Formal Specification and Verification

Classical logic (4)

13.05.2014

Viorica Sofronie-Stokkermans

e-mail: sofronie@uni-koblenz.de

Limitations of Propositional Logic

- Fixed, finite number of objects
  Cannot express: let G be group with arbitrary number of elements
- No functions or relations with arguments
  Can express: finite function/relation table p_{ij}
  Cannot express: properties of function/relation on all arguments, e.g., $+$ is associative
- Static interpretation
  Programs change value of their variables, e.g., via assignment, call, etc.
  Propositional formulas look at one single interpretation at a time

Beyond the Limitations of Propositional Logic

- First order logic

  (+ functions)

- Temporal logic

  (+ computations)

- Dynamic logic

  (+ computations + functions)

Beyond the Limitations of Propositional Logic

- First order logic

  (+ functions)

- Temporal logic

  (+ computations)

- Dynamic logic

  (+ computations + functions)

Part 2: First-Order Logic

Syntax:

- non-logical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent)
  $\Rightarrow$ Boolean combinations, quantifiers

Signature

A signature $\Sigma = (\Omega, \Pi)$, fixes an alphabet of non-logical symbols, where

- Ω is a set of function symbols f with arity $n \geq 0$ (written f/n)
- Π is a set of predicate symbols p with arity $m \geq 0$ (written p/m)

If $n = 0$ then f is also called a constant (symbol).
If $m = 0$ then p is also called a propositional variable.

Many-sorted Signature A many-sorted signature $\Sigma = (S, \Omega, \Pi)$, fixes an alphabet of non-logical symbols, where

- S is a set of sorts,
- Ω is a set of function symbols f with arity $a(f) = s_1 \dots s_n \to s$,
- Π is a set of predicate symbols p with arity $a(p) = s_1 \dots s_m$

where $s_1, \dots, s_n, s_m, s$ are sorts.

Variables

Predicate logic admits the formulation of abstract, schematic assertions. (Object) variables are the technical tool for schematization.

We assume that

$$X$$

is a given countably infinite set of symbols which we use for (the denotation of) variables.

Many-sorted case:

We assume that for every sort $s \in S$, X_s is a given countably infinite set of symbols which we use for (the denotation of) variables of sort s.

Terms

Terms over Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$
\begin{array}{llll}
t, u, v & ::= & x & , x \in X & \text{(variable)} \\
 & \mid & f(t_1, ..., t_n) & , f/n \in \Omega & \text{(functional term)}
\end{array}
$$

By $\mathrm{T}_\Sigma(X)$ we denote the set of Σ-terms (over X).
A term not containing any variable is called a ground term.
By T_Σ we denote the set of Σ-ground terms.

Many-sorted case:

a variable $x \in X_s$ is a term of sort s

if $a(f) = s_1 \ldots s_n \to s$, and t_i are terms of sort s_i, $i = 1, \ldots, n$ then $f(t_1, ..., t_n)$ is a term of sort s.

Atoms

Atoms (also called atomic formulas) over Σ are formed according to this syntax:

$$
\begin{array}{rcll}
A, B & ::= & p(t_1, ..., t_m) & , \ p/m \in \Pi \\
\left[\quad \right. & \mid & (t \approx t') & \text{(equation)} \left. \quad \right]
\end{array}
$$

Whenever we admit equations as atomic formulas we are in the realm of first-order logic with equality. Admitting equality does not really increase the expressiveness of first-order logic, (cf. exercises). But deductive systems where equality is treated specifically can be much more efficient.

Many-sorted case:

If $a(p) = s_1 \ldots s_m$, we require that t_i is a term of sort s_i for $i = 1, \ldots, m$.

