Logic

(Métodos Formais em Engenharia de Software)

Maria João Frade

Departmento de Informática
Universidade do Minho

2011/2012

Roadmap

- Classical Propositional Logic
- Classical First-Order Logic
- **First-Order Theories**
  - basic concepts; decidability issues; several theories: equality, integers, linear arithmetic, reals, arrays; combining theories
  - satisfiability modulo theories; SMT solvers; SMT-LIB; applications
- Natural Deduction

First-Order Theories

Introduction

- When judging the validity of first-order formulas we are typically interested in a particular domain of discourse, which in addition to a specific underlying vocabulary includes also properties that one expects to hold.
- For instance, in formal methods involving the integers, one is not interested in showing that the formula

$$\forall x, y.\ x < y \rightarrow x < y + y$$

is true for all possible interpretations of the symbols $<$ and $+$, but only for those interpretations in which $<$ is the usual ordering over the integers and $+$ is the addition function.
- We are not interested in validity in general but in validity with respect to some *background theory*, a logical theory that fixes the interpretations of certain predicates and function symbols.

Introduction

- Stated differently, we are often interested in moving away from pure logical validity (i.e. validity in all models) towards a more refined notion of validity restricted to a specific class of models.
- A natural way for specifying such a class of models is by providing a *set of axioms* (sentences that are expected to hold in them). Alternatively, one can pinpoint the models of interest.
- *First-order theories* provide a basis for the kind of reasoning just described.

Theories - basic definitions

Let $\mathcal{V}$ be a vocabulary of a first-order language.

- A first-order *theory* $\mathcal{T}$ is a set of $\mathcal{V}$-sentences that is closed under derivability (i.e., $\mathcal{T} \models \phi$ implies $\phi \in \mathcal{T}$).
- A *$\mathcal{T}$-structure* is a $\mathcal{V}$-structure that validates every formula of $\mathcal{T}$.
- A formula ϕ is *$\mathcal{T}$-valid* (resp. *$\mathcal{T}$-satisfiable*) if every (resp. some) $\mathcal{T}$-structure validates ϕ.
- Two formulae ϕ and ψ are *$\mathcal{T}$-equivalent* if $\mathcal{T} \models \phi \leftrightarrow \psi$ (i.e, for every $\mathcal{T}$-structure $\mathcal{M}$, $\mathcal{M} \models \phi$ iff $\mathcal{M} \models \psi$).

Theories - basic definitions

- $\mathcal{T}$ is said to be a *consistent* theory if at least one $\mathcal{T}$-structure exists.
- $\mathcal{T}$ is said to be a *complete* theory if, for every $\mathcal{V}$-sentence ϕ, either $\mathcal{T} \models \phi$ or $\mathcal{T} \models \neg\phi$.
- $\mathcal{T}$ is said to be a *decidable* theory if there exists a decision procedure for checking $\mathcal{T}$-validity.

Theories - basic definitions

- Let K be a class of $\mathcal{V}$-structures. The *theory of K*, denoted by $\mathsf{Th}(K)$, is the set of sentences valid in all members of K, i.e., $\mathsf{Th}(K) = \{\phi \mid \mathcal{M} \models \phi, \text{for all } \mathcal{M} \in K\}$.
- Given a set of $\mathcal{V}$-sentences Γ, the class of *models for Γ*, denoted by $\mathsf{Mod}(\Gamma)$, is defined as $\mathsf{Mod}(\Gamma) = \{\mathcal{M} \mid \text{for all } \phi \in \Gamma, \mathcal{M} \models \phi\}$.
- A subset $\mathcal{A} \subseteq \mathcal{T}$ is called an *axiom set* for the theory $\mathcal{T}$, when $\mathcal{T}$ is the deductive closure of $\mathcal{A}$, i.e. $\phi \in \mathcal{T}$ iff $\mathcal{A} \models \phi$. A theory $\mathcal{T}$ is *finitely* (resp. *recursively*) *axiomatisable* if it possesses a finite (resp. recursive) set of axioms.
- A *fragment* of a theory is a syntactically-restricted subset of formulae of the theory.

