the variable z_s.
If $s' \in \mathrm{Obs}$, c is called an s-observable context.

Observational equality

Definition (Contexts and Observable Contexts)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $X = (X_s)_{s \in S}$ *a family of infinite countable sets of variables (pairwise disjoint) and* $Z = \langle \{z_s\} \rangle_{s \in S}$ *an* S*-singular family of sets (pairwise disjoint) of different variables from the variables in* X*. pausa An* *s-context* *over* Σ *is a term* $c \in T(\Sigma, X \cup \{z_s\})_{s'}$*, for some* $s' \in S$*, with at least one occurrence of the variable* z_s*.*
If $s' \in \mathrm{Obs}$*,* c *is called an* *s-observable context.*

- **Some Variants:** Γ-contexts.

Observational equality

Definition (Contexts and Observable Contexts)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ be an observational signature, $X = (X_s)_{s \in S}$ a family of infinite countable sets of variables (pairwise disjoint) and $Z = \langle \{z_s\} \rangle_{s \in S}$ an S-singular family of sets (pairwise disjoint) of different variables from the variables in X. pausa An s-context over Σ is a term $c \in T(\Sigma, X \cup \{z_s\})_{s'}$, for some $s' \in S$, with at least one occurrence of the variable z_s. If $s' \in \mathrm{Obs}$, c is called an s-observable context.

- **Some Variants:** Γ-contexts.

Definition (Observational equality)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ be an observational signature and $\mathbf{A}$ a Σ-algebra. $a, a' \in A_s$ are observationally equal w.r.t. Obs, $a \equiv_{\mathbf{A}}^{\mathrm{Obs}} a'$, if for any observable s-context $c(x_1 : s_1, \ldots, x_n : s_n, z_s)$, and every $b_1 \in A_{S_1}, \ldots, b_n \in A_{S_n}$,

$$c^{\mathbf{A}}(b_1, \ldots, b_n, a) = c^{\mathbf{A}}(b_1, \ldots, b_n, a').$$

Observational equality

Definition (Contexts and Observable Contexts)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $X = (X_s)_{s \in S}$ *a family of infinite countable sets of variables (pairwise disjoint) and* $Z = \langle \{z_s\} \rangle_{s \in S}$ *an* S*-singular family of sets (pairwise disjoint) of different variables from the variables in* X*. pausa An s-context over* Σ *is a term* $c \in T(\Sigma, X \cup \{z_s\})_{s'}$*, for some* $s' \in S$*, with at least one occurrence of the variable* z_s*.*
If $s' \in \mathrm{Obs}$*,* c *is called an s-observable context.*

- **Some Variants:** Γ-contexts.

Definition (Observational equality)

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature and* $\mathbf{A}$ *a* Σ*-algebra.* $a, a' \in A_s$ *are observationally equal w.r.t.* Obs*,* $a \equiv_{\mathbf{A}}^{\mathrm{Obs}} a'$*, if for any observable s-context* $c(x_1 : s_1, \ldots, x_n : s_n, z_s)$*, and every* $b_1 \in A_{s_1}, \ldots, b_n \in A_{s_n}$*,*

$$c^{\mathbf{A}}(b_1, \ldots, b_n, a) = c^{\mathbf{A}}(b_1, \ldots, b_n, a').$$

Fact

$\equiv_{\mathbf{A}}^{\mathrm{Obs}}$ is a congruence on $\mathbf{A}$.

Behavioral satisfaction

Definition

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $\mathbf{A}$ *is a* Σ*-algebra and* $t, t' \in T(\Sigma, X)_s$. $\mathbf{A}$ *é is behavioral model of* $t \approx t'$, $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$, *if for any observable s-context* $c(x_1 : s_1, \ldots, x_n : s_n, z_s)$ $\mathbf{A} \models c[t] \approx c[t']$.

Behavioral satisfaction

Definition

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $\mathbf{A}$ *is a* Σ*-algebra and* $t, t' \in T(\Sigma, X)_s$*.* $\mathbf{A}$ *é is behavioral model of* $t \approx t'$*,* $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$*, if for any observable s-context* $c(x_1 : s_1, \ldots, x_n : s_n, z_s)$ $\mathbf{A} \models c[t] \approx c[t']$*.*

- $C \models^{\mathrm{Obs}} t \approx t'$ if for any $\mathbf{A} \in C$ $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$.

