First-Order Logic & Theories

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2018/2019

Roadmap

- **Classical First-Order Logic**
  - syntax; semantics; decision problems SAT and VAL;
  - normal forms; Herbrandization; Skolemization;
  - FOL with equality; many-sorted FOL.
- **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.
- **Practice with a SMT solver**

(Classical) First-Order Logic

Introduction

First-order logic (FOL) is a richer language than propositional logic. Its lexicon contains not only the symbols $\wedge$, $\vee$, $\neg$, and $\rightarrow$ (and parentheses) from propositional logic, but also the symbols $\exists$ and $\forall$ for "there exists" and "for all", along with various symbols to represent variables, constants, functions, and relations.

There are two sorts of things involved in a first-order logic formula:

- *terms*, which denote the objects that we are talking about;
- *formulas*, which denote truth values.

Examples:

"Not all birds can fly."
"Every mother is older than her children."
"John and Peter have the same maternal grandmother."

Syntax

The alphabet of a first-order language is organised into the following categories.

- *Variables:* $x, y, z, \ldots \in \mathcal{X}$ (arbitrary elements of an underlying domain)
- *Constants:* $a, b, c, \ldots \in \mathcal{C}$ (specific elements of an underlying domain)
- *Functions:* $f, g, h, \ldots \in \mathcal{F}$ (every function f has a fixed arity, $\mathsf{ar}(f)$)
- *Predicates:* $P, Q, R, \ldots \in \mathcal{P}$ (every predicate P has a fixed arity, $\mathsf{ar}(P)$)
- *Logical connectives:* $\top$, $\bot$, $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\forall$ (*for all*), $\exists$ (*there exists*)
- *Auxiliary symbols*: "." , "(" and ")".

We assume that all these sets are disjoint. $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{P}$ are the non-logical symbols of the language. These three sets constitute the *vocabulary* $\mathcal{V} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$.

Syntax

Terms

The set of *terms* of a first-order language over a vocabulary $\mathcal{V}$ is given by the following abstract syntax

$$\mathbf{Term}_{\mathcal{V}} \ni t \ ::= \ x \mid c \mid f(t_1, \ldots, t_{\mathsf{ar}(f)})$$

Formulas

The set $\mathbf{Form}_{\mathcal{V}}$, of *formulas* of FOL, is given by the abstract syntax

$$\begin{array}{lcl}\mathbf{Form}_{\mathcal{V}} \ni \phi, \psi & ::= & P(t_1, \ldots, t_{\mathsf{ar}(P)}) \mid \bot \mid \top \mid (\neg\phi) \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \\ & & \mid (\phi \rightarrow \psi) \mid (\forall x.\, \phi) \mid (\exists x.\, \phi)\end{array}$$

An *atomic formula* has the form $\bot$, $\top$, or $P(t_1, \ldots, t_{\mathsf{ar}(P)})$. A *ground term* is a term without variables. *Ground formulas* are formulas without variables, i.e., quantifier-free formulas ϕ such that all terms occurring in ϕ are ground terms.

Syntax

Convention

We adopt some syntactical conventions to lighten the presentation of formulas:

- Outermost parenthesis are usually dropped.
- In absence of parentheses, we adopt the following convention about precedence. Ranging from the highest precedence to the lowest, we have respectively: $\neg$, $\wedge$, $\vee$ and $\rightarrow$. Finally we have that $\rightarrow$ binds more tightly than $\forall$ and $\exists$.
- All binary connectives are right-associative.
- Nested quantifications such as $\forall x. \forall y. \phi$ are abbreviated to $\forall x, y.\, \phi$.
- $\forall \overline{x}. \phi$ denotes the nested quantification $\forall x_1, \ldots, x_n.\, \phi$.

Modeling with FOL

"Not all birds can fly."

We can code this sentence assuming the two unary predicates B and F expressing

$B(x)$ – x is a bird
$F(x)$ – x can fly

The declarative sentence "Not all birds can fly" can now be coded as

$$\neg(\forall x.\, B(x) \rightarrow F(x))$$

or, alternatively, as

$$\exists x. B(x) \wedge \neg F(x)$$

Modeling with FOL

"Every mother is older than her children."
"John and Peter have the same maternal grandmother."

Using constants symbols j and p for John and Peter, and predicates $=$, mother and older expressing that

$\mathit{mother}(x,y)$ – x is mother of y
$\mathit{older}(x,y)$ – x is older than y

these sentences could be expressed by

$$\forall x.\forall y.\, \mathit{mother}(x,y) \to \mathit{older}(x,y)$$

$$\forall x,y,u,v.\, \mathit{mother}(x,y) \land \mathit{mother}(y,j) \land \mathit{mother}(u,v) \land \mathit{mother}(v,p) \to x = u$$

A different and more elegant encoding is to represent y'mother in a more direct way, by using a function instead of a relation. We write $m(y)$ to mean y'mother. This way the two sentences above have simpler encondings.

$$\forall x.\, \mathit{older}(m(x),x) \qquad \text{and} \qquad m(m(j)) = m(m(p))$$

