IA008: Computational Logic

2. First-Order Logic

Achim Blumensath
blumens@fi.muni.cz

Faculty of Informatics, Masaryk University, Brno

Basic Concepts

First-Order Logic

Syntax

- variables $x, y, z, \ldots$
- terms $x, f(t_0, \ldots, t_n)$
- relations $R(t_0, \ldots, t_n)$ and equality $t_0 = t_1$
- operators $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
- quantifiers $\exists x \varphi$, $\forall x \varphi$

Semantics

$$\mathfrak{A} \models \varphi(\bar{a}) \qquad \mathfrak{A} = \langle A, R_0, R_1, \ldots, f_0, f_1, \ldots \rangle$$

Examples

$$\varphi := \forall x \exists y [f(y) = x],$$
$$\psi := \forall x \forall y \forall z [x \leq y \wedge y \leq z \rightarrow x \leq z].$$

Examples

Structures

- graphs $\mathfrak{G} = \langle V, E \rangle$

  $E \subseteq V \times V$ binary relation

- words $\mathfrak{W} = \langle W, \leq, (P_a)_a \rangle$

  $\leq \subseteq W \times W$ linear ordering

  $P_a \subseteq W$ positions with letter a

- transition systems $\mathfrak{S} = \langle S, (E_a)_a, (P_i)_i \rangle$

  $E_a \subseteq V \times V$ binary relation

  $P_i \subseteq V$ unary relation

Examples

Graphs $\mathfrak{G} = \langle V, E \rangle,\ E \subseteq V \times V$

- 'The graph is undirected.' (i.e., E is symmetric)

$$\forall x \forall y [E(x,y) \rightarrow E(y,x)]$$

- 'The graph has no isolated vertices.'

$$\forall x \exists y [E(x,y) \vee E(y,x)]$$

- 'Every vertex has outdegree 1.'

$$\forall x \exists y [E(x,y) \wedge \forall z [E(x,z) \rightarrow z = y]]$$

Satisfiability

Theorem

Satisfiability for first-order logic is **undecidable.**

Proof

Turing machine $\mathcal{M} = \langle Q, \Sigma, \Delta, q_0, F_+, F_- \rangle$

Q	set of states
Σ	tape alphabet
Δ	set of transitions $\langle p, a, b, m, q \rangle \in Q \times \Sigma \times \Sigma \times \{-1, 0, 1\} \times Q$
q_0	initial state
F_+	accepting states
F_-	rejecting states

Encoding in FO

$S_q(t)$	**state** q at time t
$h(t)$	**head** in field $h(t)$ at time t
$W_a(t, k)$	**letter** a in field k at time t
s	**successor** function $s(n) = n + 1$
0	**zero**

$$\varphi_w := \mathrm{ADM} \wedge \mathrm{INIT} \wedge \mathrm{TRANS} \wedge \mathrm{ACC}$$

Proof

$S_q(t)$	**state** q at time t
$h(t)$	**head** in field $h(t)$ at time t
$W_a(t,k)$	**letter** a in field k at time t
s	**successor** function $s(n) = n + 1$
0	**zero**

Admissibility formula

$$\begin{aligned} \mathrm{ADM} := \forall t \bigwedge_{p \neq q} \neg[S_p(t) \wedge S_q(t)] & \qquad \text{unique state} \\ \wedge\ \forall t \forall k \bigwedge_{a \neq b} \neg[W_a(t,k) \wedge W_b(t,k)] & \qquad \text{unique letter} \end{aligned}$$

Proof

$S_q(t)$	**state** q at time t
$h(t)$	**head** in field $h(t)$ at time t
$W_a(t,k)$	**letter** a in field k at time t
s	**successor** function $s(n) = n + 1$

