Formal Specification and Verification

Classical logic (6)

19.11.2018

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Until now

- First order logic

  Syntax:

  (Many sorted) Signature

  Terms, Formulae

  Substitutions

  Semantics:

  Σ-structures

  Models, Validity, Satisfiability

  Entailment, Equivalence

  Theories

Logical theories

Syntactic view

first-order theory: given by a set $\mathcal{F}$ of (closed) first-order Σ-formulae.

the models of $\mathcal{F}$: $\quad \mathsf{Mod}(\mathcal{F}) = \{\mathcal{A} \in \Sigma\text{-alg} \mid \mathcal{A} \models G, \text{ for all } G \text{ in } \mathcal{F}\}$

Semantic view

given a class $\mathcal{M}$ of Σ-algebras

the first-order theory of $\mathcal{M}$: $\mathsf{Th}(\mathcal{M}) = \{G \in F_\Sigma(X) \text{ closed} \mid \mathcal{M} \models G\}$

Examples

1. Groups

Let $\Sigma = (\{e/0, */2, i/1\}, \emptyset)$

Let $\mathcal{F}$ consist of all (universally quantified) group axioms:

$$\begin{array}{ll} \forall x, y, z & x * (y * z) \approx (x * y) * z \\ \forall x & x * i(x) \approx e \quad \wedge \quad i(x) * x \approx e \\ \forall x & x * e \approx x \quad \wedge \quad e * x \approx x \end{array}$$

Every group $\mathcal{G} = (G, e_G, *_G, i_G)$ is a model of $\mathcal{F}$

Mod($\mathcal{F}$) is the class of all groups

$\mathcal{F} \subset \mathrm{Th}(\mathrm{Mod}(\mathcal{F}))$

Examples

2. Linear (positive)integer arithmetic

Let $\Sigma = (\{0/0, s/1, +/2\}, \{\leq /2\})$

Let $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +, \leq)$ the standard interpretation of integers.

$\{\mathbb{Z}_+\} \subset \mathsf{Mod}(\mathsf{Th}(\mathbb{Z}_+))$

3. Uninterpreted function symbols

Let $\Sigma = (\Omega, \Pi)$ be arbitrary

Let $\mathcal{M} = \Sigma$-alg be the class of all Σ-structures

The theory of uninterpreted function symbols is Th(Σ-alg) the family of all first-order formulae which are true in all Σ-algebras.

Examples

4. Lists

Let $\Sigma = (\{\mathsf{car}/1, \mathsf{cdr}/1, \mathsf{cons}/2\}, \emptyset)$

Let $\mathcal{F}$ be the following set of list axioms:

$$\begin{aligned} \mathsf{car}(\mathsf{cons}(x, y)) &\approx x \\ \mathsf{cdr}(\mathsf{cons}(x, y)) &\approx y \\ \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) &\approx x \end{aligned}$$

$\mathrm{Mod}(\mathcal{F})$ class of all models of $\mathcal{F}$

$\mathrm{Th}_{\mathrm{Lists}} = \mathrm{Th}(\mathrm{Mod}(\mathcal{F}))$ theory of lists (axiomatized by $\mathcal{F}$)

“Most general” models

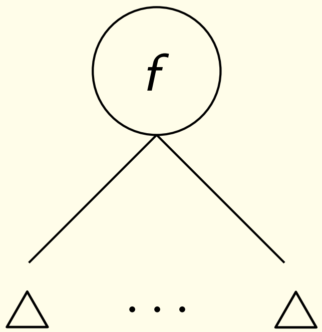
We assume that $\Pi = \emptyset$.

Term algebras

A term algebra (over Σ) is a Σ-algebra $\mathcal{A}$ such that

- $U_{\mathcal{A}} = \mathsf{T}_{\Sigma}$ ($=$ the set of ground terms over Σ)
- $f_{\mathcal{A}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n),\ \ f/n \in \Omega$

$$f_{\mathcal{A}}(\triangle, \ldots, \triangle) =$$



Term algebras

In other words, *values are fixed* to be ground terms and *functions are fixed* to be the term constructors.

Free algebras

Let $\mathcal{K}$ be the class of Σ-algebras which satisfy a set of axioms which are either equalities

$$\forall x : t(x) \approx s(x)$$

or implications:

$$\forall x : t_1(x) \approx s_1(x) \wedge \cdots \wedge t_n(x) \approx s_n(x) \to t(x) \approx s(x)$$

We can construct the "most general" model in $\mathcal{K}$:

- Construct the term algebra $T_\Sigma(X)$ (resp. T_Σ)
- Identify all terms t, t' such that $\mathcal{K} \models t \approx t'$
  (all terms which become equal as a consequence of the axioms).

  $\sim$ congruence relation

  Construct the algebra of equivalence classes: $T_\Sigma(X)/\sim$ (resp. $T_\Sigma/\sim$)
- $T_\Sigma(X)/\sim$ is the free algebra in $\mathcal{K}$ freely generated by X.