Modeling with FOL

Assume further the following predicates and constant symbols

$\mathit{flower}(x)$ – x is a flower	$\mathit{likes}(x,y)$ – x likes y
$\mathit{sport}(x)$ – x is a sport	$\mathit{brother}(x,y)$ – x is brother of y
a – Anne	

- "Anne likes John's brother." $\exists x.\, \mathit{brother}(x,j) \land \mathit{likes}(a,x)$
- "John likes all sports." $\forall x.\, \mathit{sports}(x) \to \mathit{likes}(j,x)$
- "John's mother likes flowers." $\forall x.\, \mathit{flower}(x) \to \mathit{likes}(m(j),x)$
- "John's mother does not like some sports." $\exists y.\, \mathit{sport}(y) \land \neg \mathit{likes}(m(j),y)$
- "Peter only likes sports." $\forall x.\, \mathit{likes}(p,x) \to \mathit{sports}(x)$
- "Anne has two children."

$$\exists x_1, x_2.\, \mathit{mother}(a,x_1) \land \mathit{mother}(a,x_2) \land x_1 \neq x_2 \land$$
$$\forall z.\, \mathit{mother}(a,z) \to z = x_1 \lor z = x_2$$

Free and bound variables

- The *free variables* of a formula ϕ are those variables occurring in ϕ that are not quantified. FV(ϕ) denotes the set of free variables occurring in ϕ.
- The *bound variables* of a formula ϕ are those variables occurring in ϕ that do have quantifiers. BV(ϕ) denote the set of bound variables occurring in ϕ.

Note that variables can have both free and bound occurrences within the same formula. Let ϕ be $\exists x.\, R(x,y) \land \forall y.\, P(y,x)$, then

$$\mathsf{FV}(\phi) = \{y\} \ \text{ and } \ \mathsf{BV}(\phi) = \{x,y\}.$$

- A formula ϕ is *closed* (or *a sentence*) if it does not contain any free variables.
- If FV(ϕ) $= \{x_1, \ldots, x_n\}$, then
  - its *universal closure* is $\forall x_1. \ldots \forall x_n.\, \phi$
  - its *existential closure* is $\exists x_1. \ldots \exists x_n.\, \phi$

Substitution

Substitution

- We define $u[t/x]$ to be the term obtained by replacing each occurrence of variable x in u with t.
- We define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t.

Care must be taken, because substitutions can give rise to undesired effects.

Given a term t, a variable x and a formula ϕ, we say that *t is free for x in ϕ* if no free x in ϕ occurs in the scope of $\forall z$ or $\exists z$ for any variable z occurring in t.

From now on we will assume that all substitutions satisfy this condition. That is when performing the $\phi[t/x]$ we are always assuming that t is free for x in ϕ.

Substitution

Convention

We write $\phi(x_1,\ldots,x_n)$ to denote a formula having free variables $x_1,\ldots,x_n$. We write $\phi(t_1,\ldots,t_n)$ to denote the formula obtained by replacing each free occurrence of x_i in ϕ with the term t_i. When using this notation, it should always be assumed that each t_i is free for x_i in ϕ. Also note that when writhing $\phi(x_1,...,x_n)$ we do not mean that $x_1,...,x_n$ are the only free variables of ϕ.

A *sentence* of first-order logic is a formula having no free variables.

- The presence of free variables distinguishes formulas from sentences.
- This distinction did not exist in propositional logic.

Semantics

$\mathcal{V}$-structure

Let $\mathcal{V}$ be a vocabulary. A *$\mathcal{V}$-structure* $\mathcal{M}$ is a pair $\mathcal{M} = (D, I)$ where D is a nonempty set called the *interpretation domain*, and I is an *interpretation function* that assigns constants, functions and predicates over D to the symbols of $\mathcal{V}$ as follows:

- for each constant symbol $c \in \mathcal{C}$, the interpretation of c is a constant $I(c) \in D$;
- for each $f \in \mathcal{F}$, the interpretation of f is a function $I(f) : D^{\mathsf{ar}(f)} \to D$;
- for each $P \in \mathcal{P}$, the interpretation of P is a function $I(P) : D^{\mathsf{ar}(P)} \to \{0,1\}$. In particular, 0-ary predicate symbols are interpreted as truth values.

$\mathcal{V}$-structures are also called *models* for $\mathcal{V}$.

Semantics

Assignment

An *assignment* for a domain D is a function $\alpha : \mathcal{X} \to D$.

We denote by $\alpha[x \mapsto a]$ the assignment which maps x to a and any other variable y to $\alpha(y)$.