Literals, Clauses

Literals

$$L \quad ::= \quad A \qquad \text{(positive literal)}$$
$$\quad\;\; \mid \quad \neg A \qquad \text{(negative literal)}$$

Clauses

$$C, D \quad ::= \quad \bot \qquad \text{(empty clause)}$$
$$\qquad \mid \quad L_1 \vee \ldots \vee L_k, \;\; k \geq 1 \qquad \text{(non-empty clause)}$$

General First-Order Formulas

$F_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{llll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& \mid & \top & \text{(verum)} \\
& \mid & A & \text{(atomic formula)} \\
& \mid & \neg F & \text{(negation)} \\
& \mid & (F \land G) & \text{(conjunction)} \\
& \mid & (F \lor G) & \text{(disjunction)} \\
& \mid & (F \rightarrow G) & \text{(implication)} \\
& \mid & (F \leftrightarrow G) & \text{(equivalence)} \\
& \mid & \forall x F & \text{(universal quantification)} \\
& \mid & \exists x F & \text{(existential quantification)}
\end{array}
$$

Example: Peano Arithmetic

Signature:

$$\Sigma_{PA} = (\Omega_{PA},\ \Pi_{PA})$$
$$\Omega_{PA} = \{0/0,\ +/2,\ */2,\ s/1\}$$
$$\Pi_{PA} = \{\leq/2,\ </2\}$$

$+, *, <, \leq$ infix; $* >_p + >_p < >_p \leq$

Examples of formulas over this signature are:

$$\forall x, y(x \leq y \leftrightarrow \exists z(x + z \approx y))$$
$$\exists x \forall y(x + y \approx y)$$
$$\forall x, y(x * s(y) \approx x * y + x)$$
$$\forall x, y(s(x) \approx s(y) \rightarrow x \approx y)$$
$$\forall x \exists y(x < y \land \neg \exists z(x < z \land z < y))$$

Example: Specifying LISP lists

Signature:

$\Sigma_{\mathsf{Lists}} = (\Omega_{\mathsf{Lists}}, \Pi_{\mathsf{Lists}})$

$\Omega_{\mathsf{Lists}} = \{\mathsf{car}/1, \mathsf{cdr}/1, \mathsf{cons}/2\}$

$\Pi_{\mathsf{Lists}} = \emptyset$

Examples of formulae:

$\forall x, y \;\; \mathsf{car}(\mathsf{cons}(x, y)) \approx x$

$\forall x, y \;\; \mathsf{cdr}(\mathsf{cons}(x, y)) \approx y$

$\forall x \quad \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) \approx x$

Many-sorted signatures

Example:

Signature

$S = \{\text{array}, \text{index}, \text{element}\}$ set of sorts

$\Omega = \{\text{read}, \text{write}\}$

$$a(\text{read}) = \text{array} \times \text{index} \to \text{element}$$
$$a(\text{write}) = \text{array} \times \text{index} \times \text{element} \to \text{array}$$

$\Pi = \emptyset$

$X = \{X_s \mid s \in S\}$

Examples of formulae:

$\forall x : \text{array} \;\; \forall i : \text{index} \;\; \forall j : \text{index} \;\; (i \approx j \to \text{write}(x, i, \text{read}(x, j)) \approx x)$

$\forall x : \text{array} \; \forall y : \text{array} \;\; (x \approx y \leftrightarrow \forall i : \text{index} \;\; (\text{read}(x, i) \approx \text{read}(y, i)))$

Bound and Free Variables

In QxF, $Q \in \{\exists, \forall\}$, we call F the scope of the quantifier Qx.

An *occurrence* of a variable x is called bound, if it is inside the scope of a quantifier Qx.

Any other occurrence of a variable is called free.

Formulas without free variables are also called closed formulas or sentential forms.

Formulas without variables are called ground.

Bound and Free Variables

Example:

$$\forall\, y \quad \overbrace{(\forall\, \overbrace{x \quad p(x)}^{\textit{scope}} \quad \to \quad q(x, y))}^{\textit{scope}}$$

The occurrence of y is bound, as is the first occurrence of x. The second occurrence of x is a free occurrence.

Substitutions

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic.

In general, substitutions are mappings

$$\sigma : X \to \mathsf{T}_\Sigma(X)$$

such that the domain of σ, that is, the set

$$dom(\sigma) = \{x \in X \mid \sigma(x) \neq x\},$$

is finite. The set of variables introduced by σ, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in dom(\sigma)$, is denoted by $codom(\sigma)$.