  $T_\Sigma/\sim$ is the free algebra in $\mathcal{K}$.

Universal property of the free algebras

For every $\mathcal{A} \in \mathcal{K}$ and every $\beta : X \to \mathcal{A}$ there exists a unique extension β' of β which is an algebra homomorphism:

$$\beta' : T_\Sigma(X)/\sim \to \mathcal{A}$$

Examples

$T_\Sigma(X)$ is the free algebra freely generated by X for the class of all algebras of type Σ.

Let X be a set of symbols and X^* be the class of all finite strings of elements in X, including the empty string.

We construct the monoid $(X^*, \cdot, 1)$ by defining $\cdot$ to be concatenation, and 1 is the empty string.

$(X^*, \cdot, 1)$ is the free monoid freely generated by X.

Formal specification

- Specification for program/system
- Specification for properties of program/system

Verification tasks:

Check that the specification of the program/system has the required properties.

Formal specification

- Specification languages for describing programs/processes/systems

- Specification languages for properties of programs/processes/systems

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

  Axiom-based specification

  Declarative specifications

- **Specification languages for properties of programs/processes/systems**

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

  transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

  Declarative specifications

- **Specification languages for properties of programs/processes/systems**

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

    transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

    algebraic specification

  Declarative specifications

- **Specification languages for properties of programs/processes/systems**

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

  transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

  algebraic specification

  Declarative specifications

  logic based languages (Prolog)

  functional languages, λ-calculus (Scheme, Haskell, OCaml, ...)

  rewriting systems (very close to algebraic specification): ELAN, SPIKE, ...

- **Specification languages for properties of programs/processes/systems**

Formal specification

- **Specification languages for describing programs/processes/systems**

  Model based specification

  transition systems, abstract state machines, specifications based on set theory

  Axiom-based specification

  algebraic specification

  Declarative specifications

  logic based languages (Prolog)

  functional languages, λ-calculus (Scheme, Haskell, OCaml)

  rewriting systems (very close to algebraic specification): ELAN, SPIKE

- **Specification languages for properties of programs/processes/systems**

  Temporal logic

Algebraic specification

- appropriate for specifying the interface of a module or class
- enables verification of implementation w.r.t. specification
- for every ADT operation: argument and result types (sorts)
- semantic equations over operations (axioms) e.g. for every combination of "defined function" (e.g. top, pop) and constructor with the corresponding sort (e.g. push, empty)
- problem: consistency?, completeness?

Example: Algebraic specification

```
fmod NATSTACK is
  sorts Stack .
  protecting NAT .
  op empty : -> Stack .
  op push : Nat Stack -> Stack .
  op pop : Stack -> Stack .
  op top : Stack -> Nat .
  op length : Stack -> Nat .
  var S S2 : Stack .
  var X Y : Element .
  eq pop(push(X,S)) = S .
  eq top(push(X,S)) = X .
  eq length(empty) = 0 .
  eq length(push(X,S)) =
           1 + length(S) .
endfm
```

Example: Algebraic specification

```
reduce pop(push(X,S)) == S .

reduce top(pop(push(X,push(Y,S)))) == Y .

reduce S == push(X,S2) implies push(top(S),pop(S)) == S .

reduce S == push(X,S2) implies length(pop(S)) + 1 == length(S) .
```

- the equations can be used as term rewriting rules
- this allows proving properties of the specification

Syntax of Algebraic Specifications

Signatures: as in FOL (S, Ω, Π)

Example:

$$
\begin{aligned}
STACK = (\quad & \{Stack, Nat\}, \\
& \{\mathrm{empty} : \epsilon \to Stack, \\
& \ \mathrm{push} : Nat \times Stack \to Stack, \\
& \ \mathrm{pop} : Stack \to Stack, \\
& \ \mathrm{top} : Stack \to Nat, \\
& \ \mathrm{length} : Stack \to Nat, \\
& \ 0 : \epsilon \to Nat, 1 : \epsilon \to Nat \\
& \}
\end{aligned}
$$

Semantics of Algebraic Specifications

Σ-algebras

Observations

- different Σ-algebras are not necessarily "equivalent"
- we seek the most "abstract" Σ-algebra,
  since it anticipates as little implementation decisions as possible

Semantics of Algebraic Specifications

Σ-algebras

Observations

- different Σ-algebras are not necessarily "equivalent"
- we seek the most "abstract" Σ-algebra,
  since it anticipates as little implementation decisions as possible

No equations: Term algebras

Equations/Horn clauses: free algebras

$T_\Sigma / \sim$, where

$t \sim t'$ iff
$Ax \models t \approx t'$ iff
For every $\mathcal{A} \in \mathrm{Mod}(Ax)$, $\mathcal{A} \models t \approx t'$

Algebraic Specification

“A gentle introduction to CASL”

M. Bidoit and P. Mosses

`http://www.lsv.ens-cachan.fr/~bidoit/GENTLE.pdf`

(cf. also the slides of the lecture available online)

A subset of the slides was discussed today.