Theories

- For a given $\mathcal{V}$-structure $\mathcal{M}$, the theory Th($\mathcal{M}$) (of a single-element class of $\mathcal{V}$-structures) is complete. These semantically defined theories are useful when one is interested in reasoning in some specific mathematical domain such as the natural numbers, rational numbers, etc.
- However, we remark that such theory may lack an axiomatisation, which seriously compromises its use in purely deductive reasoning.
- If a theory is complete and has a recursive set of axioms, it can be shown to be decidable.

Theories

- The decidability criterion for $\mathcal{T}$-validity is crucial for mechanised reasoning in the theory $\mathcal{T}$.
- It may be necessary (or convenient) to restrict the class of formulas under consideration to a suitable *fragment*;
- The $\mathcal{T}$-validity problem in a fragment refers to the decision about whether or not $\phi \in \mathcal{T}$ when ϕ belongs to the fragment under consideration.
- A fragment of interest is the fragment consisting of universal formulas, often referred to as the *quantifier-free (QF) fragment*.

Equality and uninterpreted functions $\mathcal{T}_\mathsf{E}$

- The vocabulary of the theory of *equality* $\mathcal{T}_\mathsf{E}$ consists of
  - equality (=), which is the only interpreted symbol (whose meaning is defined via the axioms of $\mathcal{T}_\mathsf{E}$);
  - constant, function and predicate symbols, which are uninterpreted (except as they relate to =).
- Axioms
  - *reflexivity:* $\forall x.\ x = x$
  - *symmetry:* $\forall x, y.\ x = y \rightarrow y = x$
  - *transitivity:* $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$
  - *congruence for functions:* for every function $f \in \mathcal{T}$ with $\mathsf{ar}(f) = n$,

    $$\forall \overline{x}, \overline{y}.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$
  - *congruence for predicates:* for every predicate $P \in \mathcal{T}$ with $\mathsf{ar}(P) = n$,

    $$\forall \overline{x}, \overline{y}.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow (P(x_1, \ldots, x_n) \leftrightarrow P(y_1, \ldots, y_n))$$
- $\mathcal{T}_\mathsf{E}$-validity is undecidable, but efficiently decidable for the QF fragment.

Natural numbers and integers

The semantic theories of natural numbers and integers are neither axiomatisable nor decidable.

Kurt Gödel first incompleteness theorem (1931)

Any effectively generated (i.e. recursively enumerable) theory capable of expressing elementary arithmetic cannot be both consistent and complete. In particular, for any consistent, effectively generated formal theory that proves certain basic arithmetic truths, there is an arithmetical statement that is true, but not provable in the theory.

- A semantic theory Th($\mathcal{M}$), where $\mathcal{M}$ interprets each symbol with its standard mathematical meaning in the interpretation domain, is always a complete theory.
- Therefore, the semantic theories of natural numbers and integers cannot be axiomatisable, not even by an infinite recursive set of axioms.

Peano arithmetic $\mathcal{T}_{\mathsf{PA}}$

- The theory of *Peano arithmetic* $\mathcal{T}_{\mathsf{PA}}$ (1889) is a first-order approximation of the theory of natural numbers.
- Vocabulary: $\mathcal{V}_{\mathsf{PA}} = \{0, 1, +, \times, =\}$
- Axioms:
  - axioms of $\mathcal{T}_{\mathsf{E}}$
  - $\forall x.\ \neg(x + 1 = 0)$ *(zero)*
  - $\forall x, y.\ x + 1 = y + 1 \to x = y$ *(successor)*
  - $\forall x.\ x + 0 = x$ *(plus zero)*
  - $\forall x, y.\ x + (y + 1) = (x + y) + 1$ *(plus successor)*
  - $\forall x.\ x \times 0 = 0$ *(time zero)*
  - $\forall x, y.\ x \times (y + 1) = (x \times y) + x$ *(times successor)*
  - for every formula ϕ with FV$(\phi) = \{x\}$ *(axiom schema of induction)*

$$\phi[0/x] \wedge (\forall x.\ \phi \to \phi[x + 1/x]) \to \forall x.\ \phi$$

- $\mathcal{T}_{\mathsf{PA}}$ is incomplete and undecidable, even for the quantifier-free fragment.

