First-Order Theorem Proving

Peter Baumgartner

NICTA, Logic and Computation Program, Canberra

`Peter.Baumgartner@nicta.com.au`

Slides partially based on material by Uli Furbach, Harald Ganzinger, John Slaney, Viorica Sofronie-Stockermans and Uwe Waldmann

Contents

Part I – What First-Order Theorem Proving is About

- Mission statement
- A glimpse at First-Order Theorem Proving

Mission Statement

Theorem proving is about . . .

Logics: Propositional, First-Order, Higher-Order, Modal, Description, . . .

Calculi and proof procedures: Resolution, DPLL, Tableaux, . . .

Systems: Interactive, Automated

Applications: Knowledge Representation, Verification, . . .

Mission Statement

Theorem proving is about . . .

Logics: Propositional, First-Order, Higher-Order, Modal, Description, . . .

Calculi and proof procedures: Resolution, DPLL, Tableaux, . . .

Systems: Interactive, Automated

Applications: Knowledge Representation, Verification, . . .

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Logic Based Knowledge Representation

90s: Modern Theory and Implementations, “A Basis for Applications”

2000s: Ontological Engineering, Verification

Mission Statement

In this talk, theorem proving is about . . .

Logics: Propositional, **First-Order**, Higher-Order, Modal, Description, . . .

Calculi and proof procedures: **Resolution**, DPLL, Tableaux, . . .

Systems: Interactive, **Automated**

Applications: Knowledge Representation, Verification, . . .

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Logic Based Knowledge Representation

90s: Modern Theory and Implementations, "A Basis for Applications"

2000s: Ontological Engineering, Verification

Application: Compiler Validation

Problem: prove equivalence of source and target program

Example:

```
1:   y := 1                   1: y := 1
2:   if z = x*x*x             2: R1 := x*x
3:     then y := x*x + y      3: R2 := R1*x
4:   endif                    4: jmpNE(z,R2,6)
                              5: y := R1+1
```

To prove: (indexes refer to values at line numbers; index 0 = initial values)

$$y_1 \approx 1 \wedge z_0 \approx x_0 * x_0 * x_0 \wedge y_3 \approx x_0 * x_0 + y_1$$

$$y'_1 \approx 1 \wedge R1_2 \approx x'_0 * x'_0 \wedge R2_3 \approx R1_2 * x'_0 \wedge z'_0 \approx R2_3 \wedge y'_5 \approx R1_2 + 1$$

$$\wedge\, x_0 \approx x'_0 \wedge y_0 \approx y'_0 \wedge z_0 \approx z'_0 \quad \models \quad y_3 \approx y'_5$$

A Glimpse at FOTP

A logical puzzle:

Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler.

A Glimpse at FOTP

A logical puzzle:

> Someone who lives in Dreadbury Mansion killed Aunt Agatha. Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein. A killer always hates his victim, and is never richer than his victim. Charles hates no one that Aunt Agatha hates. Agatha hates everyone except the butler. The butler hates everyone not richer than Aunt Agatha. The butler hates everyone Aunt Agatha hates. No one hates everyone. Agatha is not the butler.

Who killed Aunt Agatha?

A Glimpse at FOTP

Before solving the problem with a theorem prover we have to formalize it:

A Glimpse at FOTP

Before solving the problem with a theorem prover we have to formalize it:

Someone who lives in Dreadbury Mansion killed Aunt Agatha.

A Glimpse at FOTP

Before solving the problem with a theorem prover we have to formalize it:

Someone who lives in Dreadbury Mansion killed Aunt Agatha.

- $\exists x\ (\mathrm{lives\_at\_dreadbury}(x) \land \mathrm{killed}(x, \mathrm{a}))$

A Glimpse at FOTP

Before solving the problem with a theorem prover we have to formalize it:

Someone who lives in Dreadbury Mansion killed Aunt Agatha.

- $\exists x\ (\mathrm{lives\_at\_dreadbury}(x) \land \mathrm{killed}(x, \mathrm{a}))$

Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein.

A Glimpse at FOTP

Before solving the problem with a theorem prover we have to formalize it:

Someone who lives in Dreadbury Mansion killed Aunt Agatha.

- $\exists x\ (\text{lives\_at\_dreadbury}(x) \land \text{killed}(x, \mathrm{a}))$

Agatha, the butler, and Charles live in Dreadbury Mansion, and are the only people who live therein.

- $\forall x\ (\text{lives\_at\_dreadbury}(x) \leftrightarrow (x = \mathrm{a} \lor x = \mathrm{b} \lor x = \mathrm{c}))$

A Glimpse at FOTP

A killer always hates his victim, and is never richer than his victim.

- $\forall x, y\ (\mathrm{killed}(x, y) \rightarrow \mathrm{hates}(x, y))$
  $\forall x, y\ (\mathrm{killed}(x, y) \rightarrow \neg\,\mathrm{richer}(x, y))$

Charles hates no one that Aunt Agatha hates.

- $\forall x\ (\mathrm{hates}(\mathrm{c}, x) \rightarrow \neg\,\mathrm{hates}(\mathrm{a}, x))$

Agatha hates everyone except the butler.