Initialisation formula for input: $a_0 \dots a_{n-1}$

$$\begin{aligned} \text{INIT} := {} & S_{q_0}(0) && \text{initial state} \\ & \wedge h(0) = 0 && \text{initial head position} \\ & \wedge \bigwedge_{k<n} W_{a_k}(0,\underline{k}) \wedge \forall k[k \geq \underline{n} \to W_{\Box}(0,k)] && \text{initial tape content} \end{aligned}$$

(here $\underline{k} := s(s(\cdots s(0)))$ and $k \geq \underline{n} := \bigwedge_{i<n} k \neq \underline{i}$)

Acceptance formula

$$\text{ACC} := \exists t \bigvee_{q \in F_+} S_q(t) \qquad \text{accepting state}$$

Proof

$S_q(t)$	**state** q at time t
$h(t)$	**head** in field $h(t)$ at time t
$W_a(t,k)$	**letter** a in field k at time t
s	**successor** function $s(n) = n + 1$

Transition formula

$$\begin{aligned}\mathrm{TRANS} := \forall t \bigvee_{\langle p,a,b,m,q\rangle \in \Delta} \big[& S_p(t) \wedge W_a(t, h(t)) \wedge S_q(s(t)) \wedge \\ & h(s(t)) = h(t) + m \wedge W_b(s(t), h(t)) \big] \\ \wedge\ \forall t \forall k \bigwedge_{a \in \Sigma} \big[& k \neq h(t) \rightarrow [W_a(t,k) \leftrightarrow W_a(s(t),k)] \big]\end{aligned}$$

where

$$y = x + m := \begin{cases} y = s(x) & \text{if } m = 1\,, \\ y = x & \text{if } m = 0\,, \\ s(y) = x & \text{if } m = -1\,. \end{cases}$$

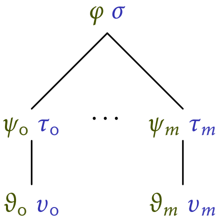
Tableaux

Tableau Proofs

For simplicity: first-order logic **without equality**

Statements φ true or φ false

Rule



Interpretation

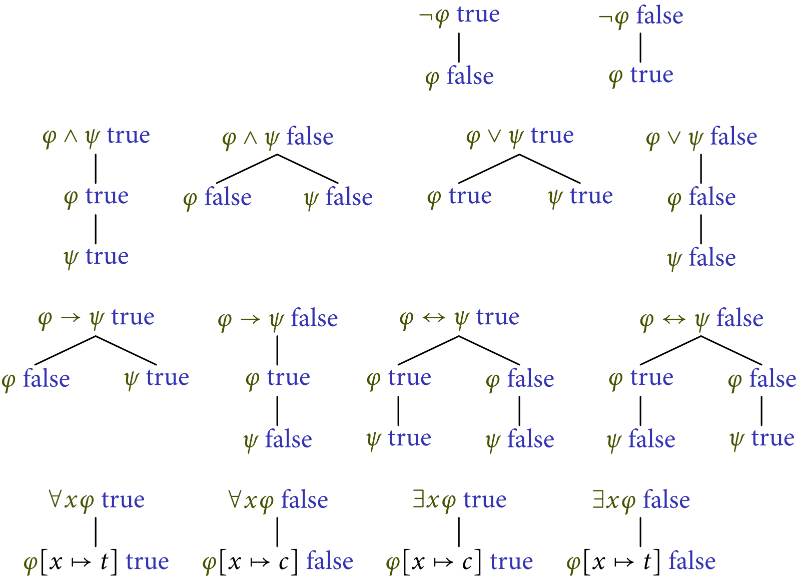
If $\varphi\ \sigma$ is **possible** then so is $\psi_i\ \tau_i, \dots, \vartheta_i\ \upsilon_i$, for some i.

