Decision Procedures for Verification

Part 1. Propositional Logic (4)

21.11.2016

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Last time

Propositional Logic

Syntax

Semantics

Canonical forms

- Computing CNF/DNF by rewriting the formulae
- Structure-Preserving Translation for CNF
- Optimized translation using polarity

Decision Procedures for Satisfiability

- Simple Decision Procedures

  truth table method

- The Resolution Procedure
- The Davis-Putnam-Logemann-Loveland Algorithm

Today

- Applications of propositional logic

- First-order logic.

Applications of propositional logic

- A toy example (sudoku)
- Scheduling
- Verification

Sudoku

1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4			2		
8		5		1					
9				8		6			

Idea: $p^d_{i,j} = \text{true}$ iff the value of square i, j is d

For example: $p^8_{3,5} = \text{true}$

Sudoku

Coding SUDOKU by propositional clauses:

1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4			2		
8		5		1					
9				8		6			

- Concrete values result in units: $p^d_{i,j}$.
- For every value, column we generate: $\neg p^d_{i,j} \vee \neg p^d_{i,k}$ (if $j \neq k$).
  Accordingly for all rows and 3×3 boxes.
- For every square we generate: $p^1_{i,j} \vee \ldots p^9_{i,j}$.
  For every two different values d, d', and every square we generate: $\neg p^d_{i,j} \vee \neg p^{d'}_{i,j}$.
- For every value d and every column we generate:
  $p^d_{i,1} \vee \ldots p^d_{i,9}$.
  Accordingly for all rows and 3×3 boxes.

Sudoku

1								1	
2	4								
3		2							
4					5		4		7
5			8				3		
6			1		9				
7	3			4			2		
8		5		1					
9				8		6			

Set of clauses satisfiable $\Leftrightarrow$ Sudoku has a solution

Let $\mathcal{A}$ be a satisfying assignment

$\mathcal{A}(p_{i,j}^k) = 1$ iff a k appears in line i, column j.

Scheduling

Example: A simple scheduling problem

In a school there are three teachers with the following specialization combinations:

Müller	Mathematics
Schmidt	German
Körner	Mathematics, German

	Group a	Group b
8:00– 8:50	Mathematics	German
9:00– 9:50	German	German
10:00–10:50	Math	Mathematics

Each teacher must teach at least two classes.

Scheduling

Müller	Mathematics
Schmidt	German
Körner	Mathematics, German

	Group a	Group b
1) 8:00– 8:50	Mathematics	German
2) 9:00– 9:50	German	German
3)10:00–10:50	Math	Mathematics

Modeling:

Propositional variables: $P_{s,k,N,f}$ 'Teacher N teaches subject f in group k in time slot s'

Scheduling

Müller	Mathematics
Schmidt	German
Körner	Mathematics, German

	Group a	Group b
1) 8:00– 8:50	Mathematics	German
2) 9:00– 9:50	German	German
3)10:00–10:50	Math	Mathematics

Modeling:

Propositional variables: $P_{s,k,N,f}$ 'Teacher N teaches subject f in group k in time slot s'

Rules: $(P_{1,a,M,m} \vee P_{1,a,K,m}) \wedge (P_{1,b,S,d} \vee P_{1,b,K,d})$

$(P_{2,a,S,d} \vee P_{2,a,K,d}) \wedge (P_{2,b,S,d} \vee P_{2,b,K,d})$

$(P_{3,a,M,m} \vee P_{3,a,K,m}) \wedge (P_{3,b,S,d} \vee P_{3,a,K,d})$

$\neg(P_{1,a,K,m} \wedge P_{1,b,K,d}) \wedge \neg(P_{2,a,K,d} \wedge P_{2,b,K,d}) \wedge \neg(P_{2,a,S,d} \wedge P_{2,b,S,d}) \wedge$

$\neg(P_{3,a,K,m} \wedge P_{3,b,K,m}) \wedge (P_{1,a,M,m} \wedge P_{1,b,M,m}) \ldots$

Program Verification

- Bounded model checking
- Model checking
- Invariant checking/generation
- Abstraction

Finite-state systems

- X finite set of variables, V finite set of possible values for the variables p^i_{xv} (in the i-th step x takes value v)
- Other propositional variables q_k, $k \in K$
- Transitions (variables change their value)

  $$Tr(i, i+1) := \bigvee \left(\text{Cond}(p^i_{x_1 v^i_1}, \dots, p^i_{x_n v^i_n}) \wedge \bigwedge_{j=1}^n p^{i+1}_{x_j v^{i+1}_j} \wedge \bigwedge_k q^{i+1}_k \right)$$

  (where v^{i+1}_j, q^{i+1}_k suitably computed)