- $\forall x\ (\neg\,\mathrm{hates}(\mathrm{a}, x) \leftrightarrow x = \mathrm{b})$

A Glimpse at FOTP

The butler hates everyone not richer than Aunt Agatha.

- $\forall x\ (\neg\ \mathrm{richer}(x, \mathrm{a}) \rightarrow \mathrm{hates}(\mathrm{b}, x))$

The butler hates everyone Aunt Agatha hates.

- $\forall x\ (\mathrm{hates}(\mathrm{a}, x) \rightarrow \mathrm{hates}(\mathrm{b}, x))$

No one hates everyone.

- $\forall x\ \exists y\ (\neg\ \mathrm{hates}(x, y))$

Agatha is not the butler.

- $\neg\ \mathrm{a} = \mathrm{b}$

A Glimpse at FOTP

Now we can derive new formulas from the given ones.

For instance:

A Glimpse at FOTP

Now we can derive new formulas from the given ones.

For instance:

$$\frac{\mathrm{killed}(x, y) \rightarrow \mathrm{hates}(x, y) \qquad \mathrm{hates}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y)}{\mathrm{killed}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y)}$$

A Glimpse at FOTP

Now we can derive new formulas from the given ones.

For instance:

$$\frac{\mathrm{killed}(x, y) \rightarrow \mathrm{hates}(x, y) \qquad \mathrm{hates}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y)}{\mathrm{killed}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y)}$$

$$\frac{\mathrm{killed}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y) \qquad \neg\, \mathrm{hates}(\mathrm{a}, y) \rightarrow y = \mathrm{b}}{\mathrm{killed}(\mathrm{c}, y) \rightarrow y = \mathrm{b}}$$

A Glimpse at FOTP

Now we can derive new formulas from the given ones.

For instance:

$$\frac{\mathrm{killed}(x, y) \rightarrow \mathrm{hates}(x, y) \qquad \mathrm{hates}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y)}{\mathrm{killed}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y)}$$

$$\frac{\mathrm{killed}(\mathrm{c}, y) \rightarrow \neg\, \mathrm{hates}(\mathrm{a}, y) \qquad \neg\, \mathrm{hates}(\mathrm{a}, y) \rightarrow y = \mathrm{b}}{\mathrm{killed}(\mathrm{c}, y) \rightarrow y = \mathrm{b}}$$

$$\frac{\mathrm{killed}(\mathrm{c}, y) \rightarrow y = \mathrm{b} \qquad \neg\, \mathrm{a} = \mathrm{b}}{\neg\, \mathrm{killed}(\mathrm{c}, \mathrm{a})}$$

A Glimpse at FOTP

By the previous reasoning we know that Charles is not the murderer.
But the further reasoning is quite tedious.

Fortunately we can use a theorem prover!

A Glimpse at FOTP

By the previous reasoning we know that Charles is not the murderer.
But the further reasoning is quite tedious.

Fortunately we can use a theorem prover!

Demo

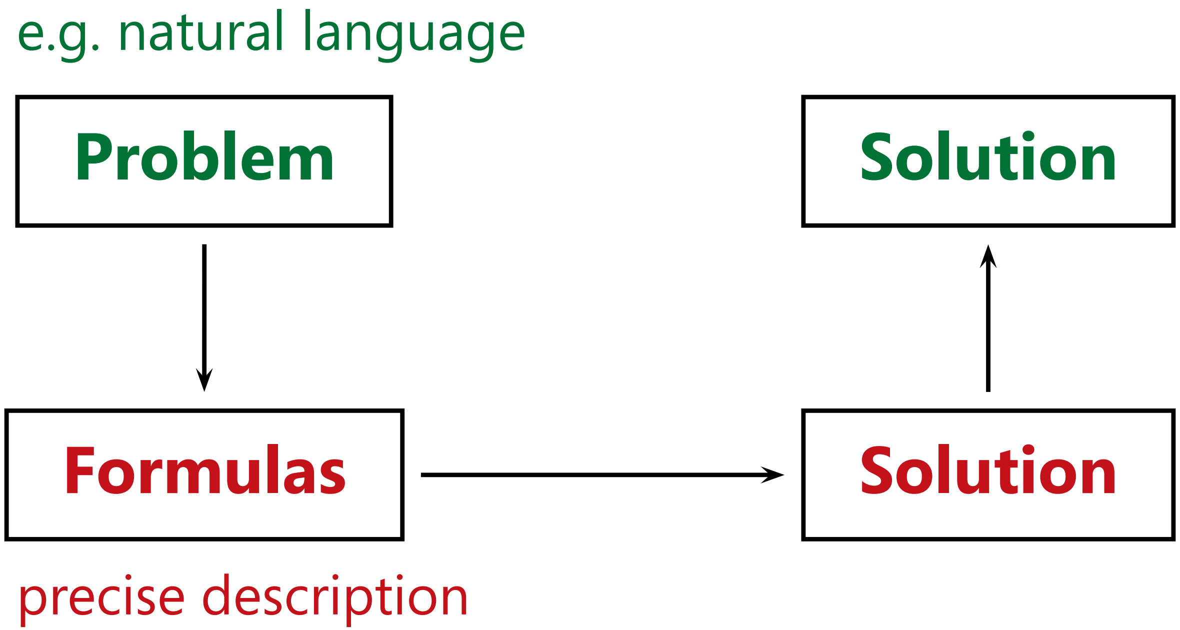
- Theorem prover: Otter
  `http://www-unix.mcs.anl.gov/AR/otter/`
- Aunt Agatha puzzle: PUZ001-2 in the TPTP
  TPTP = Thousands of Problems for Theorem Provers
  `http://www.cs.miami.edu/~tptp/`