Behavioral satisfaction

Definition

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $\mathbf{A}$ *is a* Σ*-algebra and* $t, t' \in T(\Sigma, X)_s$. $\mathbf{A}$ *é is behavioral model of* $t \approx t'$, $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$, *if for any observable s-context* $c(x_1 : s_1, \ldots, x_n : s_n, z_s)$ $\mathbf{A} \models c[t] \approx c[t']$.

- $C \models^{\mathrm{Obs}} t \approx t'$ if for any $\mathbf{A} \in C$ $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$.
- $SP \models^{\mathrm{Obs}} t \approx t'$ if $\mathrm{Mod}(SP) \models^{\mathrm{Obs}} t \approx t'$.

Behavioral satisfaction

Definition

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $\mathbf{A}$ *is a* Σ*-algebra and* $t, t' \in T(\Sigma, X)_s$. $\mathbf{A}$ *é is behavioral model of* $t \approx t'$, $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$, *if for any observable s-context* $c(x_1 : s_1, \ldots, x_n : s_n, z_s)$ $\mathbf{A} \models c[t] \approx c[t']$.

- $C \models^{\mathrm{Obs}} t \approx t'$ if for any $\mathbf{A} \in C$ $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$.
- $SP \models^{\mathrm{Obs}} t \approx t'$ if $\mathrm{Mod}(SP) \models^{\mathrm{Obs}} t \approx t'$.
- $Th^{\mathrm{Obs}}(\mathrm{C}) = \{t \approx t' \in \mathrm{Eq}(\Sigma) | \, C \models^{\mathrm{Obs}} t \approx t'\}$.

Behavioral satisfaction

Definition

Let $\langle \Sigma, \mathrm{Obs} \rangle$ *be an observational signature,* $\mathbf{A}$ *is a* Σ*-algebra and* $t, t' \in T(\Sigma, X)_s$*.* $\mathbf{A}$ *é is behavioral model of* $t \approx t'$, $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$, *if for any observable s-context* $c(x_1 \colon s_1, \ldots, x_n \colon s_n, z_s)$ $\mathbf{A} \models c[t] \approx c[t']$.

- $C \models^{\mathrm{Obs}} t \approx t'$ if for any $\mathbf{A} \in C$ $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$.
- $SP \models^{\mathrm{Obs}} t \approx t'$ if $\mathrm{Mod}(SP) \models^{\mathrm{Obs}} t \approx t'$.
- $Th^{\mathrm{Obs}}(\mathrm{C}) = \{t \approx t' \in \mathrm{Eq}(\Sigma) | \, C \models^{\mathrm{Obs}} t \approx t'\}$.

Theorem

1. $\mathbf{A} \models^{\mathrm{Obs}} t \approx t'$ *iff* $\mathbf{A}/\equiv^{\mathrm{Obs}}_{\mathbf{A}} \models t \approx t'$;
2. $SP \models^{\mathrm{Obs}} t \approx t'$ *iff* $SP^{\mathrm{Obs}} \models t \approx t'$;
3. $Th^{\mathrm{Obs}}(\mathcal{C}) = Th(\mathcal{C}^{\mathrm{Obs}})$,

where $\mathcal{C}^{\mathrm{Obs}} = \{\mathbf{A}/\equiv^{\mathrm{Obs}}_{\mathbf{A}} | \mathbf{A} \in \mathcal{C}\}$ *and* $SP^{\mathrm{Obs}} = \mathrm{Mod}(SP)^{\mathrm{Obs}}$.

Coinduction

Theorem

$\equiv_{\mathbf{A}}^{\mathrm{Obs}}$ *is the largest congruence on* $\mathbf{A}$ *which is the identity in* A_{Obs}. *I.e., if* $\approx$ *is a congruence s.t.* $(\approx)_{\mathrm{Obs}} = \Delta_{A_{\mathrm{Obs}}}$ *(called hidden congruence), then* $\approx \subseteq \equiv_{\mathbf{A}}^{\mathrm{Obs}}$.

Coinduction

Theorem

$\equiv_{\mathbf{A}}^{\mathrm{Obs}}$ *is the largest congruence on* $\mathbf{A}$ *which is the identity in* A_{Obs}*. I.e., if* $\approx$ *is a congruence s.t.* $(\approx)_{\mathrm{Obs}} = \Delta_{A_{\mathrm{Obs}}}$ *(called hidden congruence), then* $\approx \subseteq \equiv_{\mathbf{A}}^{\mathrm{Obs}}$.

