CS257: Introduction to Automated Reasoning

First-order logic: Syntax

Stanford University

CENTAUR

Motivation

Consider reasoning about the following sentences in propositional logic.

English	prop. logic
Every natural number is larger than 0	K
Not every natural number is larger than 0	$\neg K$

What facts can we logically deduce?

Propositional logic is sometimes too crude to mirror intuitively correct deductions.

First-order logic allows us to (dis)prove the validity of sentences like the above.

In this case, we need a first-order language for number theory.

Motivation

"Every natural number is larger than 0."

Intuitively, this first-order language needs to have the following features:

English	Formal language
The number 0	$\mathbf{0}$
"v_1 is greater than v_2"	$> \mathbf{v_1 v_2}$
"For every natural number"	$\forall$

Constant

Function/predicate

Quantifier

Motivation

"Every natural number is larger than 0."

Intuitively, this first-order language needs to have the following features:

English	Formal language
The number 0	$\mathbf{0}$
"v_1 is greater than v_2"	$> \mathbf{v_1 v_2}$
"For every natural number"	$\forall$

"Every natural number is larger than 0." translates to $\forall\, v_1 \; > v_1 0$

This sentence is false in the intended translation.

Plan for this week

- Syntax (MI 2.1)
- Semantics (MI 2.2)
- Proof rules for first-order logic (CC 2.3)
- Clausal Form (CC 2.5)

MI presents an single-typed first-order logic.

We will present a **many-sorted first-order logic (FOL)**.

This makes it convenient to present Satisfiability modulo Theories (starting Week 4).

Many-sorted FOL is not more expressive than single-sorted FOL.
See MI 4.3 for reducing many-sorted logic to a single sorted one.

* Some of the slides today are contributed by Clark Barrett.

Symbols

Review: what does the syntax of a logic consist of?

First-order logic is an umbrella term for different **first-order languages**. The symbols of a first-order language consist of:

1. **Logical symbols**
   - Parentheses: (,)
   - Propositional connectives: $\rightarrow$, $\neg$
   - Variables: v_1, v_2, ...
   - Quantifier: $\forall$
2. **Signature**, $\Sigma := \left\langle \Sigma^S, \Sigma^F \right\rangle$, where:
   - Σ^S is a set of **sorts**: e.g., $\mathsf{Real}, \mathsf{Int}, \mathsf{Set}$, $\mathfrak{D}$, $\boldsymbol{\odot}$
   - Σ^F is a set of **function symbols**: e.g., $+$, $+_{[2]}$, $<$, $\between$
     - For each sort σ in Σ^S, there may be an optional **equality symbol** $=_\sigma$ in Σ^F

Note 1: we require that no symbol is a finite sequence of others.
Note 2: we have infinitely many distinct symbols.

Abbreviations

- Propositional connectives: $\vee$, $\wedge$, $\leftrightarrow$
- Existential quantifier: express $\exists v$ with $\neg \forall v \neg$

Signature

The syntax of a first-order language is defined w.r.t. a **signature**, $\Sigma := \left\langle \Sigma^S, \Sigma^F \right\rangle$, where:

- Σ^S is a set of **sorts**: e.g., $\mathsf{Real}, \mathsf{Int}, \mathsf{Set}$, ☽, ☉
- Σ^F is a set of **function symbols**: e.g., $+$, $+_{[\mathbf{2}]}$, $<$, $\between$

We associate each variable symbol v with a sort in Σ^S, denoted $\mathit{sort}(v)$.

We associate each function symbol $f \in \Sigma^F$ with:

- an **arity** n: a natural number denoting the number of arguments f takes
- an $(n+1)$-tuple of sorts: $\mathit{sort}(f) = \langle \sigma_1, \ldots, \sigma_n, \sigma_{n+1} \rangle$

We say f **returns** σ_{n+1}.