The Principle

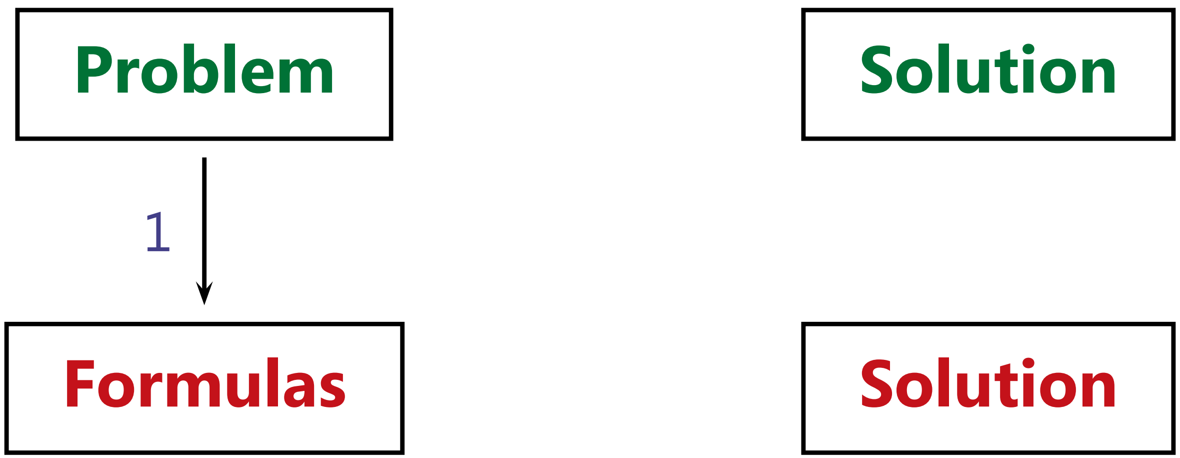
Problem - - - - - - - - - → **Solution**

Problem	Solution
Description of the situation in Dreadbury Mansion	Charles killed Agatha
	The butler killed Agatha
	Agatha committed suicide
Who killed Agatha?	Murderer unknown