Given a $\mathcal{V}$-structure $\mathcal{M} = (D, I)$ and given an assignment $\alpha : \mathcal{X} \to D$, we define an *interpretation function for terms*, $\alpha_{\mathcal{M}} : \mathbf{Term}_{\mathcal{V}} \to D$, as follows:

$$\begin{array}{lcl}
\alpha_{\mathcal{M}}(x) & = & \alpha(x) \\
\alpha_{\mathcal{M}}(c) & = & I(c) \\
\alpha_{\mathcal{M}}(f(t_1,\ldots,t_n)) & = & I(f)(\alpha_{\mathcal{M}}(t_1),\ldots,\alpha_{\mathcal{M}}(t_n))
\end{array}$$

Semantics

Satisfaction relation

Given a $\mathcal{V}$-structure $\mathcal{M} = (D, I)$ and given an assignment $\alpha : \mathcal{X} \to D$, we define the *satisfaction relation* $\mathcal{M}, \alpha \models \phi$ for each $\phi \in \mathbf{Form}_{\mathcal{V}}$ as follows:

$$\begin{array}{lll}
\mathcal{M}, \alpha \models \top & & \\
\mathcal{M}, \alpha \not\models \bot & & \\
\mathcal{M}, \alpha \models P(t_1,\ldots,t_n) & \text{iff} & I(P)(\alpha_{\mathcal{M}}(t_1),\ldots,\alpha_{\mathcal{M}}(t_n)) = 1 \\
\mathcal{M}, \alpha \models \neg\phi & \text{iff} & \mathcal{M}, \alpha \not\models \phi \\
\mathcal{M}, \alpha \models \phi \wedge \psi & \text{iff} & \mathcal{M}, \alpha \models \phi \text{ and } \mathcal{M}, \alpha \models \psi \\
\mathcal{M}, \alpha \models \phi \vee \psi & \text{iff} & \mathcal{M}, \alpha \models \phi \text{ or } \mathcal{M}, \alpha \models \psi \\
\mathcal{M}, \alpha \models \phi \to \psi & \text{iff} & \mathcal{M}, \alpha \not\models \phi \text{ or } \mathcal{M}, \alpha \models \psi \\
\mathcal{M}, \alpha \models \forall x.\, \phi & \text{iff} & \mathcal{M}, \alpha[x \mapsto a] \models \phi \text{ for all } a \in D \\
\mathcal{M}, \alpha \models \exists x.\, \phi & \text{iff} & \mathcal{M}, \alpha[x \mapsto a] \models \phi \text{ for some } a \in D
\end{array}$$

Validity and satisfiability

When $\mathcal{M}, \alpha \models \phi$, we say that $\mathcal{M}$ *satisfies* ϕ *with* α.

We write $\mathcal{M} \models \phi$ iff $\mathcal{M}, \alpha \models \phi$ holds for every assignment α.

A formula ϕ is

valid iff $\mathcal{M}, \alpha \models \phi$ holds for all structure $\mathcal{M}$ and assignments α.
A valid formula is called a *tautology*. We write $\models \phi$.

satisfiable iff there is some structure $\mathcal{M}$ and some assigment α such that $\mathcal{M}, \alpha \models \phi$ holds.

unsatisfiable iff it is not satisfiable.
An unsatisfiable formula is called a *contradiction*.

refutable iff it is not valid.

Consequence and equivalence

Given a set of formulas Γ, a model $\mathcal{M}$ and an assignment α, $\mathcal{M}$ is said to *satisfy* Γ *with* α, denoted by $\mathcal{M}, \alpha \models \Gamma$, if $\mathcal{M}, \alpha \models \phi$ for every $\phi \in \Gamma$.

Γ *entails* ϕ (or that ϕ is a *logical consequence* of Γ), denoted by $\Gamma \models \phi$, iff for all structures $\mathcal{M}$ and assignments α, whenever $\mathcal{M}, \alpha \models \Gamma$ holds, then $\mathcal{M}, \alpha \models \phi$ holds as well.

ϕ is *logically equivalent* to ψ, denoted by $\phi \equiv \psi$, iff $\{\phi\} \models \psi$ and $\{\psi\} \models \phi$.

Deduction theorem

$\Gamma, \phi \models \psi$ iff $\Gamma \models \phi \to \psi$

Consistency

The set Γ is *consistent* or *satisfiable* iff there is a model $\mathcal{M}$ and an assigment α such that $\mathcal{M}, \alpha \models \phi$ holds for all $\phi \in \Gamma$.

We say that Γ is *inconsistent* iff it is not consistent and denote this by $\Gamma \models \bot$.

Proposition

- $\{\phi, \neg\phi\} \models \bot$
- If $\Gamma \models \bot$ and $\Gamma \subseteq \Gamma'$, then $\Gamma' \models \bot$.
- $\Gamma \models \phi$ iff $\Gamma, \neg\phi \models \bot$

Substitution

- Formula ψ is a *subformula* of formula ϕ if it occurs syntactically within ϕ.
- Formula ψ is a *strict subformula* of ϕ if ψ is a subformula of ϕ and $\psi \neq \phi$

Substitution theorem

Suppose $\phi \equiv \psi$. Let θ be a formula that contains ϕ as a subformula. Let θ' be the formula obtained by safe replacing (i.e., avoiding the capture of free variables of ϕ) some occurrence of ϕ in θ with ψ. Then $\theta \equiv \theta'$.

