

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

Semantics

$$\mathcal{A} \models \varphi(\bar{a}) \quad \mathcal{A} = \langle A, R_0, R_1, \dots, f_0, f_1, \dots \rangle$$

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

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

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)

Examples

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

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

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

Examples

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

- ‘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.’

Examples

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

- ‘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.’

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

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.’

Examples

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

- ‘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

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**

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

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

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

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: $a_0 \dots a_{n-1}$

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

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

Acceptance formula

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

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

where

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

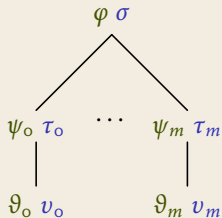
Tableaux

Tableau Proofs

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

Statements φ true or φ false

Rule



Interpretation

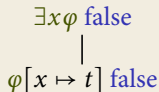
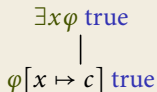
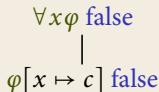
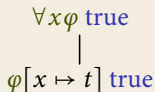
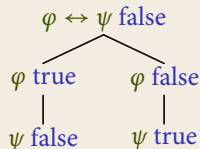
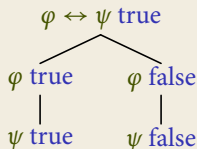
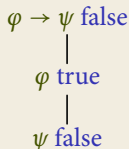
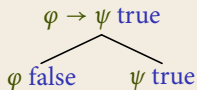
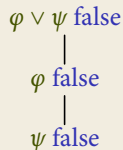
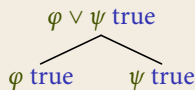
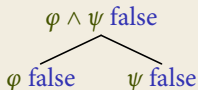
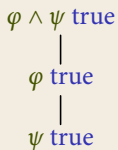
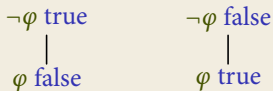
If $\varphi \sigma$ is **possible** then so is $\psi_i \tau_i, \dots, \vartheta_i v_i$, for some i .

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

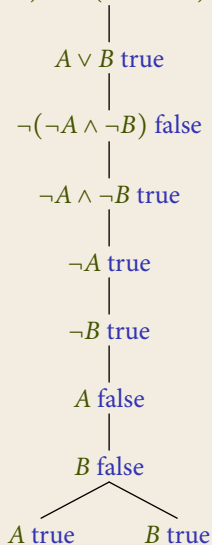
Example

$(A \vee B) \rightarrow \neg(\neg A \wedge \neg B)$ false

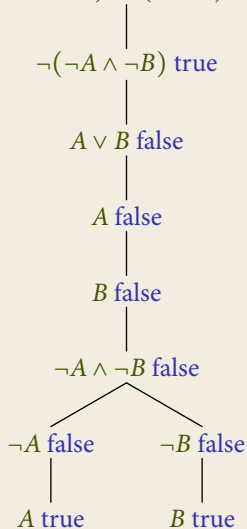
$\neg(\neg A \wedge \neg B) \rightarrow (A \vee B)$ false

Example

$(A \vee B) \rightarrow \neg(\neg A \wedge \neg B)$ false



$\neg(\neg A \wedge \neg B) \rightarrow (A \vee B)$ false



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

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

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

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

Corollary

If $\mathcal{A} \neq \varphi$, 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 $\mathfrak{A} \models \varphi$.

Natural Deduction

Proof Calculi

Notation

$\psi_1, \dots, \psi_n \vdash \varphi$ φ is **provable** with **assumptions** ψ_1, \dots, ψ_n

Proof Calculi

Notation

$\psi_1, \dots, \psi_n \vdash \varphi$ φ is **provable** with **assumptions** ψ_1, \dots, ψ_n

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

Proof Calculi

Notation

$\psi_1, \dots, \psi_n \vdash \varphi$ φ is **provable** with **assumptions** ψ_1, \dots, ψ_n

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

Rules

$$\frac{\Gamma_1 \vdash \varphi_1 \dots \Gamma_n \vdash \varphi_n}{\Delta \vdash \psi} \quad \begin{array}{l} \text{premises} \\ \text{conclusion} \end{array} \quad \varphi_1 \wedge \dots \wedge \varphi_n \Rightarrow \psi$$

Proof Calculi

Notation

$\psi_1, \dots, \psi_n \vdash \varphi$ φ is **provable** with **assumptions** ψ_1, \dots, ψ_n

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

Rules

$$\frac{\Gamma_1 \vdash \varphi_1 \dots \Gamma_n \vdash \varphi_n}{\Delta \vdash \psi} \quad \begin{array}{l} \text{premises} \\ \text{conclusion} \end{array} \quad \varphi_1 \wedge \dots \wedge \varphi_n \Rightarrow \psi$$

Axiom

$$\frac{}{\Delta \vdash \psi} \quad \text{rule without premises}$$

Proof Calculi

Notation

$\psi_1, \dots, \psi_n \vdash \varphi$ φ is **provable** with **assumptions** ψ_1, \dots, ψ_n