The Principle



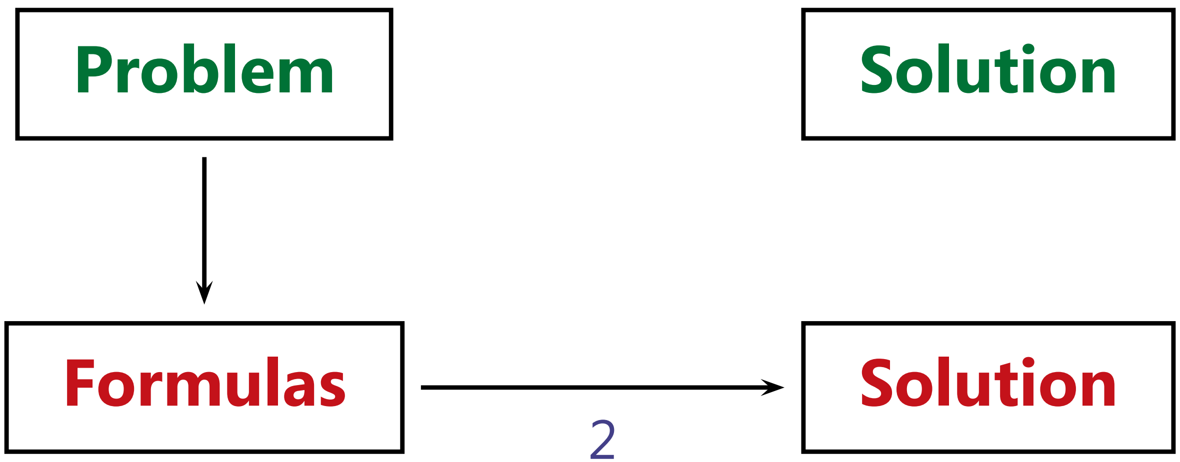
The Principle



1. Formalization: from problems to formulas

   Can sometimes be done automatically

The Principle



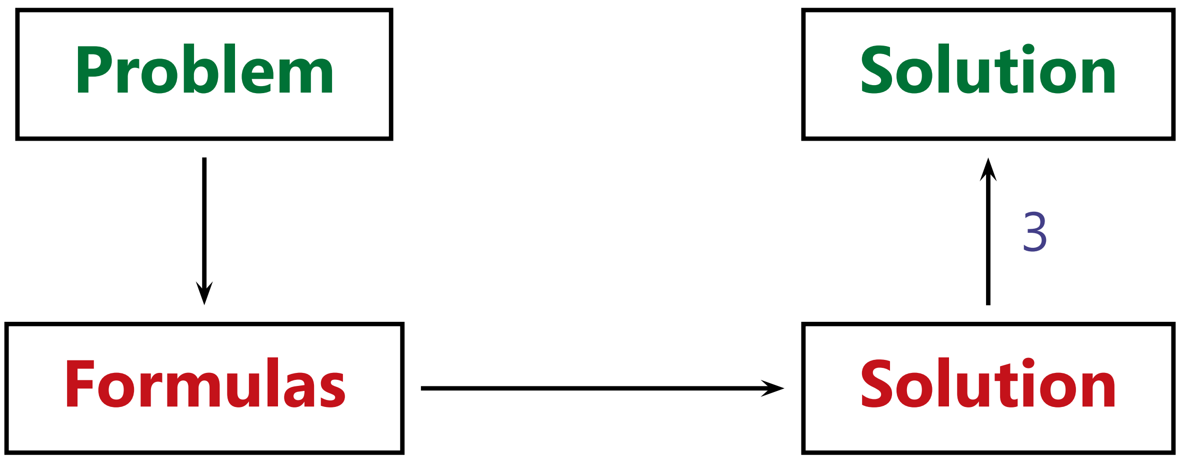
2. Solve the formalized problem

    In practice usually **very** many new formulas will be generated

    Computer support is necessary

    (even then the sheer number of formulas is the main problem)

The Principle



3. Translate back solution

    Can sometimes be done automatically

    Not always trivial!

Non-Theorems

So far, the problems had the following shape:

Does a formula (e.g.: killed(a, a)) follow from other formulas?

Non-Theorems

So far, the problems had the following shape:

Does a formula (e.g.: $\mathsf{killed}(a, a)$) follow from other formulas?

Problems of the following, complementary kind are interesting, too:

Does a formula (e.g.: $\mathsf{killed}(b, a)$) **not** follow from other formulas?

Non-Theorems

So far, the problems had the following shape:

Does a formula (e.g.: killed(a, a)) follow from other formulas?

Problems of the following, complementary kind are interesting, too:

Does a formula (e.g.: killed(b, a)) **not** follow from other formulas?

Non-entailment is much harder a problem!

Part II – First-Order Predicate Logic (from the viewpoint of ATP)

- A mathematical example
- Syntax and semantics of first-order predicate logic
- Normal forms

A Mathematical Example

The sum of two continuous function is continuous.

Definition $f : \mathbb{R} \to \mathbb{R}$ is **continuous** at a, if for every $\varepsilon > 0$ there is a $\delta > 0$, such that for all x with $|x - a| < \delta$ it holds $|f(x) - f(a)| < \varepsilon$.

Proposition If f and g are continuous, so is their sum.

Proof Let $h = f + g$ assume $\varepsilon > 0$ given. With f and g continuous, there are δ_f and δ_g greater than 0 such that, if $|x - a| < \delta_f$, then $|f(x) - f(a)| < \varepsilon/2$ and, if $|x - a| < \delta_g$, then $|g(x) - g(a)| < \varepsilon/2$. Chose $\delta = \min(\delta_f, \delta_g)$. If $|x - a| < \delta$ then we approximate:

$$\begin{aligned}
|h(x) - h(a)| &= |f(x) + g(x) - f(a) - g(a)| \\
&= |(f(x) - f(a)) + (g(x) - g(a))| \\
&\leq |f(x) - f(a)| + |g(x) - g(a)| < \varepsilon/2 + \varepsilon/2 = \varepsilon
\end{aligned}$$

The Language of Predicate Logic

"f ist continuous", expressed in first-order predicate logic:

$$\forall \varepsilon (0 < \varepsilon \rightarrow \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

in ASCII:

```
all(Eps,
  0<Eps =>
    all(A,
      exists(Delta,
        0<Delta and
         all(X, abs(X-A)<Delta =>
                abs(f(X)-f(A)) < Eps)))))
```

Can pass this formula to a theorem prover?

What does it "mean" to the prover?

Predicate Logic Syntax

$$\forall \varepsilon (0 < \varepsilon \rightarrow \forall a \exists \delta (0 < \delta \wedge \forall x (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon)))$$

Variables ε, a, δ, x

Function symbols 0, $|\_|$, $\_ - \_$, $f(\_)$

Terms are well-formed expressions over variables and function symbols

Predicate symbols $\_ < \_$, $\_ = \_$

Atoms are applications of predicate symbols to terms

Boolean connectives $\wedge$, $\vee$, $\rightarrow$, $\neg$

Quantifiers $\forall$, $\exists$

The function symbols and predicate symbols, each of given arity, comprise a signature Σ.

A **ground term** is a term without any variables.

Predicate Logic Semantics

Universe (aka Domain): Set U

Variables $\mapsto$ values in U (mapping is called "assignment")

Function symbols $\mapsto$ (total) functions over U

Predicate symbols $\mapsto$ relations over U

Boolean conectives $\mapsto$ the usual boolean functions

Quantifiers $\mapsto$ "for all ... holds", "there is a ..., such that"

Terms $\mapsto$ values in U

Formulas $\mapsto$ Boolean (Truth-) values

The underlying mathematical concept is that of a Σ-algebra.