Adquate sets of connectives for FOL

Renaming of bound variables

If y is free for x in ϕ and $y \notin \mathsf{FV}(\phi)$, then the following equivalences hold.

- $\forall x.\phi \equiv \forall y.\phi[y/x]$
- $\exists x.\phi \equiv \exists y.\phi[y/x]$

Lemma

The following equivalences hold in first-order logic.

$$\begin{array}{rcl} \forall x.\phi \land \psi \equiv (\forall x.\phi) \land (\forall x.\psi) & \qquad & \exists x.\phi \lor \psi \equiv (\exists x.\phi) \lor (\exists x.\psi) \\ \forall x.\phi \equiv (\forall x.\phi) \land \phi[t/x] & & \exists x.\phi \equiv (\exists x.\phi) \lor \phi[t/x] \\ \neg\forall x.\phi \equiv \exists x.\neg\phi & & \neg\exists x.\phi \equiv \forall x.\neg\phi \end{array}$$

As in propositional logic, there is some redundancy among the connectives and quantifiers since $\forall x.\,\phi \equiv \neg\exists x.\,\neg\phi$ and $\exists x.\,\phi \equiv \neg\forall x.\,\neg\phi$.

Decidability

Given formulas ϕ and ψ as input, we may ask:

Decision problems

Validity problem:	"Is ϕ valid ?"
Satisfiability problem:	"Is ϕ satisfiable ?"
Consequence problem:	"Is ψ a consequence of ϕ ?"
Equivalence problem:	"Are ϕ and ψ equivalent ?"

These are, in some sense, variations of the same problem.

ϕ is valid	iff	$\neg\phi$ is unsatisfiable
$\phi \models \psi$	iff	$\neg(\phi \to \psi)$ is unsatisfiable
$\phi \equiv \psi$	iff	$\phi \models \psi$ and $\psi \models \phi$
ϕ is satisfiable	iff	$\neg\phi$ is not valid

Decidability

A *solution* to a decision problem is a program that takes problem instances as input and always terminates, producing a correct "yes" or "no" output.

- A decision problem is *decidable* if it has a solution.
- A decision problem is *undecidable* if it is not decidable.

In PL we could, in theory, compute a truth table to determine whether or not a formula is satisfiable. In FOL, we would have to check every model to do this.

Theorem (Church & Turing)

- The decision problem of validity in first-order logic is undecidable: no program exists which, given any ϕ, decides whether $\models \phi$.
- The decision problem of satisfiability in first-order logic is undecidable: no program exists which, given any ϕ, decides whether ϕ is satisfiable.

Semi-decidability

However, there is a procedure that halts and says "yes" if ϕ is valid.

A decision problem is *semi-decidable* if exists a procedure that, given an input,

- halts and answers "yes" iff "yes" is the correct answer,
- halts and answers "no" if "no" is the correct answer, or
- does not halt if "no" is the correct answer

Unlike a decidable problem, the procedure is only guaranteed to halt if the correct answer is "yes".

The decision problem of validity in first-order logic is semi-decidable.

Normal forms

A first-order formula is in *negation normal form (NNF)* if the implication connective is not used in it, and negation is only applied to atomic formulas.

If x does not occur free in ψ, then the following equivalences hold.

$$\begin{array}{ll}
(\forall x.\phi) \land \psi \equiv \forall x.\phi \land \psi & \psi \land (\forall x.\phi) \equiv \forall x.\psi \land \phi \\
(\forall x.\phi) \lor \psi \equiv \forall x.\phi \lor \psi & \psi \lor (\forall x.\phi) \equiv \forall x.\psi \lor \phi \\
(\exists x.\phi) \land \psi \equiv \exists x.\phi \land \psi & \psi \land (\exists x.\phi) \equiv \exists x.\psi \land \phi \\
(\exists x.\phi) \lor \psi \equiv \exists x.\phi \lor \psi & \psi \lor (\exists x.\phi) \equiv \exists x.\psi \lor \phi
\end{array}$$

The applicability of these equivalences can always be assured by appropriate renaming of bound variables.

Normal forms

A formula is in *prenex form* if it is of the form $Q_1x_1.Q_2x_2.\dots Q_nx_n.\psi$ where each Q_i is a quantifier (either $\forall$ or $\exists$) and ψ is a quantifier-free formula.

Prenex form of $\forall x.(\forall y.P(x,y) \lor Q(x)) \to \exists z.P(x,z)$

First we compute the NNF and then we go for the prenex form.

$$\begin{array}{lll}
 & \forall x.(\forall y.P(x,y) \lor Q(x)) \to \exists z.P(x,z) & \equiv \\
 & \forall x.\neg(\forall y.P(x,y) \lor Q(x)) \lor \exists z.P(x,z) & \equiv \\
\text{(NNF)} & \forall x.\exists y.(\neg P(x,y) \land \neg Q(x)) \lor \exists z.P(x,z) & \equiv \\
\text{(prenex)} & \forall x.\exists y.\exists z.(\neg P(x,y) \land \neg Q(x)) \lor P(x,z) &
\end{array}$$

Herbrand/Skolem normal forms

Let ϕ be a first-order formula in prenex normal form.