$F(p^k_{x_1, v^k_1}, \dots, p^k_{x_n, v^k_n}, ...)$ property of assignments

Bounded model checking:

$$\bigwedge_{j=1}^{n} p^1_{x_j, v_j} \wedge \bigwedge q^1_k \wedge Tr(1, 2) \wedge \dots \wedge Tr(k-1, k) \wedge \neg F(p^k_{x_1, v^k_1}, \dots, p^k_{x_n, v^k_n}, ...)$$

Example

Question: Does BubbleSort return a sorted array?

int [] BubbleSort(int[] a) {
  int i, j, t;
  for ($i := |a| - 1; i > 0; i := i - 1$) {
    for ($j := 0; j < i; j := j + 1$) {
      if ($a[j] > a[j + 1]$){$t := a[j]$;
        $a[j] := a[j + 1]$;
        $a[j + 1] := t$};
}} return a}

Example

Question: Does BubbleSort return a sorted array?

```
int [] BubbleSort(int[] a) {
    int i, j, t;
    for (i := |a| − 1; i > 0; i := i − 1) {
        for (j := 0; j < i; j := j + 1) {
            if (a[j] > a[j + 1]){t := a[j];
                                 a[j] := a[j + 1];
                                 a[j + 1] := t};
}} return a}
```

Simpler question:

$|a| = 3$; $a[0]=7$, $a[1]=9$, $a[2]=4$

does BubbleSort applied to this array return a sorted array?

Encoding in propositional logic:

- p_{ij}^{k} (at step k, $a[i] = k$)

  Examples: $p_{07}^{1}, p_{19}^{1}, p_{24}^{1}$

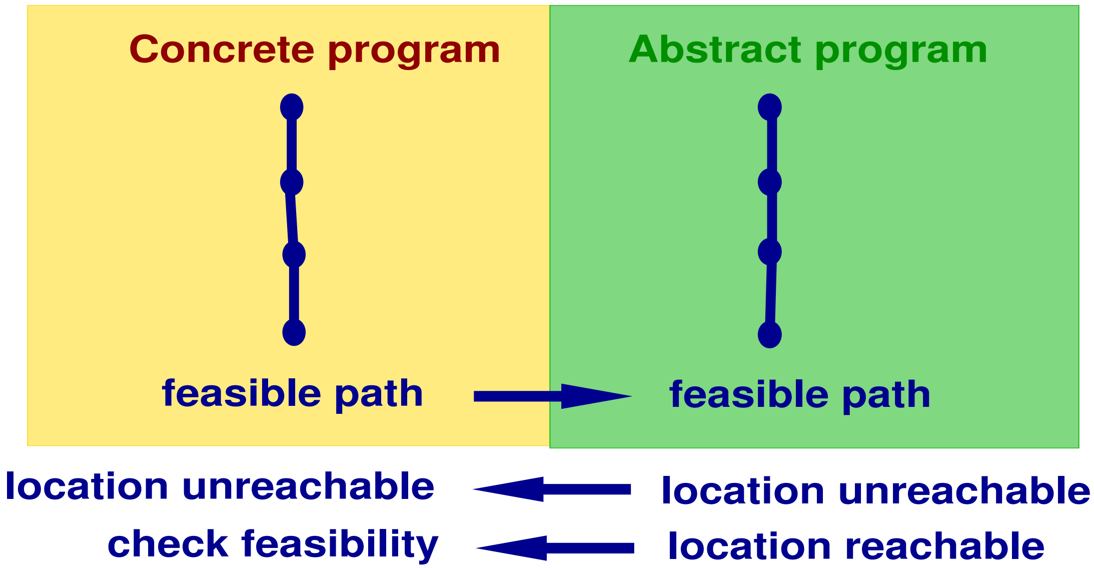
- gt_{ij}^{k} (at step k, $a[i] > a[j]$)

  Examples: $gt_{10}^{1}, \neg gt_{01}^{1}, gt_{02}^{1}, \neg gt_{20}^{1}, \dots$

Model updates with new propositional variables

(complicated; not very expressive)

Abstraction-Based Verification



$\Downarrow$

conjunction of constraints: $\phi(1) \wedge Tr(1,2) \wedge \cdots \wedge Tr(n-1,n) \wedge \neg \text{safe}(n)$

- satisfiable: feasible path
- unsatisfiable: refine abstract program s.t. the path is not feasible

Tools for SAT checking

http://www.satcompetition.org/

Examples of SAT solvers:

MiniSat: http://minisat.se/

MathSAT: http://mathsat.fbk.eu/publications.html (much more)

zChaff: http://www.princeton.edu/ chaff/zchaff.html

Example of use

Tools for SAT checking

Resolution-based theorem provers:

E: `http://www4.informatik.tu-muenchen.de/ schulz/E/E.html`

SPASS: `http://www.spass-prover.org/`

Vampire: `http://www.vprover.org/`

. . . full power for first-order logic (with equality)

Part 2: First-Order Logic

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- is not too expressive
  (e. g. not axiomatizable: natural numbers, uncountable sets)
- has a rich structure of decidable fragments
- has a rich model and proof theory

First-order logic is also called (first-order) predicate logic.