Substitutions

Substitutions are often written as $[s_1/x_1, \ldots, s_n/x_n]$, with x_i pairwise distinct, and then denote the mapping

$$[s_1/x_1, \ldots, s_n/x_n](y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

The modification of a substitution σ at x is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$

Why Substitution is Complicated

We define the application of a substitution σ to a term t or formula F by structural induction over the syntactic structure of t or F by the equations depicted on the next page.

In the presence of quantification it is surprisingly complex:
We need to make sure that the (free) variables in the codomain of σ are not *captured* upon placing them into the scope of a quantifier Qy, hence the bound variable must be renamed into a "fresh", that is, previously unused, variable z.

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$
\begin{aligned}
f(s_1, \ldots, s_n)\sigma &= f(s_1\sigma, \ldots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
p(s_1, \ldots, s_n)\sigma &= p(s_1\sigma, \ldots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg F\sigma &= \neg(F\sigma) \\
(F \rho G)\sigma &= (F\sigma \, \rho \, G\sigma) \text{ ; } \quad \text{for each binary connective } \rho \\
(Qx\, F)\sigma &= Qz\, (F\, \sigma[x \mapsto z]) \text{ ; } \quad \text{with } z \text{ a fresh variable}
\end{aligned}
$$

2.2 Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

As in the propositional case, we use a two-valued logic with truth values "true" and "false" denoted by 1 and 0, respectively.

Structures

A Σ-algebra (also called Σ-interpretation or Σ-structure) is a triple

$$\mathcal{A} = (U,\ (f_{\mathcal{A}} : U^n \to U)_{f/n \in \Omega},\ (p_{\mathcal{A}} \subseteq U^m)_{p/m \in \Pi})$$

where $U \neq \emptyset$ is a set, called the universe of $\mathcal{A}$.

Normally, by abuse of notation, we will have $\mathcal{A}$ denote both the algebra and its universe.

By $\Sigma - \mathsf{Alg}$ we denote the class of all Σ-algebras.

Many-sorted Structures

A many-sorted Σ-algebra (also called Σ-interpretation or Σ-structure), where $\Sigma = (S, \Omega, \Pi)$ is a triple

$$\mathcal{A} = (\{U_s\}_{s \in S},\ (f_{\mathcal{A}} : U_{s_1} \times \ldots \times U_{s_n} \to U_s)_{\substack{f \in \Omega, \\ a(f) = s_1 \ldots s_n \to s}}\ (p_{\mathcal{A}} : U_{s_1} \times \ldots \times U_{s_m} \to \{0, 1\})_{\substack{p \in \Pi \\ a(p) = s_1 \ldots s_m}}\ )$$

where $U \neq \emptyset$ is a set, called the universe of $\mathcal{A}$.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A (variable) assignment, also called a valuation (over a given Σ-algebra $\mathcal{A}$), is a map $\beta : X \to \mathcal{A}$.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A (variable) assignment, also called a valuation (over a given Σ-algebra $\mathcal{A}$), is a map $\beta : X \to \mathcal{A}$.

Many-sorted case:

$\beta = \{\beta_s\}_{s \in S}, \beta_s : X_s \to U_s$

Value of a Term in $\mathcal{A}$ with Respect to β

By structural induction we define

$$\mathcal{A}(\beta) : \mathsf{T}_\Sigma(X) \to \mathcal{A}$$

as follows:

$$\begin{aligned} \mathcal{A}(\beta)(x) &= \beta(x), \qquad x \in X \\ \mathcal{A}(\beta)(f(s_1, \ldots, s_n)) &= f_{\mathcal{A}}(\mathcal{A}(\beta)(s_1), \ldots, \mathcal{A}(\beta)(s_n)), \qquad f/n \in \Omega \end{aligned}$$

Value of a Term in $\mathcal{A}$ with Respect to β

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to \mathcal{A}$, for $x \in X$ and $a \in \mathcal{A}$, denote the assignment

$$\beta[x \mapsto a](y) := \begin{cases} a & \text{if } x = y \\ \beta(y) & \text{otherwise} \end{cases}$$