Example

Let Σ_{PA} be the standard signature of Peano Arithmetic.
The standard interpretation for Peano Arithmetic then is:

$$
\begin{array}{rcl}
U_{\mathbb{N}} & = & \{0, 1, 2, \ldots\} \\
0_{\mathbb{N}} & = & 0 \\
s_{\mathbb{N}} & : & n \mapsto n + 1 \\
+_{\mathbb{N}} & : & (n, m) \mapsto n + m \\
*_{\mathbb{N}} & : & (n, m) \mapsto n * m \\
\leq_{\mathbb{N}} & = & \{(n, m) \mid n \text{ less than or equal to } m\} \\
<_{\mathbb{N}} & = & \{(n, m) \mid n \text{ less than } m\}
\end{array}
$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Example

Values over $\mathbb{N}$ for sample terms and formulas:

Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$\begin{array}{rcl}
\mathbb{N}(\beta)(s(x) + s(0)) & = & 3 \\
\mathbb{N}(\beta)(x + y \approx s(y)) & = & 1 \\
\mathbb{N}(\beta)(\forall x, y (x + y \approx y + x)) & = & \mathit{True} \\
\mathbb{N}(\beta)(\forall z\ z \leq y) & = & \mathit{False} \\
\mathbb{N}(\beta)(\forall x \exists y\ x < y) & = & \mathit{True}
\end{array}$$

If ϕ is a closed formula, then, instead of $I(\phi) = \mathit{True}$ one writes $I \models \phi$ ("I is a model of ϕ").

E.g. $\mathbb{N} \models \forall x \exists y\ x < y$

Axiomatizing the Real Numbers

In our proof problem, we have to "axiomatize" all those properties of the standard functions and predicate symbols that are needed to get a proof.

There are only some of them here.

Addition and Subtraction:

$$
\begin{aligned}
x + y &= y + x \\
(x + y) + z &= x + (y + z) \\
x - y &= x + (-y) \\
-(x + y) &= (-x) + (-y)
\end{aligned}
$$

Axiomatizing the Real Numbers

Ordering:

$$
\begin{array}{rcl}
 & & \neg\, x < x \\
x < y \;\wedge\; y < z & \rightarrow & x < z \\
 & & x \leq x \\
x \leq y & \leftrightarrow & x < y \;\vee\; x = y \\
 & & x \leq y \;\vee\; y < x
\end{array}
$$

divide by 2 and absolute values:

$$
\begin{array}{rcl}
x/2 \leq 0 & \rightarrow & x \leq 0 \\
x < z/2 \;\wedge\; y < z/2 & \rightarrow & x + y < z \\
 & & |x + y| \leq |x| + |y|
\end{array}
$$

Now one can prove:

$$\text{Axioms over } \mathbb{R} \land \text{continuous}(f) \land \text{continuous}(g) \quad \models \quad \text{continuous}(f+g)$$

Now one can prove:

$$\text{Axioms over } \mathbb{R} \land \text{continuous}(f) \land \text{continuous}(g) \quad \models \quad \text{continuous}(f+g)$$

It can even be proven fully automatically!

Algorithmic Problems

The following is a list of practically relevant problems:

Validity(F): $\models F$? (is F true in every interpretation?)

Satisfiability(F): F satisfiable?

Entailment(F,G): $F \models G$? (does F entail G?),

Model(A,F): $A \models F$?

Solve(A,F): find an assignment β such that $A, \beta \models F$

Solve(F): find a substitution σ such that $\models F\sigma$

Abduce(F): find G with "certain properties" such that G entails F

Different problems may require rather different methods! But ...

Refutational Theorem Proving

- Suppose we want to prove $H \models G$.

Refutational Theorem Proving

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \rightarrow G$ is valid.

Refutational Theorem Proving

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \rightarrow G$ is valid.
- Equivalently, we can prove that $\neg F$, i.e.$H \wedge \neg G$ is unsatisfiable.

Refutational Theorem Proving

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \rightarrow G$ is valid.
- Equivalently, we can prove that $\neg F$, i.e.$H \land \neg G$ is unsatisfiable.

This principle of "refutational theorem proving" is the basis of almost all automated theorem proving methods.

Normal Forms

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving.

The main problem in first-order logic is the treatment of quantifiers. The subsequent normal form transformations are intended to eliminate many of them.

Prenex Normal Form

Prenex formulas have the form

$$Q_1 x_1 \ldots Q_n x_n \; F,$$

where F is quantifier-free and $Q_i \in \{\forall, \exists\}$;
we call $Q_1 x_1 \ldots Q_n x_n$ the **quantifier prefix** and F the **matrix** of the formula.