Tableaux

Construction

A **tableau** for a formula φ is constructed as follows:

- start with φ false
- choose a branch of the tree
- choose a statement ψ value on the branch
- choose a rule with head ψ value
- add it at the bottom of the branch
- repeat until every branch contains both statements ψ true and ψ false for some formula ψ



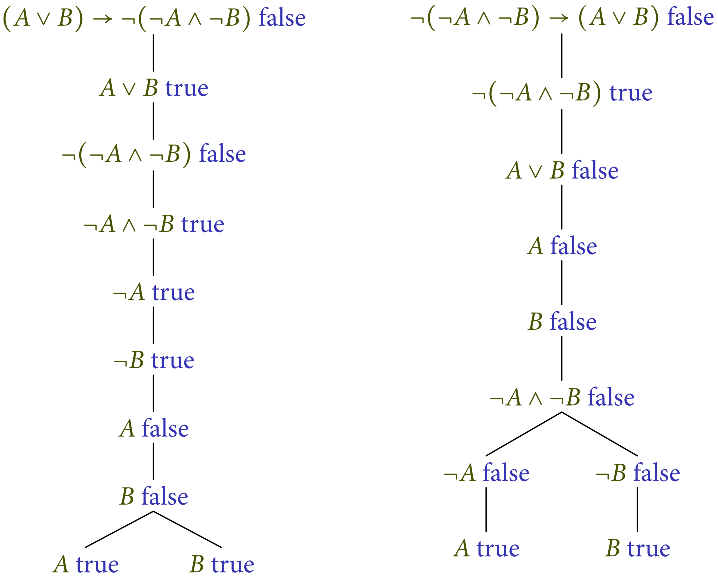
c a new constant symbol, t an arbitrary term

Example

$(A \lor B) \to \neg(\neg A \land \neg B)$ false

$\neg(\neg A \land \neg B) \to (A \lor B)$ false

Example


(A ∨ B) → ¬(¬A ∧ ¬B) false
A ∨ B true
¬(¬A ∧ ¬B) false
¬A ∧ ¬B true
¬A true
¬B true
A false
B false
A true
B true
¬(¬A ∧ ¬B) → (A ∨ B) false
¬(¬A ∧ ¬B) true
A ∨ B false
A false
B false
¬A ∧ ¬B false
¬A false
¬B false
A true
B true

Example

$\exists x \forall y R(x, y) \rightarrow \forall y \exists x R(x, y)$ false

$\forall x R(x, x) \rightarrow \forall x \exists y R(f(x), y)$ false

Example

$\exists x \forall y R(x,y) \to \forall y \exists x R(x,y)$ false

$\exists x \forall y R(x,y)$ true

$\forall y \exists x R(x,y)$ false

$\forall y R(c,y)$ true

$\exists x R(x,d)$ false

$R(c,d)$ true

$R(c,d)$ false

$\forall x R(x,x) \to \forall x \exists y R(f(x),y)$ false

$\forall x R(x,x)$ true

$\forall x \exists y R(f(x),y)$ false

$\exists y R(f(c),y)$ false

$R(f(c),f(c))$ false

$R(f(c),f(c))$ true

Soundness and Completeness

Theorem

A first-order formula φ is valid if, and only if, there exists a tableau T for φ false where every branch is contradictory.

Corollary

Validity of first-order formulae is **recursively enumerable,** but **not decidable.**

Soundness and Completeness

Theorem

A first-order formula φ is valid if, and only if, there exists a tableau T for φ false where every branch is contradictory.

Terminology

A tableau **for** a statement φ value is a tableau T where the root is labelled with φ value.

A branch β is **contradictory** if it contains both statements ψ true and ψ false, for some formula ψ.

A branch β is **consistent with** a structure $\mathfrak{A}$ if

- $\mathfrak{A} \models \psi$, for all statements ψ true on β and
- $\mathfrak{A} \not\models \psi$, for all statements ψ false on β.

A branch β is **complete** if, for every atomic formula ψ, it contains one of the statements ψ true or ψ false.

Proof Sketch: Soundness

Lemma

If β is consistent with $\mathfrak{A}$ and we extend the tableau by applying a rule, the new tableau has a branch β' extending β that is consistent with $\mathfrak{A}$.

