

# Decision Procedures for Verification

Part 1. Propositional Logic (4)

18.11.2014

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

---

- The Davis-Putnam-Logemann-Loveland Algorithm
- 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:  $p_{i,j}^d = \text{true}$  iff the value of square  $i, j$  is  $d$

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

# 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			

Coding SUDOKU by propositional clauses:

- Concrete values result in units:  $p_{i,j}^d$ .
- For every value, column we generate:  $\neg p_{i,j}^d \vee \neg p_{i,k}^d$  (if  $j \neq k$ ).  
Accordingly for all rows and  $3 \times 3$  boxes.
- For every square we generate:  $p_{i,j}^1 \vee \dots \vee p_{i,j}^9$ .  
For every two different values  $d, d'$ , and every square we generate:  $\neg p_{i,j}^d \vee \neg p_{i,j}^{d'}$ .
- For every value  $d$  and every column we generate:  
 $p_{i,1}^d \vee \dots \vee p_{i,9}^d$ .  
Accordingly for all rows and  $3 \times 3$  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:  $(P_{1,a,M,m} \vee P_{1,a,K,m}) \wedge (P_{1,b,S,d} \vee P_{1,b,K,d})$

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

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

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

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

# 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  $p_{xv}^i$  (in the  $i$ -th step  $x$  takes value  $v$ )
- Other propositional variables  $q_k, k \in K$
- Transitions (variables change their value)

$$Tr(i, i + 1) := \bigvee \left( \text{Cond}(p_{x_1 v_1}^i, \dots, p_{x_n v_n}^i) \wedge \bigwedge_{j=1}^n p_{x_j v_j}^{i+1} \wedge \bigwedge_k q_k^{i+1} \right)$$

(where  $v_j^{i+1}, q_k^{i+1}$  suitably computed)

$F(p_{x_1, v_1}^k, \dots, p_{x_n, v_n}^k, \dots)$  property of assignments

**Bounded model checking:**

$$\bigwedge_{j=1}^n p_{x_j, v_j}^1 \wedge \bigwedge q_k^1 \wedge Tr(1, 2) \wedge \dots \wedge Tr(k-1, k) \wedge \neg F(p_{x_1, v_1}^k, \dots, p_{x_n, v_n}^k, \dots)$$

# 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  $k$ ,  $a[i] > a[j]$ )

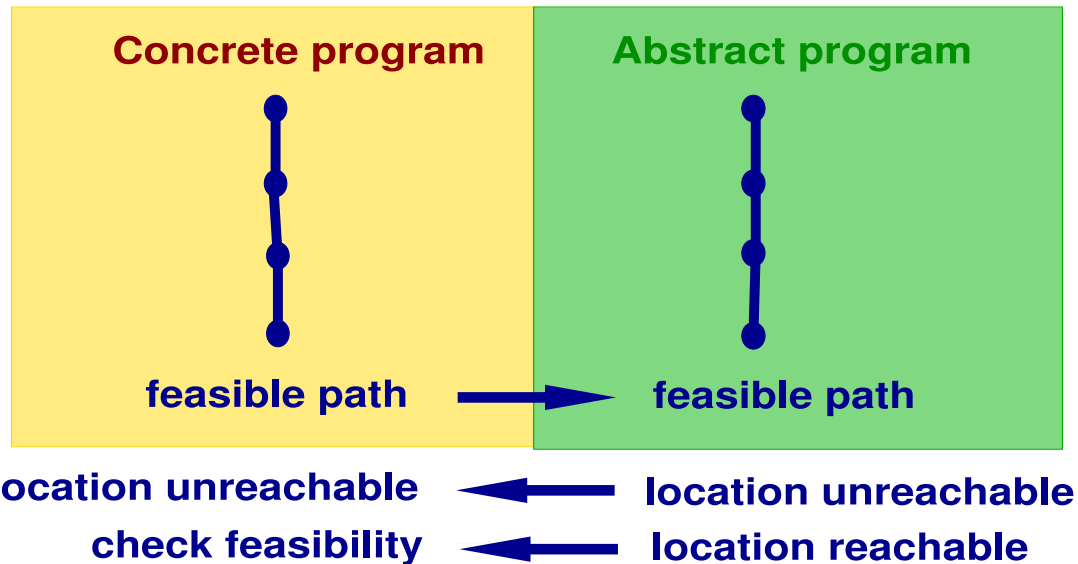
**Examples:**  $gt_{10}^1, \neg gt_{01}^1, gt_{02}^1, \neg gt_{20}^1, \dots$