Prenex Normal Form

Computing prenex normal form by the rewrite relation $\Rightarrow_P$:

$$
\begin{array}{rcll}
(F \leftrightarrow G) & \Rightarrow_P & (F \rightarrow G) \wedge (G \rightarrow F) & \\
\neg QxF & \Rightarrow_P & \overline{Q}x\neg F & (\neg Q) \\
(QxF \; \rho \; G) & \Rightarrow_P & Qy(F[y/x] \; \rho \; G), \; y \text{ fresh}, \; \rho \in \{\wedge, \vee\} & \\
(QxF \rightarrow G) & \Rightarrow_P & \overline{Q}y(F[y/x] \rightarrow G), \; y \text{ fresh} & \\
(F \; \rho \; QxG) & \Rightarrow_P & Qy(F \; \rho \; G[y/x]), \; y \text{ fresh}, \; \rho \in \{\wedge, \vee, \rightarrow\} &
\end{array}
$$

Here $\overline{Q}$ denotes the quantifier **dual** to Q, i.e., $\overline{\forall} = \exists$ and $\overline{\exists} = \forall$.

In the Example

$$\forall\varepsilon(0<\varepsilon\rightarrow\forall a\exists\delta(0<\delta\wedge\forall x(|x-a|<\delta\rightarrow|f(x)-f(a)|<\varepsilon)))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a(0<\varepsilon\rightarrow\exists\delta(0<\delta\wedge\forall x(|x-a|<\delta\rightarrow|f(x)-f(a)|<\varepsilon)))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a\exists\delta(0<\varepsilon\rightarrow 0<\delta\wedge\forall x(|x-a|<\delta\rightarrow|f(x)-f(a)|<\varepsilon))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a\exists\delta(0<\varepsilon\rightarrow\forall x(0<\delta\wedge|x-a|<\delta\rightarrow|f(x)-f(a)|<\varepsilon))$$

$$\Rightarrow_P$$

$$\forall\varepsilon\forall a\exists\delta\forall x(0<\varepsilon\rightarrow(0<\delta\wedge(|x-a|<\delta\rightarrow|f(x)-f(a)|<\varepsilon)))$$

Skolemization

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.
Transformation $\Rightarrow_S$ (to be applied outermost, **not** in subformulas):

$$\forall x_1, \ldots, x_n \exists y F \quad \Rightarrow_S \quad \forall x_1, \ldots, x_n F[f(x_1, \ldots, x_n)/y]$$

where f/n is a new function symbol (**Skolem function**).

Skolemization

Intuition: replacement of $\exists y$ by a concrete choice function computing y from all the arguments y depends on.
Transformation $\Rightarrow_S$ (to be applied outermost, **not** in subformulas):

$$\forall x_1, \ldots, x_n \exists y F \quad \Rightarrow_S \quad \forall x_1, \ldots, x_n F[f(x_1, \ldots, x_n)/y]$$

where f/n is a new function symbol (**Skolem function**).

In the Example

$$\forall \varepsilon \forall a \exists \delta \forall x (0 < \varepsilon \rightarrow 0 < \delta \land (|x - a| < \delta \rightarrow |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_S$$

$$\forall \varepsilon \forall a \forall x (0 < \varepsilon \rightarrow 0 < d(\varepsilon, a) \land (|x - a| < d(\varepsilon, a) \rightarrow |f(x) - f(a)| < \varepsilon))$$

Skolemization

Together: $F \overset{*}{\Rightarrow}_P \underbrace{G}_{\text{prenex}} \overset{*}{\Rightarrow}_S \underbrace{H}_{\text{prenex, no } \exists}$

Theorem: The given and the final formula are equi-satisfiable.

Clausal Normal Form (Conjunctive Normal Form)

$$
\begin{array}{rcl}
(F \leftrightarrow G) & \Rightarrow_K & (F \rightarrow G) \land (G \rightarrow F) \\
(F \rightarrow G) & \Rightarrow_K & (\neg F \lor G) \\
\neg(F \lor G) & \Rightarrow_K & (\neg F \land \neg G) \\
\neg(F \land G) & \Rightarrow_K & (\neg F \lor \neg G) \\
\neg\neg F & \Rightarrow_K & F \\
(F \land G) \lor H & \Rightarrow_K & (F \lor H) \land (G \lor H) \\
(F \land \top) & \Rightarrow_K & F \\
(F \land \bot) & \Rightarrow_K & \bot \\
(F \lor \top) & \Rightarrow_K & \top \\
(F \lor \bot) & \Rightarrow_K & F
\end{array}
$$

These rules are to be applied modulo associativity and commutativity of $\land$ and $\lor$.

The first five rules, plus the rule $(\neg Q)$, compute the **negation normal form** (NNF) of a formula.

In the Example

$$\forall \varepsilon \forall a \forall x (0 < \varepsilon \rightarrow 0 < d(\varepsilon, a) \wedge (|x - a| < d(\varepsilon, a) \rightarrow |f(x) - f(a)| < \varepsilon))$$

$$\Rightarrow_K$$

$$0 < d(\varepsilon, a) \ \vee \ \neg (0 < \varepsilon)$$
$$\neg (|x - a| < d(\varepsilon, a)) \ \vee \ |f(x) - f(a)| < \varepsilon \ \vee \ \neg (0 < \varepsilon)$$

Note: The universal quantifiers for the variables ε, a and x, as well as the conjunction symbol $\wedge$ between the clauses are not written, for convenience.

