IA008: Computational Logic

4. Deduction

Achim Blumensath blumens@fi.muni.cz

Faculty of Informatics, Masaryk University, Brno

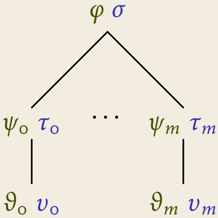
Tableaux

Tableau Proofs

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

Statements φ true or φ false

Rule



Interpretation

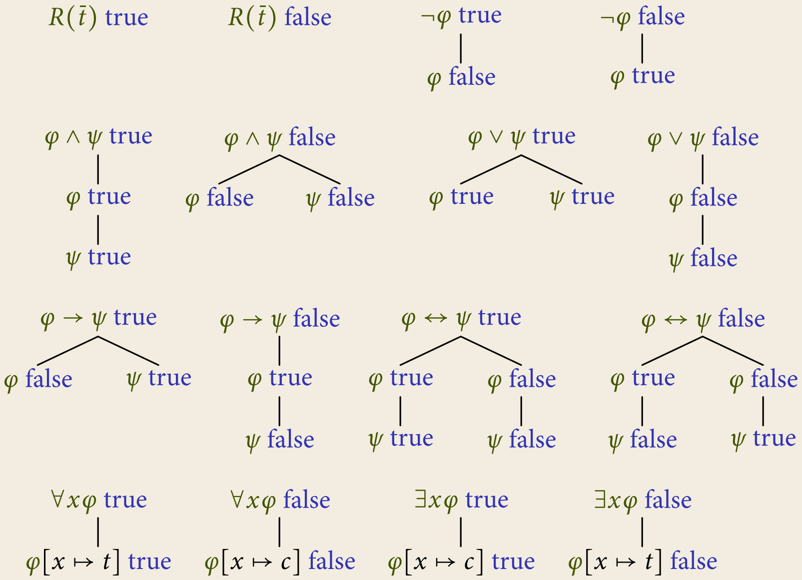
If $\varphi\ \sigma$ is **possible** then so is $\psi_i\ \tau_i, \dots, \vartheta_i\ \upsilon_i$, for some i.

Tableaux

Construction

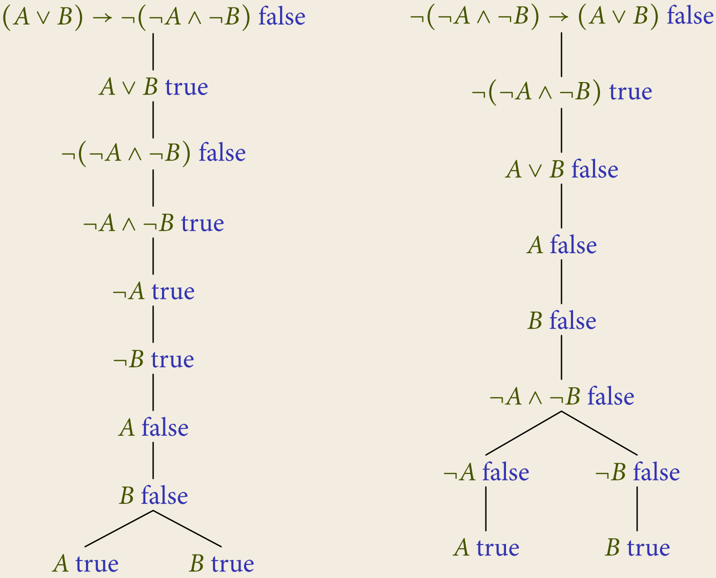
A **tableau** for a formula φ is constructed as follows:

- 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 ∨ 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

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

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** $\psi_1, \ldots, \psi_n$

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 $\vdash \varphi$.

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

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

Proof Calculi

Notation

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

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

Rules

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

Axiom

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

Proof Calculi

Notation

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

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

Rules

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

Axiom

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

Remark

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

Proof Calculi

Derivation

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

tree of rules

Natural Deduction (propositional part)

$$(\mathrm{I}_\top)\ \frac{}{\Gamma \vdash \top} \qquad (\mathrm{Ax})\ \frac{}{\Gamma, \varphi \vdash \varphi}$$

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

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

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

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

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

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

Examples

$$
\dfrac{\dfrac{\dfrac{}{\varphi \vee \psi, \neg\varphi \wedge \neg\psi \vdash \varphi \vee \psi} \quad \dfrac{\dfrac{}{\varphi \vdash \varphi} \quad \dfrac{\dfrac{}{\neg\varphi \wedge \neg\psi \vdash \neg\varphi \wedge \neg\psi}}{\neg\varphi \wedge \neg\psi \vdash \neg\varphi}}{\varphi, \neg\varphi \wedge \neg\psi \vdash \bot} \quad \dfrac{\cdots}{\psi, \neg\varphi \wedge \neg\psi \vdash \bot}}{\dfrac{\varphi \vee \psi, \neg\varphi \wedge \neg\psi \vdash \bot}{\varphi \vee \psi \vdash \neg(\neg\varphi \wedge \neg\psi)}}}{\vdash (\varphi \vee \psi) \to \neg(\neg\varphi \wedge \neg\psi)}
$$

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

$s = t \vdash t = s$

Examples

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

Examples

$s = t \vdash t = s$

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

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

Examples

$s = t \vdash t = s$

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

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

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

Examples

$s = t \vdash t = s$

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

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

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

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

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:** $f\ x\ y,\ x + y, \ldots$
- **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 $==, <, \ldots$

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)
```

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

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"
```

```
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. `⋀x1 xs. rev (rev xs) = xs ==>`
   `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. ⋀x1 xs. rev (rev xs) = xs ==>
   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. ⋀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 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_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)
```

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:**

$$P \text{ implies } \neg Q \quad \text{but} \quad P, Q \text{ does not imply } \neg Q\,.$$

Default Logic

Rule

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

α_i	assumptions
β_i	restraints
γ	consequence

Derive γ provided that we can derive $\alpha_0, \ldots, \alpha_m$, but none of $\beta_0, \ldots, \beta_n$.

Example

$$\frac{\mathrm{bird}(x) : \mathrm{penguin}(x) \ \mathrm{ostrich}(x)}{\mathrm{can\_fly}(x)}$$

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

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

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

The system

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

has consistent sets

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