Truth Value of a Formula in $\mathcal{A}$ with Respect to β

$\mathcal{A}(\beta) : \mathsf{F}_\Sigma(X) \to \{0,1\}$ is defined inductively as follows:

$$
\begin{aligned}
\mathcal{A}(\beta)(\bot) &= 0 \\
\mathcal{A}(\beta)(\top) &= 1 \\
\mathcal{A}(\beta)(p(s_1,\dots,s_n)) &= p_{\mathcal{A}}(\mathcal{A}(\beta)(s_1),\dots,\mathcal{A}(\beta)(s_n)) \\
\mathcal{A}(\beta)(s \approx t) = 1 &\quad\Leftrightarrow\quad \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \\
\mathcal{A}(\beta)(\neg F) = 1 &\quad\Leftrightarrow\quad \mathcal{A}(\beta)(F) = 0 \\
\mathcal{A}(\beta)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
& \text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho \\
\mathcal{A}(\beta)(\forall x F) &= \min_{a \in U} \{\mathcal{A}(\beta[x \mapsto a])(F)\} \\
\mathcal{A}(\beta)(\exists x F) &= \max_{a \in U} \{\mathcal{A}(\beta[x \mapsto a])(F)\}
\end{aligned}
$$

Example

The "Standard" Interpretation for Peano Arithmetic:

$$
\begin{array}{rcl}
U_{\mathbb{N}} & = & \{0, 1, 2, \ldots\} \\
0_{\mathbb{N}} & = & 0 \\
s_{\mathbb{N}} : U_{\mathbb{N}} \rightarrow U_{\mathbb{N}} & & s_{\mathbb{N}}(n) = n + 1 \\
+_{\mathbb{N}} : U_{\mathbb{N}}^2 \rightarrow U_{\mathbb{N}} & & +_{\mathbb{N}}(n, m) = n + m \\
*_{\mathbb{N}} : U_{\mathbb{N}}^2 \rightarrow U_{\mathbb{N}} & & *_{\mathbb{N}}(n, m) = n * m \\
\leq_{\mathbb{N}}: U_{\mathbb{N}}^2 \rightarrow \{0, 1\} & & \leq_{\mathbb{N}} (n, m) = 1 \text{ iff } n \text{ less than or equal to } m \\
<_{\mathbb{N}}: U_{\mathbb{N}}^2 \rightarrow \{0, 1\} & & \leq_{\mathbb{N}} (n, m) = 1 \text{ iff } n \text{ less than } m
\end{array}
$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Example

Values over $\mathbb{N}$ for Sample Terms and Formulas:

Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$
\begin{aligned}
\mathbb{N}(\beta)(s(x) + s(0)) &= 3 \\
\mathbb{N}(\beta)(x + y \approx s(y)) &= 1 \\
\mathbb{N}(\beta)(\forall x, y(x + y \approx y + x)) &= 1 \\
\mathbb{N}(\beta)(\forall z\ z \leq y) &= 0 \\
\mathbb{N}(\beta)(\forall x \exists y\ x < y) &= 1
\end{aligned}
$$

2.3 Models, Validity, and Satisfiability

F is valid in $\mathcal{A}$ under assignment β:

$$\mathcal{A}, \beta \models F \quad :\Leftrightarrow \quad \mathcal{A}(\beta)(F) = 1$$

F is valid in $\mathcal{A}$ ($\mathcal{A}$ is a model of F):

$$\mathcal{A} \models F \quad :\Leftrightarrow \quad \mathcal{A}, \beta \models F, \text{ for all } \beta \in X \to U_{\mathcal{A}}$$

F is valid (or is a tautology):

$$\models F \quad :\Leftrightarrow \quad \mathcal{A} \models F, \text{ for all } \mathcal{A} \in \Sigma\text{-alg}$$

F is called satisfiable iff there exist $\mathcal{A}$ and β such that $\mathcal{A}, \beta \models F$.
Otherwise F is called unsatisfiable.