Example: In the first-order language of number theory

- Σ^S contains a sort Nat
- For each variable v, $\mathit{sort}(v) = \mathsf{Nat}$
- Σ^F contains a function $+$
- $+$ has arity 2 and $\mathit{sort}(+) = \langle \mathsf{Nat}, \mathsf{Nat}, \mathsf{Nat} \rangle$

Signature

We assume Σ^S implicitly includes a distinguished sort Bool

We assume Σ^F implicitly contains distinguished symbols $\{\top, \bot\}$ and $\mathit{sort}(\bot) = \mathit{sort}(\top) = \langle \text{Bool} \rangle$

There are two special kinds of function symbols:

- **Constant symbol**: a function symbol with 0 arity (e.g., $\bot$, $\top$, π, John, **0**)
- **Predicate symbol**: a function symbol that returns Bool
  - Each equality symbol $=_\sigma$ is a predicate symbol with $\mathit{sort}(=_\sigma) = \langle \sigma, \sigma, \text{Bool} \rangle$
  - $\mathit{sort}(<) = \langle \text{Nat}, \text{Nat}, \text{Bool} \rangle$

First-Order Languages: Examples

A first-order language is defined w.r.t. a signature $\Sigma := \langle \Sigma^S, \Sigma^F \rangle$. To specify a signature:

1. say what are the sorts;
2. say whether the equality symbol is present for each sort;
3. say what are the other function symbols.

Set Theory

- $\Sigma^S : \{\mathrm{Set}, \mathrm{Bool}\}$
- Equality: **yes** for Set
- $\Sigma^F : \{\in, \varnothing, =_{\mathrm{Set}}\}$

where:

- $sort(\in) = \langle \mathrm{Set}, \mathrm{Set}, \mathrm{Bool} \rangle$
- $sort(\varnothing) = \langle \mathrm{Set} \rangle$

First-Order Languages: Examples

A first-order language is defined w.r.t. a signature $\Sigma := \left\langle \Sigma^S, \Sigma^F \right\rangle$. To specify a signature:

1. say what are the sorts;
2. say whether the equality symbol is present for each sort;
3. say what are the other function symbols.

Elementary Number Theory

- $\Sigma^S : \{\mathsf{Nat}, \mathsf{Bool}\}$
- Equality: **yes** for Nat
- $\Sigma^F : \{<, 0, S, +, \times, =_{\mathsf{Nat}}\}$

where:

- $sort(<) = \langle \mathsf{Nat}, \mathsf{Nat}, \mathsf{Bool} \rangle$
- $sort(0) = \langle \mathsf{Nat} \rangle$
- $sort(S) = \langle \mathsf{Nat}, \mathsf{Nat} \rangle$
- $sort(+/\times) = \langle \mathsf{Nat}, \mathsf{Nat}, \mathsf{Nat} \rangle$

Expressions

Recall from Lecture 1, an expression is any finite sequence of symbols.
For example:

- $\forall v_1((< 0 v_1) \rightarrow (\neg \forall v_2(< v1v2)))$
- $v_1 < \forall v_2))$

Most expressions are nonsensical.

Expressions of interest in first-order logic are the **terms** and the **well-formed formulas (wffs)**.

Terms

Terms are building blocks of wffs in a first-order language.

Concretely, terms are expressions that can be built up from the constant symbols and the variables by prefixing the function symbols.

Formally, let $\mathcal{B}$ be the set of all variables and the constant symbols.
For each non-constant function symbol $f \in \Sigma^F$ (i.e., with arity $n > 0$), we define a term-building operation $\mathcal{F}_f$:

$$\mathcal{F}_f(\alpha_1, \ldots, \alpha_n) = f\alpha_1, \ldots, \alpha_n$$

Denote this set of operations $\mathcal{F}$.

Terms are expressions that are generated by $\mathcal{F}$ from $\mathcal{B}$.

Examples of terms in the language of number theory:

- $+v_2S0$
- $SSSS0$
- $S < 00$

We do not want terms like $S < 00$, because S takes as argument terms with sort Nat but < 00 has sort Bool.

Well-sorted terms

We formulate the notion of **well-sortedness**.

We define $\overline{sort}$, a function from terms to sorts as follows:

- If v is a variable, then $\overline{sort}(v) = sort(v)$.
- If f is a constant, where $sort(f) = \langle\sigma\rangle$, then $\overline{sort}(f) = \sigma$.
- If $t = ft_1 \ldots t_n$, where $sort(f) = \langle\sigma_1, \ldots, \sigma_n, \sigma_{n+1}\rangle$, then $\overline{sort}(t) = \sigma_{n+1}$.

We define a function *well* from terms to $\{1, 0\}$.