Peano arithmetic $\mathcal{T}_{\mathsf{PA}}$

- The incompleteness result is indeed striking because, at the end of the 19th century, G. Peano had given a set of axioms that were shown to characterise natural numbers up to isomorphism. One of these axioms – the *axiom of induction* – involves quantification over arbitrary properties of natural numbers: "*for every unary predicate* P*, if* $P(0)$ *and* $\forall n.\ P(n) \to P(n+1)$ *then* $\forall n.\ P(n)$", which is not a first-order axiom.

- It is however important to notice that the approximation done by a first-order axiom scheme that replaces the arbitrary property P by a first-order formula ϕ with a free variable x:

$$\phi[0/x] \wedge (\forall x.\ \phi \to \phi[x + 1/x]) \to \forall x.\ \phi$$

restrict reasoning to properties that are definable by first-order formulas, which can only capture a small fragment of all possible properties of natural number. (Recall that the set of first-order formulas is countable while the set of arbitrary properties of natural numbers is $\mathcal{P}(\mathbb{N})$, which is uncountable.)

Presburger arithmetic $\mathcal{T}_{\mathbb{N}}$

- The theory of *Presburger arithmetic* $\mathcal{T}_{\mathbb{N}}$ is the additive fragment of the theory of Peano.
- Vocabulary: $\mathcal{V}_{\mathbb{N}} = \{0, 1, +, =\}$
- Axioms:
  - axioms of $\mathcal{T}_{\mathsf{E}}$
  - $\forall x.\ \neg(x + 1 = 0)$ *(zero)*
  - $\forall x, y.\ x + 1 = y + 1 \to x = y$ *(successor)*
  - $\forall x.\ x + 0 = x$ *(plus zero)*
  - $\forall x, y.\ x + (y + 1) = (x + y) + 1$ *(plus successor)*
  - for every formula ϕ with FV$(\phi) = \{x\}$ *(axiom schema of induction)*

$$\phi[0/x] \wedge (\forall x.\ \phi \to \phi[x + 1/x]) \to \forall x.\ \phi$$

- $\mathcal{T}_{\mathbb{N}}$ is both complete and decidable (Presburger, 1929), but it has double exponential complexity.

Linear integer arithmetic $\mathcal{T}_{\mathbb{Z}}$

- Vocabulary: $\mathcal{V}_{\mathbb{Z}} = \{\ldots, -2, -1, 0, 1, 2, \ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots, +, -, >, =\}$
- Each symbol is interpreted with its standard mathematical meaning in $\mathbb{Z}$.
  - Note: $\ldots, -3\cdot, -2\cdot, 2\cdot, 3\cdot, \ldots$ are unary functions. For example, the intended meaning of $3 \cdot x$ is $x + x + x$, and of $-2 \cdot x$ is $-x - x$.

$\mathcal{T}_{\mathbb{Z}}$ and $\mathcal{T}_{\mathbb{N}}$ have the same expressiveness

- For every formula of $\mathcal{T}_{\mathbb{Z}}$ there is an equisatisfiable formula of $\mathcal{T}_{\mathbb{N}}$.
- For every formula of $\mathcal{T}_{\mathbb{N}}$ there is an equisatisfiable formula of $\mathcal{T}_{\mathbb{Z}}$.

Let ϕ be a formula of $\mathcal{T}_{\mathbb{Z}}$ and ψ a formula of $\mathcal{T}_{\mathbb{N}}$. ϕ and ψ are *equisatisfiable* if

ϕ is $\mathcal{T}_{\mathbb{Z}}$-satisfiable iff ψ is $\mathcal{T}_{\mathbb{N}}$-satisfiable

- $\mathcal{T}_{\mathbb{Z}}$ is both complete and decidable via the rewriting of $\mathcal{T}_{\mathbb{Z}}$-formulae into $\mathcal{T}_{\mathbb{N}}$-formulae.