Corollary

If $\mathfrak{A} \not\models \varphi$, then every tableau for φ false has a branch that is not contradictory.

Corollary

If φ is not valid, there is no tableau for φ false where all branches are contradictory.

Proof Sketch: Completeness

Lemma

If every tableau for φ false has a non-contradictory branch, there exists a tableau for φ false with a branch β that is complete and non-contradictory.

Lemma

If a branch β is complete and non-contradictory, there exists a structure $\mathfrak{A}$ such that β is consistent with $\mathfrak{A}$.

Corollary

If every tableau for φ false has a non-contradictory branch, there exists a structure $\mathfrak{A}$ with $\mathfrak{A} \not\models \varphi$.

Natural Deduction

Proof Calculi

Notation

$\psi_1, \ldots, \psi_n \vdash \varphi$ $\quad$ φ is **provable** with **assumptions** $\psi_1, \ldots, \psi_n$

φ is **provable** if $\vdash \varphi$.

Rules

$$\frac{\Gamma_1 \vdash \varphi_1 \ \ \ldots \ \ \Gamma_n \vdash \varphi_n}{\Delta \vdash \psi}$$

premises

conclusion

$\varphi_1 \wedge \cdots \wedge \varphi_n \Rightarrow \psi$

Axiom

$$\frac{}{\Delta \vdash \psi}$$

rule without premises

Remark

Tableaux speak about **possibilities** while Natural Deduction proofs speak about **necesseties.**

Proof Calculi

Derivation

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \varphi} \quad \dfrac{}{\Delta_0 \vdash \psi_0}}{\Delta_1 \vdash \psi_1} \qquad \dfrac{}{\Gamma' \vdash \varphi'}}{\Sigma \vdash \vartheta}$$

tree of rules

Natural Deduction (propositional part)

$$(\mathrm{I}_\top)\ \frac{}{\Gamma \vdash \top}$$

$$(\mathrm{Ax})\ \frac{}{\Gamma, \varphi \vdash \varphi}$$

$$(\mathrm{I}_\wedge)\ \frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{\Gamma, \Delta \vdash \varphi \wedge \psi}$$

$$(\mathrm{E}_\wedge)\ \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi} \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}$$

$$(\mathrm{I}_\vee)\ \frac{\Gamma, \neg\psi \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \qquad \frac{\Gamma, \neg\varphi \vdash \psi}{\Gamma \vdash \varphi \vee \psi}$$

$$(\mathrm{E}_\vee)\ \frac{\Gamma \vdash \varphi \vee \psi \quad \Delta, \varphi \vdash \vartheta \quad \Delta', \psi \vdash \vartheta}{\Gamma, \Delta, \Delta' \vdash \vartheta}$$

$$(\mathrm{I}_\neg)\ \frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}$$

$$(\mathrm{E}_\neg)\ \frac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}$$

$$(\mathrm{I}_\bot)\ \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \bot}$$

$$(\mathrm{E}_\bot)\ \frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}$$

$$(\mathrm{I}_\rightarrow)\ \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}$$

$$(\mathrm{E}_\rightarrow)\ \frac{\Gamma \vdash \varphi \quad \Delta \vdash \varphi \rightarrow \psi}{\Gamma, \Delta \vdash \psi}$$

$$(\mathrm{I}_\leftrightarrow)\ \frac{\Gamma, \varphi \vdash \psi \quad \Delta, \psi \vdash \varphi}{\Gamma, \Delta \vdash \varphi \leftrightarrow \psi}$$

$$(\mathrm{E}_\leftrightarrow)\ \frac{\Gamma \vdash \varphi \quad \Delta \vdash \varphi \leftrightarrow \psi}{\Gamma, \Delta \vdash \psi} \quad (+\ \text{sym.})$$