Coinduction Method

Let $\langle \Sigma, \mathrm{Obs} \rangle$ be an observational signature and $\mathbf{A}$ a Σ-algebra. $a, a' \in A_s$: To Show that $a \equiv_{\mathbf{A}}^{\mathrm{Obs}} a'$,

Coinduction

Theorem

$\equiv^{\mathrm{Obs}}_{\mathbf{A}}$ is the largest congruence on $\mathbf{A}$ which is the identity in A_{Obs}. I.e., if $\approx$ is a congruence s.t. $(\approx)_{\mathrm{Obs}} = \Delta_{A_{\mathrm{Obs}}}$ (called hidden congruence), then $\approx \subseteq \equiv^{\mathrm{Obs}}_{\mathbf{A}}$.

Coinduction Method

Let $\langle \Sigma, \mathrm{Obs} \rangle$ be an observational signature and $\mathbf{A}$ a Σ-algebra. $a, a' \in A_s$: To Show that $a \equiv^{\mathrm{Obs}}_{\mathbf{A}} a'$, do

1. Define an appropriated binary relation R on A;
2. Show that R is an hidden congruence;
3. Show that $a\ R\ a'$.

Example

In $\mathcal{L}_{Flags}$ we have that $rev^{\mathbf{A}}(rev^{\mathbf{A}}(a)) \equiv^{\mathbf{A}}_{\mathrm{Obs}} a$. However, $rev(rev(x)) \approx x$ is not an equational consequence of the specification $\mathcal{L}_{Flags}$.

An example

```
bth SET[X :: TRIV] is sort Set .
  op empty : -> Set .
  op _in_  : Elt Set -> Bool .
  op add : Elt Set -> Set .
  ops (_U_) (_&_) : Set Set -> Set .
  vars E E' : Elt .  vars S S' : Set .
  eq E in empty = false .
  eq E in add(E' , S) = (E == E') or (E in S).
  eq E in S U S' = (E in S)  or (E in S') .
  eq E in S & S' = (E in S) and (E in S') .
end
```

An example

```
bth SET[X :: TRIV] is sort Set .
  op empty : -> Set .
  op _in_  : Elt Set -> Bool .
  op add : Elt Set -> Set .
  ops (_U_) (_&_) : Set Set -> Set .
  vars E E' : Elt .  vars S S' : Set .
  eq E in empty = false .
  eq E in add(E' , S) = (E == E') or (E in S).
  eq E in S U S' = (E in S)  or (E in S') .
  eq E in S & S' = (E in S) and (E in S') .
end
```

Some equations are consequences of the specification (use CafeOBJ).

$$E \text{ in } (S\&(S'US)) \approx E \text{ in } S$$

An example

```
bth SET[X :: TRIV] is sort Set .
  op empty : -> Set .
  op _in_  : Elt Set -> Bool .
  op add : Elt Set -> Set .
  ops (_U_) (_&_) : Set Set -> Set .
  vars E E' : Elt .  vars S S' : Set .
  eq E in empty = false .
  eq E in add(E' , S) = (E == E') or (E in S).
  eq E in S U S' = (E in S)  or (E in S') .
  eq E in S & S' = (E in S) and (E in S') .
end
```

Some equations are consequences of the specification (use CafeOBJ).

$$E \text{ in } (S\&(S'US)) \approx E \text{ in } S$$

And some others are not!

$$(S\&(S'US)) \approx S$$

An example

```
bth SET[X :: TRIV] is sort Set .
  op empty : -> Set .
  op _in_  : Elt Set -> Bool .
  op add : Elt Set -> Set .
  ops (_U_) (_&_) : Set Set -> Set .
  vars E E' : Elt .  vars S S' : Set .
  eq E in empty = false .
  eq E in add(E' , S) = (E == E') or (E in S).
  eq E in S U S' = (E in S)  or (E in S') .
  eq E in S & S' = (E in S) and (E in S') .
end
```

Some equations are consequences of the specification (use CafeOBJ).

$$E \text{ in } (S\&(S'US)) \approx E \text{ in } S$$

And some others are not!

$$(S\&(S'US)) \approx S$$

However it is behavioral valid. Use the following relation

$$S\ R\ S \quad iff \quad \forall e \quad e \text{ in} S\ iff\ e \text{ in } S'$$

Complementary topics

Related issues:

- Behavioral refinement

Complementary topics

Related issues:

- Behavioral refinement
- Definability of the observational equality

Complementary topics

Related issues:

- Behavioral refinement
- Definability of the observational equality
- Semi-automatic provers for behavioral requirements

Complementary topics

Related issues:

- Behavioral refinement
- Definability of the observational equality
- Semi-automatic provers for behavioral requirements
- Calculus for structured behavioral specifications