$\mathcal{T}_{\mathbb{Z}}$ versus $\mathcal{T}_{\mathbb{N}}$

Consider the $\mathcal{T}_{\mathbb{Z}}$-formula $\forall x, y. \exists z.\ y + 3x - 3 > -2z$

- For each variable v ranging over the integers, introduce two variables, v_p and v_n ranging over the nonnegative integers.

$$\forall x_p, x_n, y_p, y_n. \exists z_p, z_n.\ (y_p - y_n) + 3(x_p - x_n) - 4 > -2(z_p - z_n)$$

- Eliminate negation.

$$\forall x_p, x_n, y_p, y_n. \exists z_p, z_n.\ y_p + 3x_p + 2z_p > 2z_n + y_n + 3x_n + 4$$

- Eliminate $>$ and numbers.

$$\forall x_p, x_n, y_p, y_n. \exists z_p, z_n. \exists u.\ \neg(u = 0) \ \wedge\ y_p + x_p + x_p + x_p + z_n + z_p = z_n + z_n + y_n + x_n + x_n + x_n + 1 + 1 + 1 + 1 + u$$

This is a $\mathcal{T}_{\mathbb{N}}$-formula equisatisfiable to the original one.

$\mathcal{T}_{\mathbb{N}}$ versus $\mathcal{T}_{\mathbb{Z}}$

The $\mathcal{T}_{\mathbb{N}}$-formula

$$\forall x. \exists y.\ x = y + 1$$

is equisatisfiable to the $\mathcal{T}_{\mathbb{Z}}$-formula

$$\forall x.\ x > -1 \rightarrow \exists y.\ y > -1 \wedge x = y + 1$$

To decide $\mathcal{T}_{\mathbb{Z}}$-validity for a $\mathcal{T}_{\mathbb{Z}}$-formula ϕ

- transform $\neg\phi$ to an equisatisfiable $\mathcal{T}_{\mathbb{N}}$-formula $\neg\psi$
- decide $\mathcal{T}_{\mathbb{N}}$-validity of ψ

Linear rational arithmetic $\mathcal{T}_{\mathbb{Q}}$

- The full theory of rational numbers (with addition and multiplication) is *undecidable*, since the property of being a natural number can be encoded in it.
- But the theory of *linear arithmetic over rational numbers $\mathcal{T}_{\mathbb{Q}}$* is decidable, and actually more efficiently than the corresponding theory of integers.
- Vocabulary: $\mathcal{V}_{\mathbb{Q}} = \{0, 1, +, -, =, \geq\}$
- Axioms: 10 (see Manna's book)
- Rational coefficients can be expressed in $\mathcal{T}_{\mathbb{Q}}$.

The formula $\frac{5}{2}x + \frac{4}{3}y \leq 6$ can be written as the $\mathcal{T}_{\mathbb{Q}}$-formula

$$36 \geq 15x + 8y$$

- $\mathcal{T}_{\mathbb{Q}}$ is decidable and its quantifier-free fragment is efficiently decidable.

Reals $\mathcal{T}_{\mathbb{R}}$

- Surprisingly, the *theory of reals $\mathcal{T}_{\mathbb{R}}$* is decidable even in the presence of multiplication and quantifiers.
- Vocabulary: $\mathcal{V}_{\mathbb{R}} = \{0, 1, +, \times, -, =, \geq\}$
- Axioms: 17 (see Manna's book)

The inclusion of multiplication allows a formula like $\exists x.\ x^2 = 3$ to be expressed (x^2 abbreviates $x \times x$). This formula should be $\mathcal{T}_{\mathbb{R}}$-valid, since the assignment $x \mapsto \sqrt{3}$ satisfies $x^2 = 3$.

- $\mathcal{T}_{\mathbb{R}}$ is decidable (Tarski, 1949). However, it has a high time complexity (doubly exponential).

Difference arithmetic

- *Difference logic* is a fragment (a sub-theory) of linear arithmetic.
- Atomic formulas have the form $x - y \leq c$, for variables x and y and constant c.
- Conjunctions of difference arithmetic inequalities can be checked very efficiently for satisfiability by searching for negative cycles in weighted directed graphs.

  Graph representation: each variable corresponds to a node, and an inequality of the form $x - y \leq c$ corresponds to an edge from y to x with weight c.
- The quantifier-free satisfiability problem is solvable in $\mathcal{O}(|V||E|)$.