Examples

$$\frac{}{\vdash (\varphi \vee \psi) \to \neg(\neg\varphi \wedge \neg\psi)}$$

Examples

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \lor \psi, \neg\varphi \land \neg\psi \vdash \varphi \lor \psi} \qquad \dfrac{\dfrac{}{\varphi \vdash \varphi} \qquad \dfrac{\dfrac{}{\neg\varphi \land \neg\psi \vdash \neg\varphi \land \neg\psi}}{\neg\varphi \land \neg\psi \vdash \neg\varphi}}{\varphi, \neg\varphi \land \neg\psi \vdash \bot} \qquad \dfrac{\cdots}{\psi, \neg\varphi \land \neg\psi \vdash \bot}}{\varphi \lor \psi, \neg\varphi \land \neg\psi \vdash \bot}}{\varphi \lor \psi \vdash \neg(\neg\varphi \land \neg\psi)}}{\vdash (\varphi \lor \psi) \to \neg(\neg\varphi \land \neg\psi)}$$

Natural Deduction (quantifiers and equality)

$$(\mathrm{I}_\exists)\ \frac{\Gamma \vdash \varphi[x \mapsto t]}{\Gamma \vdash \exists x \varphi} \qquad (\mathrm{E}_\exists)\ \frac{\Gamma \vdash \exists x \varphi \quad \Delta, \varphi[x \mapsto c] \vdash \psi}{\Gamma, \Delta \vdash \psi}$$

$$(\mathrm{I}_\forall)\ \frac{\Gamma \vdash \varphi[x \mapsto c]}{\Gamma \vdash \forall x \varphi} \qquad (\mathrm{E}_\forall)\ \frac{\Gamma \vdash \forall x \varphi}{\Gamma \vdash \varphi[x \mapsto t]}$$

$$(\mathrm{I}_=)\ \frac{}{\Gamma \vdash t = t} \qquad (\mathrm{E}_=)\ \frac{\Gamma \vdash s = t \quad \Delta \vdash \varphi[x \mapsto s]}{\Gamma, \Delta \vdash \varphi[x \mapsto t]}$$

c a **new** constant symbol, s, t arbitrary terms

Examples

$s = t \vdash t = s$

$$\dfrac{\dfrac{}{s = t \vdash s = t} \qquad \dfrac{}{\vdash s = s}}{s = t \vdash t = s}\ (\mathrm{E}_=)$$

$s = t,\ t = u \vdash s = u$

$$\dfrac{\dfrac{}{t = u \vdash t = u} \qquad \dfrac{}{s = t \vdash s = t}}{s = t,\ t = u \vdash s = u}\ (\mathrm{E}_=)$$

$\exists x \forall y R(x, y) \vdash \forall y \exists x R(x, y)$

$$\dfrac{\dfrac{}{\exists x \forall y R(x, y) \vdash \exists x \forall y R(x, y)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\forall y R(c, y) \vdash \forall y R(c, y)}}{\forall y R(c, y) \vdash R(c, d)}\ (\mathrm{E}_\forall)}{\forall y R(c, y) \vdash \exists x R(x, d)}\ (\mathrm{I}_\exists)}{\forall y R(c, y) \vdash \forall y \exists x R(x, y)}\ (\mathrm{I}_\forall)}{\exists x \forall y R(x, y) \vdash \forall y \exists x R(x, y)}\ (\mathrm{E}_\exists)$$

Soundness and Completeness

Theorem

A formula φ is provable using Natural Deduction if, and only if, it is valid.

Corollary

The set of valid first-order formulae is recursively enumerable.

Isabelle/HOL

Isabelle/HOL

Proof assistant designed for software verification.

General structure

```
theory T
imports T1 ... Tn
begin
  declarations, definitions, and proofs
end
```

Syntax

Two levels:

- the **meta-language** (Isabelle) used to define theories,
- the **logical language** (HOL) used to write formulae.