The Complete Picture

$$
\begin{array}{llll}
F & \overset{*}{\Rightarrow}_P & Q_1 y_1 \ldots Q_n y_n \; G & \text{(G quantifier-free)} \\[1em]
 & \overset{*}{\Rightarrow}_S & \forall x_1, \ldots, x_m \; H & \text{($m \leq n$, H quantifier-free)} \\[1em]
 & \overset{*}{\Rightarrow}_K & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \; \bigwedge_{i=1}^{k} \; \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'} &
\end{array}
$$

$N = \{C_1, \ldots, C_k\}$ is called the **clausal (normal) form** (CNF) of F.

Note: the variables in the clauses are implicitly universally quantified.

The Complete Picture

$$\begin{array}{llll} F & \Rightarrow^*_P & Q_1 y_1 \ldots Q_n y_n \; G & (G \text{ quantifier-free}) \\ & \Rightarrow^*_S & \forall x_1, \ldots, x_m \; H & (m \leq n,\, H \text{ quantifier-free}) \\ & \Rightarrow^*_K & \underbrace{\underbrace{\forall x_1, \ldots, x_m}_{\text{leave out}} \bigwedge_{i=1}^{k} \underbrace{\bigvee_{j=1}^{n_i} L_{ij}}_{\text{clauses } C_i}}_{F'} & \end{array}$$

$N = \{C_1, \ldots, C_k\}$ is called the **clausal (normal) form** (CNF) of F.

Note: the variables in the clauses are implicitly universally quantified.

Now we arrived at "low-level predicate logic" and the proof problem, proper, i.e. to prove that the clause set is unsatisfiable.

Propositional Clause Logic

A particular syntactically simple, yet practically most significant case.

Propositional clause logic = clause logic without variables

Propositional clause: a disjunction of literals, e.g. $A \vee B \vee \neg C \vee \neg \mathrm{D}$

Propositional clause set: a (finite) set of propositional clauses.

Propositional Clause Logic

A particular syntactically simple, yet practically most significant case.
Propositional clause logic = clause logic without variables

Propositional clause: a disjunction of literals, e.g. $A \vee B \vee \neg C \vee \neg \mathrm{D}$

Propositional clause set: a (finite) set of propositional clauses.

Interpretation: maps atoms to $\{true, false\}$, e.g.

A	B	C	D
true	*false*	*true*	*false*

Represented as the set of its true atoms, e.g. $\{A, C\}$

Propositional Clause Logic

A particular syntactically simple, yet practically most significant case.
Propositional clause logic = clause logic without variables

Propositional clause: a disjunction of literals, e.g. $A \vee B \vee \neg C \vee \neg D$

Propositional clause set: a (finite) set of propositional clauses.

Interpretation: maps atoms to $\{true, false\}$, e.g.

A	B	C	D
true	*false*	*true*	*false*

Represented as the set of its true atoms, e.g. $\{A, C\}$

We don't specialize on methods for propositional logic here.
See lecture by Toby Walsh.

Herbrand Theory

Some thoughts

- Suppose we want to prove $H \models G$.

Herbrand Theory

Some thoughts

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \land \neg G$ is unsatisfiable.

Herbrand Theory

Some thoughts

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \land \neg G$ is unsatisfiable.
- We have seen how F can be syntactically simplified to clause form F' in a satisfiability preserving way.

Herbrand Theory

Some thoughts

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \land \neg G$ is unsatisfiable.
- We have seen how F can be syntactically simplified to clause form F' in a satisfiability preserving way.
- It remains to prove that F' is unsatisfiable.

Herbrand Theory

Some thoughts

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \land \neg G$ is unsatisfiable.
- We have seen how F can be syntactically simplified to clause form F' in a satisfiability preserving way.
- It remains to prove that F' is unsatisfiable.
- Does this mean that "**all** interpretations have to be searched"?

Herbrand Theory

Some thoughts

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \land \neg G$ is unsatisfiable.
- We have seen how F can be syntactically simplified to clause form F' in a satisfiability preserving way.
- It remains to prove that F' is unsatisfiable.
- Does this mean that "**all** interpretations have to be searched"?
  No! It suffices to "search only through Herbrand interpretations"

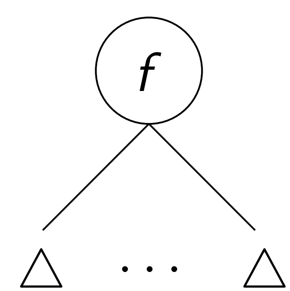
Herbrand Theory

Significance: semantical basis for most theorem proving systems

A **Herbrand interpretation** (over a given signature Σ) is a Σ-algebra $\mathcal{A}$ such that

- $U_{\mathcal{A}} = \mathrm{T}_{\Sigma}$ (= the set of ground terms over Σ)
- $f_{\mathcal{A}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n), \ f/n \in \Omega$

$$f_{\mathcal{A}}(\triangle, \ldots, \triangle) =$$



Herbrand Interpretations

In other words, **values are fixed** to be ground terms and **functions are fixed** to be the **term constructors**. Only predicate symbols $p/m \in \Pi$ may be freely interpreted as relations $p_{\mathcal{A}} \subseteq \mathsf{T}_{\Sigma}^{m}$.