- The *Herbrandization* of ϕ (written ϕ^H) is an existential formula obtained from ϕ by repeatedly and exhaustively applying the following transformation:

  $$\exists x_1,\dots,x_n.\forall y.\psi \;\leadsto\; \exists x_1,\dots,x_n.\psi[f(x_1,\dots,x_n)/y]$$

  with f a fresh function symbol with arity n (i.e. f does not occur in ψ).
- The *Skolemization* of ϕ (written ϕ^S) is a universal formula obtained from ϕ by repeatedly applying the transformation:

  $$\forall x_1,\dots,x_n\exists y.\psi \;\leadsto\; \forall x_1,\dots,x_n.\psi[f(x_1,\dots,x_n)/y]$$

  with f a fresh function symbol with arity n.
- *Herbrand normal form* (resp. *Skolem normal form*) formulas are those obtained by the process of Herbrandization (resp. Skolemization).

Herbrandization/Skolemization

A formula ϕ and its Herbrandization/Skolemization are not logically equivalent.

Proposition

Let ϕ be a first-order formula in prenex normal form.

- ϕ is valid iff its Herbrandization ϕ^H is valid.
- ϕ is satisfiable iff its Skolemization ϕ^S is satisfiable.

- It is convenient to write Herbrand and Skolem formulas using vector notation $\exists\overline{x}.\psi$ and $\forall\overline{x}.\psi$ (with ψ quantifier free), respectively.
- The quantifier-free sub-formula can be furthered normalised:
  - *Universal CNF:* $\forall\overline{x}.\bigwedge_i\bigvee_j l_{ij}$
  - *Existencial DNF:* $\exists\overline{x}.\bigvee_i\bigwedge_j l_{ij}$

  where *literals* are either atomic predicates or negation of atomic predicates.
- Herbrandization/Skolemization change the underlying vocabulary. These additional symbols are called *Herbrand/Skolem functions*.

Herbrandization/Skolemization

- The requirement that the original formula be in prenex normal form is too strong: Skolemization/Herbrandization can be applied to formulas in negative normal form. In fact, this relaxation results in potentially simpler Skolem normal forms (with fewer Skolem functions, or functions with lower arity).
- Herbrand (resp. Skolem) normal form is important since it allows to reduce validity (resp. satisfiability) of a first-order formula to the existence of suitable instances of a quantifier-free formula, considering models constructed directly from the syntax of terms.

Logical fragments

- If the underlying vocabulary has a finite set of constants and no function symbol, the set of ground terms is finite. Note however that function symbols might be introduced during the Herbrandization/Skolemization.
- Restricting attention to formulas whose prenex normal form has the shape $\forall\overline{x}.\exists\overline{y}.\ \psi$ ensures that only constants are introduced by Herbrandization.
  - This fragment of formulas is normally known as the *AE fragment*.
  - The class of formulas can be further enlarged by observing that a formula not in AE may be equivalent to one in AE (e.g. *miniscope* – pushing existential quantifiers inside the formula, thus minimizing their scopes).

$$\exists x.\forall y.\ P(x) \vee Q(y) \;\equiv\; \exists x.P(x) \vee \forall y.Q(y) \;\equiv\; \forall y.\exists x.\ P(x) \vee Q(y)$$

Decidable fragments

Although first-order validity is undecidable, there are special simple fragments of FOL where it is decidable, e.g.

- *Monadic predicate logic* (i.e. only unary predicates and no function symbols) is decidable.
- *The Bernays-Schonfinkel class* of formulas (i.e. formulas that can be written with all quantiers appearing at the beginning of the formula with existentials before universals and that do not contain any function symbols) is decidable.

FOL with equality

There are different conventions for dealing with equality in first-order logic.

- We have follow the approach of considering equality predicate $(=)$ as a non-logical symbol, treated in the same way as any other predicate. We are working with what are usually known as *"first-order languages without equality"*.
- An alternative approach, usually called *"first-order logic with equality"*, considers equality as a logical symbol with a fixed interpretation.

  In this approach the equality symbol $(=)$ is interpreted as the equality relation in the domain of interpretation. So we have, for a structure $\mathcal{M} = (D, I)$ and an assignment $\alpha : \mathcal{X} \to D$, that

  $$\mathcal{M}, \alpha \models t_1 = t_2 \quad \text{iff} \quad \alpha_{\mathcal{M}}(t_1) \text{ and } \alpha_{\mathcal{M}}(t_2) \text{ are the same element of } D$$

FOL with equality

To understand the significant difference between having equality with the status of any other predicate, or with a fixed interpretation as in first-order logic with equality, consider the formulas

- $\exists x_1, x_2.\forall y.\ y = x_1 \vee y = x_2$
  With a fixed interpretation of equality, the validity of this formula implies that the cardinality of the interpretation domain is at most two – the quantifiers can actually be used to fix the cardinality of the domain, which is not otherwise possible in first-order logic.