Arrays $\mathcal{T}_\mathsf{A}$ and $\mathcal{T}_\mathsf{A}^=$

- Arrays are modeled in logic as applicative data structures.
- Vocabulary: $\mathcal{V}_\mathsf{A} = \{read, write, =\}$
- Axioms:
  - (reflexivity), (symmetry) and (transitivity) of $\mathcal{T}_\mathsf{E}$
  - $\forall a, i, j.\ i = j \rightarrow read(a, i) = read(a, j)$
  - $\forall a, i, j, v.\ i = j \rightarrow read(write(a, i, v), j) = v$
  - $\forall a, i, j, v.\ \neg(i = j) \rightarrow read(write(a, i, v), j) = read(a, j)$
- $=$ is only defined for array elements.
- $\mathcal{T}_\mathsf{A}^=$ is the theory $\mathcal{T}_\mathsf{A}$ plus an axiom (extensionality) to capture $=$ on arrays.
  - $\forall a, b.\ (\forall i.\ read(a, i) = read(b, i)) \leftrightarrow a = b$
- Both $\mathcal{T}_\mathsf{A}$ and $\mathcal{T}_\mathsf{A}^=$ are undecidable. But their quantifier-free fragments are decidable.
- Alternative fragments are often preferred that subsume the quantifier-free fragment (allowing restricted forms of index quantification).

Other theories

- *Fixed-size bit-vectors*

  Model bit-level operations of machine words, including 2^n-modular operations (where n is the word size), shift operations, etc.
  Decision procedures for the theory of fixed-size bit vectors often rely on appropriate encodings in propositional logic.
- *Algebraic data structures*

  The theories describe data structures that are ubiquitous in programming like lists, stacks, binary trees, etc.

  These theories are built around the theory of equality with uninterpreted functions, and are normally efficiently decidable for the quantifier-free fragment.
- ...

Combining theories

- In practice, the most of the formulae we want to check need a combination of theories.

Checking $x + 2 = y \rightarrow f(read(write(a, x, 3), y - 2)) = f(y - x + 1)$
involves 3 theories: equality and uninterpreted functions, arrays and arithmetic.

- Given theories $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\mathcal{V}_1 \cap \mathcal{V}_2 = \{=\}$, the *combined theory* $\mathcal{T}_1 \cup \mathcal{T}_2$ has vocabulary $\mathcal{V}_1 \cup \mathcal{V}_2$ and axioms $A_1 \cup A_2$
- Nelson and Oppen showed that if
  - satisfiability of the quantifier-free fragment of $\mathcal{T}_1$ is decidable,
  - satisfiability of the quantifier-free fragment of $\mathcal{T}_2$ is decidable, and
  - certain technical requirements are met,

  then the satisfiability in the quantifier-free fragment of $\mathcal{T}_1 \cup \mathcal{T}_2$ is decidable.
- Most methods available are based on the Nelson-Oppen combination method.

Satisfiability Modulo Theories

- The *Satisfiability Modulo Theories (SMT) problem* is a variation of the SAT problem for first-order logic, with the interpretation of symbols constrained by (a combination of) specific theories (i.e., it is the problem of determining, for a theory $\mathcal{T}$ and given a formula ϕ, whether ϕ is $\mathcal{T}$-satisfiable).
- Usually SMT solvers address the issue of satisfiability of quantifier-free first-order CNF formulas, using as building blocks:
  - a propositional SAT solver, and
  - state-of-the-art theory solvers.
- More precisely, generic Boolean reasoning is separated from theory reasoning, reducing the theory solver to its essence. The common practice is to write theory solvers just for conjunctions of literals.
- A standard technique for integrating SAT solvers and theory solvers is the *"lazy offline"* approach.

SMT solvers - "lazy offline" approach

Given a formula ψ with atoms $\{a_1, \ldots, a_n\}$ and a set of propositional variables $\{P_1, \ldots, P_n\}$ not occurring in ψ,

- The *abstraction mapping* prop from formulas over $\{a_1, \ldots, a_n\}$ to propositional formulas over $\{P_1, \ldots, P_n\}$ is defined as the homomorphism induced by $\mathsf{prop}(a_i) = P_i$.
- The inverse prop^{-1} of such an abstraction mapping prop simply replaces propositional variables P_i with their associated atom a_i.