To distinguish the levels, one encloses formulae of the logical language in quotes.

```
datatype 'a list = Nil                   ("[]")
                 | Cons 'a "'a list"  (infixr "#" 65)

primrec app :: "'a list => 'a list => 'a list"
                                      (infixr "@" 65)
where
"[] @ ys       = ys" |
"(x # xs) @ ys = x # (xs @ ys)"
```

Logical Language

Types

- **base types:** `bool`, `nat`, `int`,…
- **type constructors:** α `list`, α `set`,…
- **function types:** $\alpha \Rightarrow \beta$
- **type variables:** `'a`, `'b`,…

Terms

- **application:** $f\ x\ y$, $x + y$,…
- **abstraction:** $\lambda x.t$
- **type annoation:** $t :: \alpha$
- `if` b `then` t `else` u
- `let` $x = t$ `in` u
- `case` x `of` $p_0 \Rightarrow t_0 \mid \cdots \mid p_n \Rightarrow t_n$

Formulae

- terms of type `bool`
- boolean operations $\neg, \wedge, \vee, \rightarrow$
- quantifiers $\forall x$, $\exists x$
- predicates $==$, $<$,…

Basic Types

```
datatype bool = True | False

fun conj :: "bool => bool => bool" where
"conj True True = True" |
"conj _    _    = False"

datatype nat = 0 | Suc nat

fun add :: "nat => nat => nat" where
"add 0       n = n" |
"add (Suc m) n = Suc (add m n)"

lemma add_02: "add m 0 = m"
apply (induction m)
apply (auto)
done
```

Proofs

```
lemma add_02: "add m 0 = m"
```

```
apply (induction m)
```

1. `add 0 0 = 0`
2. `⋀m. add m 0 = m ==> add (Suc m) 0 = Suc m`

```
apply (auto)
```

```
datatype 'a list = Nil                    ("[]")
                 | Cons 'a "'a list"  (infixr "#" 65)

fun app :: "'a list => 'a list => 'a list"
                                      (infixr "@" 65)
where
"[] @ ys       = ys" |
"(x # xs) @ ys = x # (xs @ ys)"

fun rev :: "'a list => 'a list" where
"rev []        = []" |
"rev (x # xs)  = (rev xs) @ (x # [])"
```

```
theorem rev_rev [simp]: "rev (rev xs) = xs"
```

```
apply(induction xs)
```

1. rev (rev Nil) = Nil
2. ⋀x1 xs. rev (rev xs) = xs ==>
   rev (rev (Cons x1 xs)) = Cons x1 xs

```
apply(auto)
```

1. ⋀x1 xs.
   rev (rev xs) = xs ==>
   rev (rev xs @ Cons x1 Nil) = Cons x1 xs

```
lemma app_Nil2 [simp]: "xs @ Nil = xs"
apply(induction xs)
apply(auto)
done
```

```
lemma rev_app [simp]: "rev (xs @ ys) = rev ys @ rev xs"
apply(induction xs)
apply(auto)
```

```
1. ⋀x1 xs.
  rev (xs @ ys) = rev ys @ rev xs ==>
  (rev ys @ rev xs) @ Cons x1 Nil =
  rev ys @ (rev xs @ Cons x1 Nil)
```

```
lemma app_assoc [simp]: "(xs @ ys) @ zs = xs @ (ys @ zs)"
apply (induction xs)
apply (auto)
done
```

```
lemma app_Nil2 [simp]: "xs @ [] = xs"
apply(induction xs)
apply(auto)
done

lemma app_assoc [simp]: "(xs @ ys) @ zs = xs @ (ys @ zs)"
apply(induction xs)
apply(auto)
done

lemma rev_app [simp]: "rev(xs @ ys) = (rev ys) @ (rev xs)"
apply(induction xs)
apply(auto)
done

theorem rev_rev [simp]: "rev(rev xs) = xs"
apply(induction xs)
apply(auto)
done

end
```