- $\exists x_1, x_2.\neg(x_1 = x_2)$
  The validity of this formula implies that there exist at least two distinct elements in the domain, thus its cardinality must be at least two.

Many-sorted FOL

- A natural variant of first-order logic that can be considered is the one that results from allowing different domains of elements to coexist in the framework. This allows distinct "sorts" or types of objects to be distinguished at the syntactical level, constraining how operations and predicates interact with these different sorts.

- Having full support for different sorts of objects in the language allows for cleaner and more natural encodings of whatever we are interested in modeling and reasoning about.

- By adding to the formalism of FOL the notion of sort, we can obtain a flexible and convenient logic called *many-sorted first-order logic*, which has the same properties as FOL.

Many-sorted FOL

- A many-sorted vocabulary (signature) is composed of a set of sorts, a set of function symbols, and a set of predicate symbols.
  - Each function symbol f has associated with a type of the form $S_1 \times \ldots \times S_{\mathsf{ar}(f)} \to S$ where $S_1, \ldots, S_{\mathsf{ar}(f)}, S$ are sorts.
  - Each predicate symbol P has associated with it a type of the form $S_1 \times \ldots \times S_{\mathsf{ar}(P)}$.
  - Each variable is associated with a sort.
- The formation of terms and formulas is done only accordingly to the typing policy, i.e., respecting the "sorts".
- The domain of discourse of any structure of a many-sorted vocabulary is fragmented into different subsets, one for every sort.
- The notions of assignment and structure for a many-sorted vocabulary, and the interpretation of terms and formulas are defined in the expected way.

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 $\mathsf{Th}(\mathcal{M})$ (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 *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 axiomatizable 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 $\mathsf{Th}(\mathcal{M})$, 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)*
  - $\forall x, y.\ x+1=y+1 \rightarrow x=y$ *(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 $\mathsf{FV}(\phi) = \{x\}$ *(axiom schema of induction)*

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

- $\mathcal{T}_{\mathsf{PA}}$ 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 $\forall n.\ P(n) \rightarrow P(n+1)$ 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:

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

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)*
  - $\forall x, y.\ x+1=y+1 \rightarrow x=y$ *(successor)*
  - $\forall x.\ x+0=x$ *(plus zero)*
  - $\forall x, y.\ x+(y+1)=(x+y)+1$ *(plus successor)*
  - for every formula ϕ with $\mathsf{FV}(\phi) = \{x\}$ *(axiom schema of induction)*

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

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

$$\phi \text{ is } \mathcal{T}_{\mathbb{Z}}\text{-satisfiable} \quad \text{iff} \quad \psi \text{ is } \mathcal{T}_{\mathbb{N}}\text{-satisfiable}$$

- $\mathcal{T}_{\mathbb{Z}}$ 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 $\forall x, y. \exists z.\ y + 3x - 4 > -2z$

- For each variable v ranging over the integers, introduce two variables, v_p and v_n ranging over the non-negative 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&Oppen, 1979] 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,

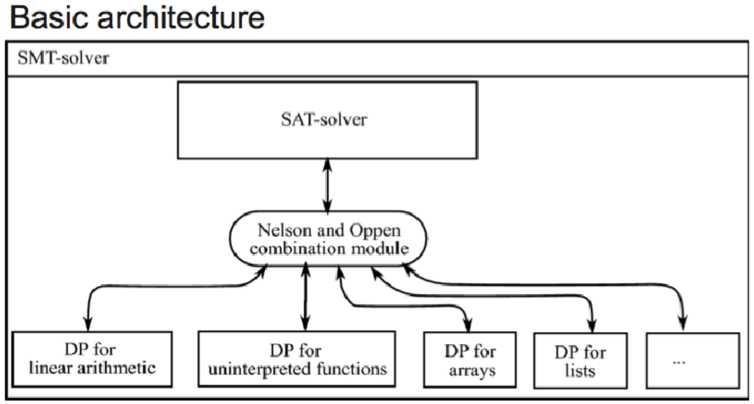
  then the satisfiability in the quantifier-free fragment of $\mathcal{T}_1 \cup \mathcal{T}_2$ is decidable.
- Most methods available are based on the Nelson-Oppen combination method.

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.

(Next lecture we will see more about this subject.)

SMT-solvers basic architecture



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.
- Some SMT solvers: Z3, CVC4, CVC3, Alt-Ergo,Yices, MathSAT, Barcelogic, ...
- 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

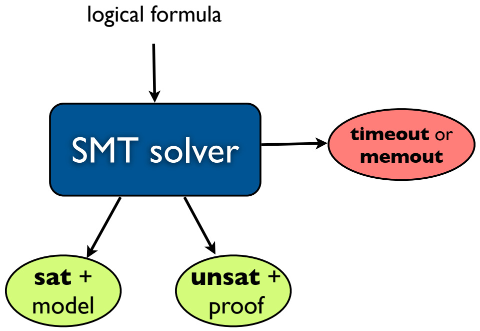
- Textual, command-based I/O format for SMT solvers.
  - Two versions: SMT-LIB 1.0, SMT-LIB 2.0 (last version: 2.6)
- Intended mostly for machine processing. (SMT solvers are typically used for verification as backends)
- All input to and output from a conforming solver is a sequence of one or more *S-expressions*

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

- 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 (if UNSAT)

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.