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

Rules

$$\frac{\Gamma_1 \vdash \varphi_1 \dots \Gamma_n \vdash \varphi_n}{\Delta \vdash \psi} \quad \begin{array}{l} \text{premises} \\ \text{conclusion} \end{array} \quad \varphi_1 \wedge \dots \wedge \varphi_n \Rightarrow \psi$$

Axiom

$$\frac{}{\Delta \vdash \psi} \quad \text{rule without premises}$$

Remark

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

Proof Calculi

Derivation

$$\frac{\frac{\overline{\Gamma \vdash \varphi} \quad \overline{\Delta_0 \vdash \psi_0}}{\Delta_1 \vdash \psi_1} \quad \overline{\Gamma' \vdash \varphi'}}{\Sigma \vdash \vartheta}$$

tree of rules

Natural Deduction (propositional part)

$$(I_{\top}) \frac{}{\Gamma \vdash \top}$$

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

$$(I_{\wedge}) \frac{\Gamma \vdash \varphi \quad \Delta \vdash \psi}{\Gamma, \Delta \vdash \varphi \wedge \psi}$$

$$(E_{\wedge}) \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi} \quad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}$$

$$(I_{\vee}) \frac{\Gamma, \neg\psi \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \quad \frac{\Gamma, \neg\varphi \vdash \psi}{\Gamma \vdash \varphi \vee \psi}$$

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

$$(I_{\neg}) \frac{\Gamma, \varphi \vdash \perp}{\Gamma \vdash \neg\varphi}$$

$$(E_{\neg}) \frac{\Gamma, \neg\varphi \vdash \perp}{\Gamma \vdash \varphi}$$

$$(I_{\perp}) \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \perp}$$

$$(E_{\perp}) \frac{\Gamma \vdash \perp}{\Gamma \vdash \varphi}$$

$$(I_{\rightarrow}) \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}$$

$$(E_{\rightarrow}) \frac{\Gamma \vdash \varphi \quad \Delta \vdash \varphi \rightarrow \psi}{\Gamma, \Delta \vdash \psi}$$

$$(I_{\leftrightarrow}) \frac{\Gamma, \varphi \vdash \psi \quad \Delta, \psi \vdash \varphi}{\Gamma, \Delta \vdash \varphi \leftrightarrow \psi}$$

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

Examples

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

Natural Deduction (quantifiers and equality)

$$(I_{\exists}) \frac{\Gamma \vdash \varphi[x \mapsto t]}{\Gamma \vdash \exists x \varphi}$$

$$(E_{\exists}) \frac{\Gamma \vdash \exists x \varphi \quad \Delta, \varphi[x \mapsto c] \vdash \psi}{\Gamma, \Delta \vdash \psi}$$

$$(I_{\forall}) \frac{\Gamma \vdash \varphi[x \mapsto c]}{\Gamma \vdash \forall x \varphi}$$

$$(E_{\forall}) \frac{\Gamma \vdash \forall x \varphi}{\Gamma \vdash \varphi[x \mapsto t]}$$

$$(I_{=}) \frac{}{\Gamma \vdash t = t}$$

$$(E_{=}) \frac{\Gamma \vdash s = t \quad \Delta \vdash \varphi[x \mapsto s]}{\Gamma, \Delta \vdash \varphi[x \mapsto t]}$$

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

Examples

$$s = t \vdash t = s$$

Examples

$$s = t \vdash t = s \qquad \frac{\frac{}{s = t \vdash s = t} \quad \frac{}{\vdash s = s}}{s = t \vdash t = s} \quad (\text{E}_=)$$

Examples

$$s = t \vdash t = s \quad \frac{\frac{}{s = t \vdash s = t} \quad \frac{}{\vdash s = s}}{s = t \vdash t = s} \quad (\text{E}_=)$$

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

Examples

$$s = t \vdash t = s \quad \frac{\frac{}{s = t \vdash s = t} \quad \frac{}{\vdash s = s}}{s = t \vdash t = s} \quad (\text{E}_=)$$

$$s = t, t = u \vdash s = u \quad \frac{\frac{}{t = u \vdash t = u} \quad \frac{}{s = t \vdash s = t}}{s = t, t = u \vdash s = u} \quad (\text{E}_=)$$

Examples

$$s = t \vdash t = s \quad \frac{\overline{s = t \vdash s = t} \quad \overline{\vdash s = s}}{s = t \vdash t = s} \quad (\text{E}_=)$$

$$s = t, t = u \vdash s = u \quad \frac{\overline{t = u \vdash t = u} \quad \overline{s = t \vdash s = t}}{s = t, t = u \vdash s = u} \quad (\text{E}_=)$$