Entailment and Equivalence

F entails (implies) G (or G is a consequence of F), written $F \models G$

:$\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-alg and $\beta \in X \to U_{\mathcal{A}}$,
whenever $\mathcal{A}, \beta \models F$ then $\mathcal{A}, \beta \models G$.

F and G are called equivalent

:$\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-alg und $\beta \in X \to U_{\mathcal{A}}$ we have
$\mathcal{A}, \beta \models F \quad \Leftrightarrow \quad \mathcal{A}, \beta \models G$.

Entailment and Equivalence

Proposition 2.6:
F entails G iff $(F \rightarrow G)$ is valid

Proposition 2.7:
F and G are equivalent iff $(F \leftrightarrow G)$ is valid.

Extension to sets of formulas N in the "natural way", e.g., $N \models F$

$:\Leftrightarrow$ for all $\mathcal{A} \in \Sigma$-alg and $\beta \in X \rightarrow U_{\mathcal{A}}$:
if $\mathcal{A}, \beta \models G$, for all $G \in N$, then $\mathcal{A}, \beta \models F$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 2.8:

$$F \text{ valid} \quad \Leftrightarrow \quad \neg F \text{ unsatisfiable}$$

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

Q: In a similar way, entailment $N \models F$ can be reduced to unsatisfiability. How?

Algorithmic Problems

Validity(F): $\models F$?

Satisfiability(F): F satisfiable?

Entailment(F,G): does F entail G?

Model($\mathcal{A}$,F): $\mathcal{A} \models F$?

Solve($\mathcal{A}$,F): find an assignment β such that $\mathcal{A}, \beta \models F$

Solve(F): find a substitution σ such that $\models F\sigma$

Abduce(F): find G with "certain properties" such that G entails F

Decidability/Undecidability

In 1931, Gödel published his incompleteness theorems in

> "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme"

(in English "On Formally Undecidable Propositions of Principia Mathematica and Related Systems").

He proved for any computable axiomatic system that is powerful enough to describe the arithmetic of the natural numbers (e.g. the Peano axioms or Zermelo-Fraenkel set theory with the axiom of choice), that:

- If the system is consistent, it cannot be complete.
- The consistency of the axioms cannot be proven within the system.

Decidability/Undecidability

These theorems ended a half-century of attempts, beginning with the work of Frege and culminating in Principia Mathematica and Hilbert's formalism, to find a set of axioms sufficient for all mathematics.

The incompleteness theorems also imply that not all mathematical questions are computable.

Consequences of Gödel's Famous Theorems

1. For most signatures Σ, validity is undecidable for Σ-formulas. (One can easily encode Turing machines in most signatures.)
2. For each signature Σ, the set of valid Σ-formulas is recursively enumerable.
   (This is proved by giving complete deduction systems.)
3. For $\Sigma = \Sigma_{PA}$ and $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the theory $Th(\mathbb{N}_*)$ is not recursively enumerable.

These undecidability results motivate the study of subclasses of formulas (fragments) of first-order logic

Q: Can you think of any fragments of first-order logic for which validity is decidable?

Some Decidable Fragments/Problems

Validity/Satisfiability/Entailment: Some decidable fragments:

- Variable-free formulas without equality:
  satisfiability is NP-complete. (why?)
- Variable-free Horn clauses (clauses with at most one positive atom): entailment is decidable in linear time.
- Monadic class: no function symbols, all predicates unary;
  validity is NEXPTIME-complete.
- Q: Other decidable fragments of FOL (with variables)?
  Which methods for proving decidability?

Decidable problems.

Finite model checking is decidable in time polynomial in the size of the structure and the formula.

Calculi

There exist Hilbert style calculi and sequent calculi for first-order logic.