For an assignment α of $\mathsf{prop}(\psi)$, let the set $\Phi(\alpha)$ of first-order literals be defined as follows

$$\Phi(\alpha) = \{\mathsf{prop}^{-1}(P_i) \mid \alpha(P_i) = 1\} \cup \{\neg\mathsf{prop}^{-1}(P_i) \mid \alpha(P_i) = 0\}$$

SMT solvers - "lazy offline" approach

- Given a CNF A, SAT-Solver(A) returns a tuple (r, α) where r is SAT if A is satisfiable and UNSAT otherwise, and α is an assignment that satisfies A if r is SAT.
- Given a set of literals S, T-Solver(S) returns a tuple (r, J) where r is SAT if S is $\mathcal{T}$-satisfiable and UNSAT otherwise, and J is a justification if r is UNSAT.
- Given an unsatisfiable set of literals S, a *justification* for S is any unsatisfiable subset J of S. A justification J is *non-redundant* if there is no strict subset J' of J that is also unsatisfiable.

SMT solvers - "lazy offline" approach

Basic SAT and theory solver integration

```
SMT-Solver ($\psi$) {
  $A \leftarrow$ prop($\psi$)
  **loop** {
    $(r, \alpha) \leftarrow$ SAT-Solver($A$)
    **if** $r$ = UNSAT **then return** UNSAT
    $(r, J) \leftarrow$ T-Solver($\Phi(\alpha)$)
    **if** $r$ = SAT **then return** SAT
    $C \leftarrow \bigvee_{B \in J} \neg$prop($B$)
    $A \leftarrow A \wedge C$
  }
}
```

If a valuation α satisfying A is found, but $\Phi(\alpha)$ is unsatisfiable, we add to A a clause C which has the effect of excluding α when the SAT solver is invoked again in the next iteration. This clause is called a *"theory lemma"*.

SMT-Solver($g(a) = x \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$)

- $A =$ prop$(\psi) = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$
- SAT-Solver(A) = SAT, $\alpha = \{P_1 \mapsto 1, P_2 \mapsto 0, P_4 \mapsto 0\}$
- $\Phi(\alpha) = \{g(a) = x, f(g(a)) \neq f(c), c \neq d\}$
  T-Solver$(\Phi(\alpha))$ = UNSAT, $J = \{g(a) = x, f(g(a)) \neq f(c), c \neq d\}$
- $C = \neg P_1 \vee P_2 \vee P_4$

- $A = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4 \wedge (\neg P_1 \vee P_2 \vee P_4)$
  SAT-Solver(A) = SAT, $\alpha = \{P_1 \mapsto 1, P_2 \mapsto 1, P_3 \mapsto 1, P_4 \mapsto 0\}$
- $\Phi(\alpha) = \{g(a) = x, f(g(a)) = f(c), g(a) = d, c \neq d\}$
  T-Solver$(\Phi(\alpha))$ = UNSAT, $J = \{g(a) = x, f(g(a)) = f(c), g(a) = d, c \neq d\}$
- $C = \neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4$

- $A = P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4 \wedge (\neg P_1 \vee P_2 \vee P_4) \wedge (\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4)$
  SAT-Solver(A) = UNSAT

SMT solvers

- The main advantage of the lazy approach is its flexibility, since it can easily combine any SAT solver with any theory solver.

- There are many refinements for this basic algorithm that make the SMT procedure more efficient. The basic idea is to have a tighter integration between the two procedures.

SMT solvers

- In the last two decades, SMT procedures have undergone dramatic progress. There has been enormous improvements in efficiency and expressiveness of SMT procedures for the more commonly occurring theories.
  - The annual competition[1] for SMT procedures plays an important rule in driving progress in this area.
  - A key ingredient is SMT-LIB[2], an online resource that proposes, as a standard, a unified notation and a collection of benchmarks for performance evaluation and comparison of tools.

- Current SMT solvers: Z3, Yices, MathSAT, Barcelogic, CVC3, openSMT, Alt-Ergo, etc.
- Usually, SMT solvers accept input either in a proprietary format or in SMT-LIB format.