2.1 Syntax

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.
We use letters P, Q, R, S, to denote propositional variables.

Signature

Refined concept for practical applications:
many-sorted signatures (corresponds to simple type systems in programming languages).

Most results established for one-sorted signatures extend in a natural way to many-sorted signatures.

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 $a(f) = s_1 \ldots s_n \to s$,
- Π is a set of predicate symbols p with arity $a(p) = s_1 \ldots s_m$

where $s_1, \ldots, s_n, s_m, s$ 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.

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:

$$
\begin{array}{llll}
t, u, v & ::= & x & , x \in X & \text{(variable)} \\
 & | & f(s_1, ..., s_n) & , f/n \in \Omega & \text{(functional term)}
\end{array}
$$

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.

Terms

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

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

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

Terms

In other words, terms are formal expressions with well-balanced brackets which we may also view as marked, ordered trees.

The markings are function symbols or variables.

The nodes correspond to the subterms of the term.

A node v that is marked with a function symbol f of arity n has exactly n subtrees representing the n immediate subterms of v.

Atoms

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

$$
\begin{array}{llll}
A, B & ::= & p(t_1, \ldots, t_m) & , \; p/m \in \Pi \\
\Big[& \mid & (t \approx t') & \text{(equation)} \quad \Big]
\end{array}
$$

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.

Atoms

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

$$
\begin{array}{llll}
A, B & ::= & p(t_1, ..., t_m) & , \; p/m \in \Pi \\
\Big[& | & (t \approx t') & \text{(equation)} \quad \Big]
\end{array}
$$

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

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

Clauses

$$
\begin{array}{lcll}
C, D & ::= & \bot & \text{(empty clause)} \\
 & \mid & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)}
\end{array}
$$

General First-Order Formulas

$\mathsf{F}_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

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

Notational Conventions

We omit brackets according to the following rules:

- $\neg \quad >_p \quad \wedge \quad >_p \quad \vee \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
  (binding precedences)
- $\vee$ and $\wedge$ are associative and commutative
- $\rightarrow$ is right-associative

$Qx_1, \dots, x_n\, F$ abbreviates $Qx_1 \dots Qx_n\, F$.

Notational Conventions

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

$$
\begin{array}{rcc}
s + t * u & \text{for} & +(s, *(t, u)) \\
s * u \leq t + v & \text{for} & \leq (*(s, u), +(t, v)) \\
-s & \text{for} & -(s) \\
0 & \text{for} & 0()
\end{array}
$$

Conventions

In what follows we will use the following conventions:

constants (0-ary function symbols) are denoted with $a, b, c, d, \ldots$

function symbols with arity ≥ 1 are denoted

- $f, g, h, \ldots$ if the formulae are interpreted into arbitrary algebras
- $+, -, s, \ldots$ if the intended interpretation is into numerical domains

predicate symbols with arity 0 are denoted $P, Q, R, S, \ldots$

predicate symbols with arity ≥ 1 are denoted

- $p, q, r, \ldots$ if the formulae are interpreted into arbitrary algebras
- $\leq, \geq, <, >$ if the intended interpretation is into numerical domains

variables are denoted $x, y, z, \ldots$

Example: Peano Arithmetic

Signature:

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

Examples of formulas over this signature are:

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

Remarks About the Example

We observe that the symbols $\leq$, $<$, 0, s are redundant as they can be defined in first-order logic with equality just with the help of $+$. The first formula defines $\leq$, while the second defines zero. The last formula, respectively, defines s.

Eliminating the existential quantifiers by Skolemization (cf. below) reintroduces the "redundant" symbols.

Consequently there is a *trade-off* between the complexity of the quantification structure and the complexity of the signature.

Example: Specifying LISP lists

Signature:

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

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

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

Examples of formulae:

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

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:

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

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

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

In general, substitutions are mappings

$$\sigma : X \rightarrow T_{\Sigma}(X)$$

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 $codom(\sigma)$.

Many-sorted case: Substitutions must be sort-preserving:
If x is a variable of sort s, then $\sigma(x)$ must be a term of sort s.

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:

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

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 Σ-Alg we denote the class of all Σ-algebras.

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 Σ-Alg we denote the class of all Σ-algebras.

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 $U_s \neq \emptyset$ is a set, called the universe of $\mathcal{A}$ of sort s.

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 $\beta : X \to \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

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 $\beta : X \to \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

Many-sorted case:

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