- For every variable v, $well(v) = 1$.
- For every constant f, $well(f) = 1$.
- If $t = ft_1 \ldots t_n$, where $sort(f) = \langle\sigma_1, \ldots, \sigma_n, \sigma_{n+1}\rangle$, $well(t) = 1$ iff $(well(t_1) = 1) \wedge \cdots \wedge (well(t_n) = 1) \wedge (\overline{sort}(t_1) = \sigma_1) \wedge \cdots \wedge (\overline{sort}(t_n) = \sigma_n)$.

A term t is **well-sorted** if $well(t) = 1$.

Well-sorted terms: example

Elementary Number Theory

Let $\Sigma^S = \{\mathrm{Nat}, \mathrm{Bool}\}$ and $\Sigma^F = \{0, S, +, \times, <, =_{\mathrm{Nat}}\}$.
Suppose we have variables v_i where $sort(v_i) = \mathrm{Nat}$ for all v_i. Define $sort$ as follows:

- $sort(0) = \langle \mathrm{Nat} \rangle$
- $sort(S) = \langle \mathrm{Nat}, \mathrm{Nat} \rangle$
- $sort(+/\times) = \langle \mathrm{Nat}, \mathrm{Nat}, \mathrm{Nat} \rangle$
- $sort(< / =_{\mathrm{Nat}}) = \langle \mathrm{Nat}, \mathrm{Nat}, \mathrm{Bool} \rangle$

Are the following well-sorted?

- $+0v_5$
- $+ + 0v_5$
- $S + 0v_5$
- $=_{\mathrm{Nat}}\ S\ v_3 + 1\ v_1$

Note: we are using prefix notation. In practice, there are first-order languages for which it is more standard to use infix notation.

Σ-Formulas

An **atomic formula** is a well-sorted term t with $\overline{sort}(t) = \mathsf{Bool}$.

Example: $=_{\mathsf{Nat}}\ 0\ S0$

We define the following formula-building operations, denoted $\mathcal{F}$:

- $\mathcal{E}_{\neg}(\alpha) = (\neg\alpha)$
- $\mathcal{E}_{\rightarrow}(\alpha, \beta) = (\alpha \rightarrow \beta)$
- For each variable v, $\mathcal{Q}_v(\alpha) = \forall\ v\ \alpha$

Given a signature Σ, the set of **well-formed formulas** (also called **Σ-formulas**) is the set of expressions generated from the atomic formulas by $\mathcal{F}$.

Let $\Sigma_N = \left\langle \Sigma^S := \{\mathsf{Nat}\}, \Sigma^F := \{0, S, +, \times, <, =_{\mathsf{Nat}}\} \right\rangle$. Are the following Σ_N-formulas?

$=_{\mathsf{Nat}}\ +v_1 0 v_2$	**yes**
$+0v_1$	**no**
$\forall\ v_1\ \ =_{\mathsf{Nat}}\ +0v_1v_1$	**yes**

Σ-Formulas

An **atomic formula** is a well-sorted term t with $\overline{sort}(t) = \mathsf{Bool}$.

We define the following formula-building operations, denoted $\mathcal{F}$:

- $\mathcal{E}_{\neg}(\alpha) = (\neg\alpha)$
- $\mathcal{E}_{\rightarrow}(\alpha, \beta) = (\alpha \rightarrow \beta)$
- For each variable v, $\mathcal{Q}_v(\alpha) = \forall\, v\, \alpha$

Given a signature Σ, the set of **well-formed formulas** (also called **Σ-formulas**) is the set of expressions generated from the atomic formulas by $\mathcal{F}$.

Exercise: draw a Venn Diagram that illustrates the relations between A: terms, B: well-sorted terms, C: atomic formulas, D: well-formed formulas, and E: expressions.

Describe the relations between B, C, and D, and submit your answer to

`https://pollev.com/andreww095`

Free and Bound Variables

We define a recursive function *free* from Σ-formulas and variables to $\{1, 0\}$ to capture what it means for a variable x to **occur free** in a **wff** α:

- When α is an atomic formula, then $\mathit{free}(\alpha, x) = 1$ iff x occurs in α;
- When $\alpha := (\neg\beta)$, then $\mathit{free}(\alpha, x) = \mathit{free}(\beta, x)$;
- When $\alpha := (\beta \to \gamma)$, then $\mathit{free}(\alpha, x) = \max(\mathit{free}(\beta, x), \mathit{free}(\gamma, x))$;
- When $\alpha := \forall\, v\, \beta$, then $\mathit{free}(\alpha, x) = \mathit{free}(\beta, x)$ if $x \neq v$, and 0 otherwise.