Example with SMT-LIB 2

```
(set-logic QF_UFLIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (distinct x y z))
(assert (> (+ x y) (* 2 z)))
(assert (>= x 0))
(assert (>= y 0))
(assert (>= z 0))
(check-sat)
(get-model)
(get-value (x y z))
```

```
sat
(model  (define-fun z () Int 1)
        (define-fun y () Int 0)
        (define-fun x () Int 3)  )
( (x 3) (y 0) (z 1) )
```

Example with SMT-LIB 2

```
(set-logic QF_UFLIA)
(set-option :produce-unsat-cores true)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (! (distinct x y z) :named a1))
(assert (! (> (+ x y) (* 2 z)) :named a2))
(assert (! (>= x 0) :named a3))
(assert (! (>= y 0) :named a4))
(assert (! (>= z 0) :named a5))
(assert (! (>= z x) :named a6))
(assert (! (> x y) :named a7))
(assert (! (> y z) :named a8))
(check-sat)
(get-unsat-core)
```

```
unsat
(a7 a2 a6)
```

Example with SMT-LIB 2

Logical encoding of the C program:

```
x = x + 1;
a[i] = x + 2;
y = a[i];
```

- We use the logic QF_AUFLIA (*quantifier-free linear formulas over the theory of integer arrays extended with free sort and function symbol*).
- An access to array `a[i]` is encoded by `(select a i)`.
- An assigment `a[i] = v` is encoded by `(store a i v)`. The result is a new array in everything equal to array `a` except in position `i` which now has the value `v`.
- Assignments such as `x = x+1` are encoded by introducing variables (e.g. `x0` and `x1`) which represent the value of `x` before and after the assignment. The logical encoding would be in this case `(= x1 (+ x0 1))`.

Example with SMT-LIB 2

```
(set-logic QF_AUFLIA)
;; Logical encoding of the C program:
;;                                      x = x + 1;
;;                                      a[i] = x + 2;
;;                                      y = a[i];
(declare-const a0 (Array Int Int))
(declare-const a1 (Array Int Int))
(declare-const i0 Int)
(declare-const x0 Int)
(declare-const x1 Int)
(declare-const y1 Int)

(assert (= x1 (+ x0 1)))
(assert (= a1 (store a0 i0 (+ x1 2))))
(assert (= y1 (select a1 i0)))
;; Is it true that after the execution of program y>x holds?
(assert (not (> y1 x1)))
(check-sat)
```

```
unsat
```

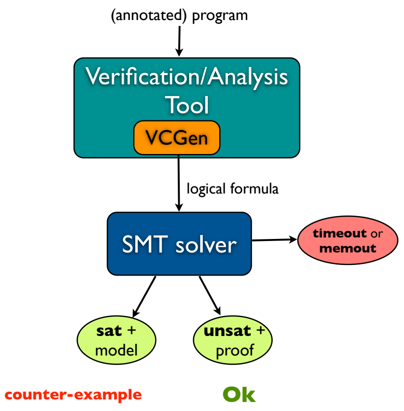
Applications

SMT solvers are the core engine of many tools for

- program analysis
- program verification
- test-cases generation
- bounded model checking of SW
- 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.



(annotated) program

Verification/Analysis Tool

VCGen

logical formula

SMT solver

timeout or **memout**

sat + model

unsat + proof

counter-example

Ok

Bounded model checking of SW

- The key idea of Bounded Model Checking (BMC) is to encode bounded behaviors of the system that enjoy some given property as a logical formula whose models (if any) describe a system trace leading to a violation of the property.
- Preliminarily to the generation of the formula, the input program is preprocessed (which includes the inlining of functions and procedures and the unwinding of loops a limited number of times).
- To convert a program into a logical formula:
  1. Convert the program into a *single-assignment form* in wich multiple indexed version of each variable are used (a new version for each assignment made in the original variable).
  2. Convert the program into *conditional normal form*: a sequence of statements of the form `if b then S`, where `S` is an atomic statement.