[1] http://www.smtcomp.org
[2] http://www.smtlib.org

The SMT-LIB repository

- Catalog of theory declarations - semi-formal specification of theories of interest
  - A theory defines a vocabulary of sorts and functions. The meaning of the theory symbols are specified in the theory declaration.
- Catalog of logic declarations - semi-formal specification of fragments of (combinations of) theories
  - A logic consists of one or more theories, together with some restrictions on the kinds of expressions that may be used within that logic.
- Library of benchmarks
- Utility tools (parsers, converters, ...)
- Useful links (documentation, solvers, ...)
- See http://www.smtlib.org

The SMT-LIB language

- Textual, command-based I/O format for SMT solvers.
- Intended mostly for machine processing. (SMT solvers are typically used for verification as backends)
- All input to and output from a conforming solver is a sequence of one or more *S-expressions*

$$\langle \text{S-exp} \rangle \quad ::= \quad \langle \text{token} \rangle \mid (\langle \text{S-exp} \rangle^*)$$

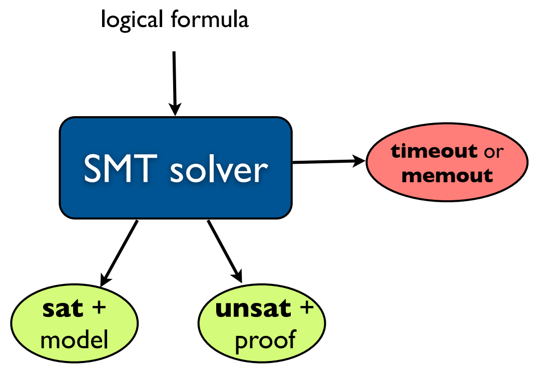
- SMT-LIB language expresses logical statements in a many-sorted first-order logic. Each well-formed expression has a unique *sort* (type).
- Typical usage:
  - Asserting a series of logical statements, in the context of a given logic.
  - Checking their satisfiability in the logic.
  - Exploring resulting models (if SAT) or proofs (is UNSAT)

The SMT-LIB language

- Theories are defined with theory declaration schemes
  - Signature (sorts and function symbols) formally specified.
  - Semantics informally specified.
- Current theories
  - ArraysEx - Functional arrays with extensionality.
  - Fixed_Size_BitVectors - Bit vectors with arbitrary size.
  - Core - Core theory, defining the basic Boolean operators.
  - Ints - Integer numbers.
  - Reals - Real numbers.
  - Reals_Ints - Real and integer numbers.
- Some logics (theories; free symbols; sintax restrictions)
  - QF_UF - Core; free sort and function symbols; no quantifiers
  - QF_LIA - Ints; free constant symbols: no quantifiers, only linear
  - AUFLIA - ArraysEx, Ints; free sort and function symbols; only linear terms, only arrays of sort (Array Int Int)

Theorem provers / SAT checkers

ϕ is valid iff $\neg\phi$ is unsatisfiable



It may happen that, for a given formula, a SMT solver returns a timeout, while another SMT solver returns a concrete answer.

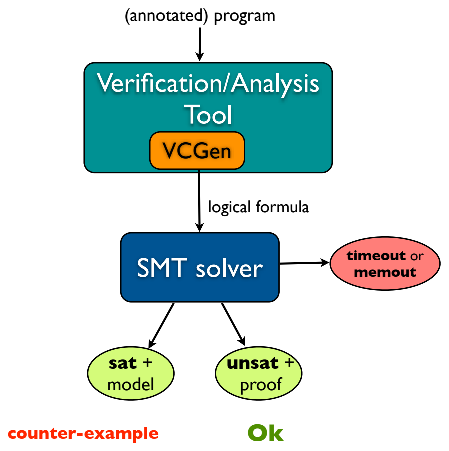
Applications

SMT solvers are the core engine of many tools for

- program analysis
- program verification
- test-cases generation
- bounded model checking
- modeling
- planning and scheduling
- ...

Program verification/analysis

The general architecture of program verification/analysis tools is powered by a *Verification Conditions Generator (VCGen)* that produces verification conditions (also called "proof obligations") that are then passed to a SMT solver to be "discharged". Examples of such tools: Boogie, Frama-C, ESC/JAVA2.



Bounded model checking

- The key idea of Bounded Model Checking (BMC) is to encode bounded behaviours of the system that enjoy some given property as a formula whose models (if any) describe a system trace leading to a violation of the property.
- Preliminarily to the generation of the formula, we preprocess the input program.
- Given a bound $n > 0$, this amounts to applying a number of transformations which lead to a simplified program whose execution traces have finite length and correspond to the (possibly truncated) traces of the original program.
- The quantifier-free formula is then obtained by generating a quantifier-free formula for each statement of the resulting program and the resulting formula is fed to a SMT solver.
- If an execution path leading to a violation of an assert statement occurring in the original program is detected, then a corresponding trace is built and returned to the user for inspection.