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

Examples

$$s = t \vdash t = s \quad \frac{\frac{}{s = t \vdash s = t} \quad \frac{}{\vdash s = s}}{s = t \vdash t = s} \quad (\text{E}_=)$$

$$s = t, t = u \vdash s = u \quad \frac{\frac{}{t = u \vdash t = u} \quad \frac{}{s = t \vdash s = t}}{s = t, t = u \vdash s = u} \quad (\text{E}_=)$$

$$\begin{array}{l} \exists x \forall y R(x, y) \vdash \forall y \exists x R(x, y) \\ \frac{}{\forall y R(c, y) \vdash \forall y R(c, y)} \quad (\text{E}_\forall) \\ \frac{}{\forall y R(c, y) \vdash R(c, d)} \quad (\text{I}_\exists) \\ \frac{}{\forall y R(c, y) \vdash \exists x R(x, d)} \quad (\text{I}_\forall) \\ \frac{\frac{}{\exists x \forall y R(x, y) \vdash \exists x \forall y R(x, y)}}{\forall y R(c, y) \vdash \forall y \exists x R(x, y)} \quad (\text{E}_\exists) \\ \frac{}{\exists x \forall y R(x, y) \vdash \forall y \exists x R(x, y)} \end{array}$$

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 annotation:** $t :: \alpha$
- ▶ if b then t else u
- ▶ let $x = t$ in u
- ▶ case x of $p_0 \Rightarrow t_0 \mid \dots \mid p_n \Rightarrow t_n$

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"

Proofs

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

```
apply (induction m)
```


Proofs

lemma add_02: "add m 0 = m"

apply (induction m)

1. add 0 0 = 0

2. $\wedge m. \text{add } m \ 0 = m \implies \text{add } (\text{Suc } m) \ 0 = \text{Suc } m$

Proofs

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

```
apply (induction m)
```

```
1. add 0 0 = 0
```

```
2.  $\wedge m. \text{add } m \ 0 = m \implies \text{add } (\text{Suc } m) \ 0 = \text{Suc } m$ 
```

```
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"
```

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

```
apply(induction xs)
```

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

apply(induction xs)

1. rev (rev Nil) = Nil

2. $\wedge x1\ xs. \text{rev (rev xs) = xs} \implies$

$\text{rev (rev (Cons x1 xs)) = Cons x1 xs}$

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

```
apply(induction xs)
```

```
1. rev (rev Nil) = Nil
```

```
2.  $\wedge x1\ xs. \text{rev (rev xs) = xs} \implies$ 
```

```
   rev (rev (Cons x1 xs)) = Cons x1 xs
```

```
apply(auto)
```

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

apply(induction xs)

1. rev (rev Nil) = Nil

2. $\wedge x1$ xs. rev (rev xs) = xs ==>

rev (rev (Cons x1 xs)) = Cons x1 xs

apply(auto)

1. $\wedge x1$ 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 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. $\wedge x1$ xs.

$$\begin{aligned} \text{rev (xs @ ys)} &= \text{rev ys @ rev xs} \implies \\ (\text{rev ys @ rev xs}) @ \text{Cons } x1 \text{ Nil} &= \\ \text{rev ys @ (rev xs @ Cons } x1 \text{ Nil)} \end{aligned}$$

```
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. $\wedge x1$ xs.

$$\begin{aligned} \text{rev (xs @ ys)} &= \text{rev ys @ rev xs} \implies \\ (\text{rev ys @ rev xs}) @ \text{Cons } x1 \text{ Nil} &= \\ \text{rev ys @ (rev xs @ Cons } x1 \text{ Nil)} & \end{aligned}$$

```
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
```

Nonmonotonic Logic

Negation as Failure

Goal

Develop a proof calculus supporting Negation as Failure as used in Prolog.

Monotonicity

Ordinary deduction is **monotone**: if we add new assumption, all consequences we have already derived remain. More information does not invalidate already made deductions.

Non-Monotonicity

Negation as Failure is **non-monotone**:

P implies $\neg Q$ but P, Q does not imply $\neg Q$.

Default Logic

Rule

$$\frac{\alpha_0 \dots \alpha_m : \beta_0 \dots \beta_n}{\gamma}$$

α_i	assumptions
β_i	restraints
γ	consequence

Derive γ provided that we can derive $\alpha_0, \dots, \alpha_m$, but none of β_0, \dots, β_n .

Example

$$\frac{\text{bird}(x) : \text{penguin}(x) \text{ ostrich}(x)}{\text{can_fly}(x)}$$

Semantics

Definition

A set Φ of formulae is **consistent** with respect to a set of rules R if, for every rule

$$\frac{\alpha_0 \dots \alpha_m : \beta_0 \dots \beta_n}{\gamma} \in R$$

such that $\alpha_0, \dots, \alpha_m \in \Phi$ and $\beta_0, \dots, \beta_n \notin \Phi$, we have $\gamma \in \Phi$.

Note

If there are no restraints β_i , consistent sets are **closed under intersection**.

\Rightarrow There is a unique smallest such set, that of all **provable** formulae.

If there are restraints, this may not be the case. Formulae that belong to all consistent sets are called **secured consequences**.

Examples

The system

$$\frac{\quad}{\alpha} \quad \frac{\alpha : \beta}{\beta}$$

has a unique consistent set $\{\alpha, \beta\}$.

The system

$$\frac{\quad}{\alpha} \quad \frac{\alpha : \beta}{\gamma} \quad \frac{\alpha : \gamma}{\beta}$$

has consistent sets

$$\{\alpha, \beta\}, \quad \{\alpha, \gamma\}, \quad \{\alpha, \beta, \gamma\}.$$