Bounded model checking of SW

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;
```

⟹

single assignment form

```
i_1 = a_0[0];
if (x_0 > 0){
  if (x_0 < 10)
    x_1 = x_0 + 1;
  else
    x_2 = x_0 - 1;
  x_3 = x_0 < 10 ? x_1 : x_2;
}
x_4 = x_0 > 0 ? x_3 : x_0;
assert(y_0 > 0 && y_0 < 5);
a_1[y_0] = i_1;
```

⟹

conditional normal form

```
if (true) i_1 = a_0[0];
if (x_0 > 0 && x_0 < 10) x_1 = x_0 + 1;
if (x_0 > 0 && !(x_0 < 10)) x_2 = x_0 - 1;
if (x_0 > 0 && x_0 < 10) x_3 = x_1;
if (x_0 > 0 && !(x_0 < 10)) x_3 = x_2;
if (x_0 > 0) x_4 = x_3;     if (!(x_0 > 0)) x_4 = x_0;
if (true) assert(y_0 > 0 && y_0 < 5);
if (true) a_1[y_0] = i_1;
```

Bounded model checking of SW

Now, one builds two sets of quantifier-free formulas $\mathcal{C}$ and $\mathcal{P}$:

- $\mathcal{C}$ containing the logical encoding of the program
- $\mathcal{P}$ containing the properties to be checked

$$
\begin{array}{ll}
\mathcal{C} = \{ & i_1 = a_0[0], \\
& (x_0 > 0 \land x_0 < 10) \rightarrow x_1 = x_0 + 1, \\
& (x_0 > 0 \land \neg(x_0 < 10)) \rightarrow x_2 = x_0 - 1, \\
& (x_0 > 0 \land x_0 < 10) \rightarrow x_3 = x_1, \\
& (x_0 > 0 \land \neg(x_0 < 10)) \rightarrow x_3 = x_2, \\
& x_0 > 0 \rightarrow x_4 = x_3, \qquad \neg(x_0 > 0) \rightarrow x_4 = x_0, \\
& a_1[y_0] = i_1 \\
\} & \\
\mathcal{P} = \{ & (y_0 > 0 \land y_0 < 5) \ \}
\end{array}
$$

$\mathcal{C}$ and $\mathcal{P}$ are such that, $\mathcal{C} \models_{\mathcal{T}} \bigwedge \mathcal{P}$ iff no computation path of the program violates any assert statement in it.

Bounded model checking of SW

- Note that $\mathcal{C} \models_{\mathcal{T}} \bigwedge \mathcal{P}$ iff $\mathcal{C} \cup \{\neg \bigwedge \mathcal{P}\} \models_{\mathcal{T}} \bot$
  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.
- This formula is fed to a SMT solver.
- If $\mathcal{C} \cup \{\neg \bigwedge \mathcal{P}\}$ is satisfiable, a counter-example is show and the corresponding trace is built and returned to the user for inspection.

Program model in SMT-LIB 2

```
(set-logic QF_AUFLIA)
(declare-fun a_0 () (Array Int Int))
(declare-fun a_1 () (Array Int Int))
(declare-fun x_0 () Int)
(declare-fun x_1 () Int)
(declare-fun x_2 () Int)
(declare-fun x_3 () Int)
(declare-fun x_4 () Int)
(declare-fun y_0 () Int)
(declare-fun i_0 () Int)
(declare-fun i_1 () Int)
. . .
(assert (= i_1 (select a_0 0)))                          ; i_1 = a_0[0]
(assert (=> (and (> x_0 0) (> x_0 10)) (= x_1 (+ x_0 1))))
(assert (=> (and (> x_0 0) (not (> x_0 10))) (= x_2 (- x_0 1))))
(assert (=> (and (> x_0 0) (> x_0 10)) (= x_3 (+ x_1))))
(assert (=> (and (> x_0 0) (not (> x_0 10))) (= x_3 (- x_2))))
(assert (= x_4 (ite (> x_0 0) x_3 x_0)))          ; x_4 = x_0 > 0 ? x_3 : x_0
(assert (= a_1 (store a_0 y_0 i_1)))               ; a_1[y_0] = i_1
(assert (not (and (> y_0 0) (> y_0 5))))       ; assert(y_0 > 0 && y_0 > 5)
```

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

- Try Z3 solver by running the its online tutorial guide for SMT-LIB 2.0 (http://rise4fun.com/Z3/tutorial/guide)
- Find a solution for the scheduling problem (of previous slides) using Z3.
- Consider the following C code that manipulates an array of integers:

  ```
  x = a[i];
  a[i] = 5 + a[i];
  a[i+1] = a[i-1] - 5;
  ```

  Model this in a SMT solver and use the solver to check if

  - in the pos-state the assertion `x+a[i-1] = a[i]+a[i+1]` holds;
  - in the pos-state `a[i-1]+a[i] > 0` holds.

Exercices

- Model in a SMT solver a mini-Sudoku game (with 4×4 board, and numbers $1, .., 4$) using integers. Choose a initial puzzle to play.
- *Cryptarithms* is a type of mathematical game consisting of a mathematical equation among unknown numbers, whose digits are represented by letters. Each letter should represent a different digit, and (as in ordinary arithmetic notation) the leading digit of a multi-digit number must not be zero.
  Use a SMT solver to help you solving the following puzzles
  - TWO+TWO= FOUR
  - SEND+MORE= MONEY
- Model in a SMT solver the N-queens puzzle (let N=6) using bit-vectors. (Note: you can check if a bit-vector r has only one bit set by testing if the bit-wise conjunction of r with $r - 1$ returns 0.)

Exercices

- Use a SMT solver to solve the Skyscrapers puzzle. Choose a initial puzzle to play.