Bounded model checking

original program

```
i = a[0];
if (x > 0){
  if (x < 10)
    x = x + 1;
  else
    x = x - 1;
}
assert(y > 0 && y < 5);
a[y] = i;
```

$\Longrightarrow$

single assignment form

```
i₁ = a₀[0];
if (x₀ > 0){
  if (x₀ < 10)
    x₁ = x₀ + 1;
  else
    x₁ = x₀ - 1;
}
assert(y₀ > 0 && y₀ < 5);
a₁[y₀] = i₁;
```

$\Longrightarrow$

conditional normal form

```
if (true) i₁ = a₀[0];
if (x₀ > 0 && x₀ < 10) x₁ = x₀ + 1;
if (x₀ > 0 && !x₀ < 10) x₁ = x₀ - 1;
if (true) assert(y₀ > 0 && y₀ < 5);
if (true) a₁[y₀] = i₁;
```

Bounded model checking

Now, for some given background theory $\mathcal{T}$ two set of quantifier-free formulae $\mathcal{C}$ and $\mathcal{P}$ such that $\mathcal{C} \models_{\mathcal{T}} \bigwedge \mathcal{P}$ iff no computation path of the program violates any assert statement in it.

$$\begin{array}{ll} \mathcal{C} = \{ & \top \to i_1 = read(a_0, 0), \\ & x_0 > 0 \wedge x_0 < 10 \to x_1 = x_0 + 1, \\ & x_0 > 0 \wedge \neg(x_0 < 10) \to x_1 = x_0 - 1, \\ & \top \to write(a_1, y_0, i_1) \\ \} & \\ \mathcal{P} = \{ & \top \to y_0 > 0 \wedge y_0 < 5 \ \ \} \end{array}$$

Note that $\mathcal{C} \models_{\mathcal{T}} \bigwedge \mathcal{P}$ iff $\models_{\mathcal{T}} \bigwedge \mathcal{C} \to \bigwedge \mathcal{P}$
iff $\neg(\bigwedge \mathcal{C} \to \bigwedge \mathcal{P})$ is $\mathcal{T}$-unsatisfiable
iff $\bigwedge \mathcal{C} \wedge \neg \bigwedge \mathcal{P}$ is $\mathcal{T}$-unsatisfiable

The $\mathcal{T}$-models of $(\bigwedge \mathcal{C} \wedge \neg \bigwedge \mathcal{P})$ (if any) correspond to the execution paths of the program that lead to an assertion violation.

Scheduling

Job-shop-scheduling decision problem

- Consider n jobs.
- Each job has m tasks of varying duration that must be performed consecutively on m machines.
- The start of a new task can be delayed as long as needed in order for a machine to become available, but tasks cannot be interrupted once they are started.

Given a total maximum time max and the duration of each task, the problem consists of deciding whether there is a schedule such that the end-time of every task is less than or equal to max time units.

Two types of constraints

- Precedence between two tasks in the same job.
- Resource: a machine cannot run two different tasks at the same time.

Scheduling

- d_{ij} - duration of the j-th task of the job i
- t_{ij} - start-time for the j-th task of the job i
- Constraints
  - Precedence: for every i, j, $t_{i\,j+1} \geq t_{ij} + d_{ij}$
  - Resource: for every $i \neq i'$, $(t_{ij} \geq t_{i'j} + d_{i'j}) \vee (t_{i'j} \geq t_{ij} + d_{ij})$
  - The start time of the first task of every job i must be greater than or equal to zero $t_{i1} \geq 0$
  - The end time of the last task must be less than or equal to max $t_{im} + d_{im} \leq max$

Find a solution for this problem

d_{ij}	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

and $max = 8$

Exercices

- Run by hand the SMT-Solver procedure to decide about the satisfiability of $\neg(a \geq 2) \wedge (a \geq 2 \vee a \geq 7)$.
- Visit the online tutorial guide of the Z3 theorem prover, and experiment.
- Pick up a SMT solver.
- Play with simple examples.