If $\forall\, v$ appears in α, then v is said to be **bound** in α.

Can a variable both occur free and be bound in α?

This can be confusing, so we typically require the set of free and bound variables to be disjoint.

We say a Σ-formula α is **closed** or α is a **sentence**, if no variable occurs free in α.

Induction and recursion

- To define a set C **inductively**:
  1. Define a universe U. (e.g., set of expressions)
  2. Define a base set $\mathcal{B} \subseteq U$. (e.g., set of atomic formulas)
  3. Define a family of building operators, $\mathcal{F}$, each of which takes one or more element of U as arguments and returns an element of U. (e.g., One for each of $\neg$, $\rightarrow$, $\forall$)

C is defined to be the set generated from $\mathcal{B}$ by $\mathcal{F}$ (e.g., wffs).

- To define a function h on C **recursively**:
  1. Define $h(b)$ for each $b \in \mathcal{B}$. (e.g., define *free* on atomic formulas)
  2. For each $f \in F$, define the value of $h(f(\alpha_1, \ldots, \alpha_k))$ in terms of $h(\alpha_1), \ldots, h(\alpha_k)$. (e.g., define *free* on $(\neg\beta)$ in terms of $\mathit{free}(\beta)$)

In general, is h always well-defined? **No!**

Induction and Recursion: Pitfalls

Consider the following inductive definition:

- Universe U: the set of real numbers
- Base set $\mathcal{B}$: $\{0\}$
- Building operators $\mathcal{F}$: $f(x, y) = x \cdot y$ and $g(x) = x + 1$

Now define h recursively as:

- $h(0) = 0$
- $h(f(x, y)) = h(x) + h(y)$
- $h(g(x)) = h(x) + 2$

Is h well-defined? Try computing $h(1)$?

$h(1) = h(g(0)) = h(0) + 2 = 2$
$h(1) = h(f(g(0), g(0))) = h(g(0)) + h(g(0)) = 2 + 2 = 4$ Why does this happen?

Induction and Recursion

We say C is **freely generated** from $\mathcal{B}$ by $\mathcal{F}$ iff C is generated by $\mathcal{B}$, and in addition:

- The range of each $f \in \mathcal{F}$ is disjoint from the ranges of all other functions in $\mathcal{F}$ and from $\mathcal{B}$
- each $f \in \mathcal{F}$ is one-to-one

The Recursion Theorem: Let C be the set freely generated from $\mathcal{B}$ by $\mathcal{F}$. Assume $\mathcal{V}$ is a set, $h_0 : \mathcal{B} \mapsto \mathcal{V}$ is a function, and $h_f : \mathcal{V}^k \mapsto \mathcal{V}$ for each $f \in \mathcal{F}$ with arity $k > 0$.

Then there exists a unique function $h : C \mapsto V$, such that:

- $h(b) = h_0(b)$ for each $b \in \mathcal{B}$;
- for each $f \in \mathcal{F}$, $h(f(\alpha_1, \ldots, \alpha_k)) = h_f(h(\alpha_1), \ldots, h(\alpha_k))$

To show a recursive function h on an inductive set C is well-defined, it suffices to show that C is freely generated.

Induction and Recursion: Unique Readability Theorem

Theorem: the set of terms is freely generated from the set of variables and constant symbols by the term-building operations.

Proof: First, given $f, g \in F$, where $f \neq g$, the range of f is clearly disjoint from the range of g, because they result in terms with different prefixes. Further, f's range is also disjoint from the set of variables and constant symbols.

It remains to show that f is one-to-one. That is, suppose f has arity n, for any terms $t_1, \ldots, t_n, t'_1, \ldots, t'_n$, if $ft_1 \ldots t_n = ft'_1 \ldots t'_n$, then $t_1 = t'_1, \ldots$, and $t_n = t'_n$.

The proof makes use of the following fact, which you will prove in the homework.

Lemma A: No proper initial segment of a term is itself a term.

By deleting the first symbol, we have $t_1 \ldots t_n = t'_1 \ldots t'_n$.

t_1 must be equal to t'_1, because otherwise, one would be a proper initial segment of the other, contradicting Lemma A. The same argument can be repeated to show $t_2 \ldots t_n = t'_2 \ldots t'_n$.

Theorem: the set of formulas is freely generated from the atomic formulas and the formula-building operations.