Checking satisfiability of formulae:

- Resolution
- Semantic tableaux

Verification: Logical theories

Theory of a Structure

Let $\mathcal{A} \in \Sigma$-alg. The (first-order) theory of $\mathcal{A}$ is defined as

$$Th(\mathcal{A}) = \{G \in \mathsf{F}_\Sigma(X) \mid \mathcal{A} \models G\}$$

Problem of axiomatizability:

For which structures $\mathcal{A}$ can one axiomatize $Th(\mathcal{A})$, that is, can one write down a formula F (or a recursively enumerable set F of formulas) such that

$$Th(\mathcal{A}) = \{G \mid F \models G\}?$$

Analogously for sets of structures.

Two Interesting Theories

Let $\Sigma_{Pres} = (\{0/0, s/1, +/2\}, \ \emptyset)$ and $\mathbb{Z}_+ = (\mathbb{Z}, 0, s, +)$ its standard interpretation on the integers.

$Th(\mathbb{Z}_+)$ is called Presburger arithmetic (M. Presburger, 1929).

(There is no essential difference when one, instead of $\mathbb{Z}$, considers the natural numbers $\mathbb{N}$ as standard interpretation.)

Presburger arithmetic is decidable in 3EXPTIME (D. Oppen, JCSS, 16(3):323–332, 1978), and in 2EXPSPACE, using automata-theoretic methods (and there is a constant $c \geq 0$ such that $Th(\mathbb{Z}_+) \notin \mathrm{NTIME}(2^{2^{cn}})$).

Two Interesting Theories

However, $\mathbb{N}_* = (\mathbb{N}, 0, s, +, *)$, the standard interpretation of $\Sigma_{PA} = (\{0/0, s/1, +/2, */2\}, \emptyset)$, has as theory the so-called Peano arithmetic which is undecidable, not even recursively enumerable.

Note: The choice of signature can make a big difference with regard to the computational complexity of theories.

Logical theories

Syntactic view

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

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

Semantic view

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

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

Theories

$\mathcal{F}$ set of (closed) first-order formulae

$$\mathrm{Mod}(\mathcal{F}) = \{A \in \Sigma\text{-alg} \mid \mathcal{A} \models G, \text{ for all } G \text{ in } \mathcal{F}\}$$

$\mathcal{M}$ class of Σ-algebras

$$\mathrm{Th}(\mathcal{M}) = \{G \in F_\Sigma(X) \text{ closed} \mid \mathcal{M} \models G\}$$

$\mathrm{Th}(\mathrm{Mod}(\mathcal{F}))$ the set of formulae true in all models of $\mathcal{F}$
represents exactly the set of consequences of $\mathcal{F}$

Theories

$\mathcal{F}$ set of (closed) first-order formulae

$$\mathrm{Mod}(\mathcal{F}) = \{A \in \Sigma\text{-alg} \mid \mathcal{A} \models G, \text{ for all } G \text{ in } \mathcal{F}\}$$

$\mathcal{M}$ class of Σ-algebras

$$\mathrm{Th}(\mathcal{M}) = \{G \in F_\Sigma(X) \text{ closed} \mid \mathcal{M} \models G\}$$

$\mathrm{Th}(\mathrm{Mod}(\mathcal{F}))$ the set of formulae true in all models of $\mathcal{F}$
represents exactly the set of consequences of $\mathcal{F}$

Note: $\mathcal{F} \subseteq \mathrm{Th}(\mathrm{Mod}(\mathcal{F}))$ (typically strict)

$\mathcal{M} \subseteq \mathrm{Mod}(\mathrm{Th}(\mathcal{M}))$ (typically strict)

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

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

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

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

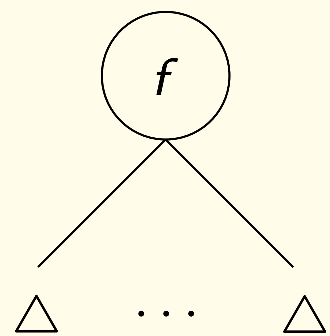
We assume that $\Pi = \emptyset$.

Term algebras

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

- $U_{\mathcal{A}} = \mathrm{T}_{\Sigma}$ (= 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:

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

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:

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

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 $\mathcal{A} \in \text{Mod}(Ax)$, $\mathcal{A} \models t \approx t'$

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.