Model updates with new propositional variables

(complicated; not very expressive)

# Abstraction-Based Verification

---



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

- 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)  
⇒ terms, atomic formulas
- logical symbols (domain-independent)  
⇒ Boolean combinations, quantifiers

# Signature

---

A signature

$$\Sigma = (\Omega, \Pi),$$

fixes an alphabet of non-logical symbols, where

- $\Omega$  is a set of **function symbols**  $f$  with **arity**  $n \geq 0$ , written  $f/n$ ,
- $\Pi$  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,
- $\Omega$  is a set of **function symbols**  $f$  with **arity**  $a(f) = s_1 \dots s_n \rightarrow s$ ,
- $\Pi$  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**.

# 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  $\Sigma$  (resp.,  $\Sigma$ -terms) are formed according to these syntactic rules:

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

By  $T_\Sigma(X)$  we denote the set of  $\Sigma$ -terms (over  $X$ ).

A term not containing any variable is called a **ground term**.

By  $T_\Sigma$  we denote the set of  $\Sigma$ -ground terms.

# Terms

---

**Terms** over  $\Sigma$  (resp.,  $\Sigma$ -terms) are formed according to these syntactic rules:

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

By  $T_\Sigma(X)$  we denote the set of  $\Sigma$ -terms (over  $X$ ).

A term not containing any variable is called a **ground term**.

By  $T_\Sigma$  we denote the set of  $\Sigma$ -ground terms.

## Many-sorted case:

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

if  $a(f) = s_1 \dots s_n \rightarrow s$ , and  $t_i$  are terms of sort  $s_i$ ,  $i = 1, \dots, n$  then  $f(t_1, \dots, t_n)$  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  $\Sigma$  are formed according to this syntax:

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

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  $\Sigma$  are formed according to this syntax:

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

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

# Literals

---

$L ::= A$  (positive literal)  
|  $\neg A$  (negative literal)

# Clauses

---

$C, D ::= \perp$  (empty clause)  
|  $L_1 \vee \dots \vee L_k, k \geq 1$  (non-empty clause)

# General First-Order Formulas

---

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

$F, G, H$	$::=$	$\perp$	(falsum)
		$\top$	(verum)
		$A$	(atomic formula)
		$\neg F$	(negation)
		$(F \wedge G)$	(conjunction)
		$(F \vee G)$	(disjunction)
		$(F \rightarrow G)$	(implication)
		$(F \leftrightarrow G)$	(equivalence)
		$\forall xF$	(universal quantification)
		$\exists xF$	(existential quantification)



# Notational Conventions

---

We omit brackets according to the following rules:

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

$Q_{x_1, \dots, x_n} F$  abbreviates  $Q_{x_1} \dots Q_{x_n} F$ .

# Notational Conventions

---

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

Examples:

$$s + t * u \quad \text{for} \quad +(s, *(t, u))$$

$$s * u \leq t + v \quad \text{for} \quad \leq (*(s, u), +(t, v))$$

$$-s \quad \text{for} \quad -(s)$$

$$0 \quad \text{for} \quad 0()$$

# Conventions

---

In what follows we will use the following conventions:

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

**function symbols** with arity  $\geq 1$  are denoted

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

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

**predicate symbols** with arity  $\geq 1$  are denoted

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

**variables** are denoted  $x, y, z, \dots$

# 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:

$$\forall x, y (x \leq y \leftrightarrow \exists z (x + z \approx y))$$

$$\exists x \forall y (x + y \approx y)$$

$$\forall x, y (x * s(y) \approx x * y + x)$$

$$\forall x, y (s(x) \approx s(y) \rightarrow x \approx y)$$

$$\forall x \exists y (x < y \wedge \neg \exists z (x < z \wedge z < y))$$

# 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

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

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

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

$$\Pi = \emptyset$$

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

### Examples of formulae:

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

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