Proposition
Every set of ground atoms I uniquely determines a Herbrand interpretation $\mathcal{A}$ via

$$(s_1, \ldots, s_n) \in p_{\mathcal{A}} \quad :\Leftrightarrow \quad p(s_1, \ldots, s_n) \in I$$

Thus we shall identify Herbrand interpretations (over Σ) with sets of Σ-ground atoms.

Herbrand Interpretations

Example: $\Sigma_{Pres} = (\{0/0, s/1, +/2\},\ \{</2, \leq/2\})$

$\mathbb{N}$ as Herbrand interpretation over Σ_{Pres}:

$$
\begin{aligned}
I = \{ \quad & 0 \leq 0,\ 0 \leq s(0),\ 0 \leq s(s(0)),\ \ldots, \\
& 0 + 0 \leq 0,\ 0 + 0 \leq s(0),\ \ldots, \\
& \ldots,\ (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0)) \\
& \ldots \\
& s(0) + 0 < s(0) + 0 + 0 + s(0) \\
& \ldots\}
\end{aligned}
$$

Existence of Herbrand Models

A Herbrand interpretation I is called a **Herbrand model** of F iff $I \models F$.

Theorem

Let N be a set of Σ-clauses.

$$
\begin{array}{lll}
N \text{ is satisfiable} & \Leftrightarrow & N \text{ has a Herbrand model (over } \Sigma) \\
 & \Leftrightarrow & G_\Sigma(N) \text{ has a Herbrand model (over } \Sigma)
\end{array}
$$

where

$$G_\Sigma(N) = \{ C\sigma \text{ ground clause} \mid C \in N,\ \sigma : X \to \mathrm{T}_\Sigma \}$$

is the set of **ground instances** of N.

Example of a G_Σ

For Σ_{Pres} one obtains for

$$C = (x < y) \vee (y \leq s(x))$$

the following ground instances:

$(0 < 0) \vee (0 \leq s(0))$

$(s(0) < 0) \vee (0 \leq s(s(0)))$

$\ldots$

$(s(0) + s(0) < s(0) + 0) \vee (s(0) + 0 \leq s(s(0) + s(0)))$

$\ldots$

Herbrand's Theorem

Theorem (Skolem-Herbrand-Theorem)

$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

Herbrand's Theorem

Theorem (Skolem-Herbrand-Theorem)

$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

Applied to clause logic:

Theorem

Let N be a set of Σ-clauses.

N is unsatisfiable $\Leftrightarrow$ $G_\Sigma(N)$ has no Herbrand model (over Σ)

$\Leftrightarrow$ there is a **finite** subset of $G_\Sigma(N)$

that has no Herbrand model (over Σ)

Herbrand's Theorem

Theorem (Skolem-Herbrand-Theorem)

$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

Applied to clause logic:

Theorem

Let N be a set of Σ-clauses.

N is unsatisfiable $\Leftrightarrow$ $G_\Sigma(N)$ has no Herbrand model (over Σ)

$\Leftrightarrow$ there is a **finite** subset of $G_\Sigma(N)$ that has no Herbrand model (over Σ)

Significance: It's the core argument to show that validity in first-order logic is semi-decidable.

Part III: Proof Systems

Two fundamental results limit what can be achieved:

Theorem (Gödel, 1929)
There are proof systems that enumerate all valid formulas of first-order predicate logic. (This is also a consequence of Herbrand's Theorem)

Theorem (Church/Turing, about 1935)
The validity problem of first-order logic formulas is undecidable.

Part III: Proof Systems

Two fundamental results limit what can be achieved:

Theorem (Gödel, 1929)
There are proof systems that enumerate all valid formulas of first-order predicate logic. (This is also a consequence of Herbrand's Theorem)

Theorem (Church/Turing, about 1935)
The validity problem of first-order logic formulas is undecidable.

(Thus, the model existence problem is undecidable, too.)

Part III: Proof Systems

Two fundamental results limit what can be achieved:

Theorem (Gödel, 1929)
There are proof systems that enumerate all valid formulas of first-order predicate logic. (This is also a consequence of Herbrand's Theorem)

Theorem (Church/Turing, about 1935)
The validity problem of first-order logic formulas is undecidable.

(Thus, the model existence problem is undecidable, too.)

Automated theorem proving is oriented at the first, positive result.

Inference Systems and Proofs

Inference systems Γ (proof calculi) are sets of tuples

$$(F_1, \ldots, F_n, F_{n+1}), \ n \geq 0,$$

called **inferences** or **inference rules**, and written

$$\frac{\overbrace{F_1 \ \ldots \ F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}} \ .$$

Clausal inference system: premises and conclusions are clauses. One also considers inference systems over other data structures.

Proofs

A **proof** in Γ of a formula F from a a set of formulas N (called **assumptions**) is a sequence $F_1, \ldots, F_k$ of formulas where

1. $F_k = F$,
2. for all $1 \leq i \leq k$: $F_i \in N$, or else there exists an inference $(F_{i_1}, \ \ldots, \ F_{i_{n_i}}, \ F_i)$ in Γ, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness and Completeness

Provability $\vdash_\Gamma$ of F from N in Γ:

$N \vdash_\Gamma F \quad :\Leftrightarrow$ there exists a proof Γ of F from N.

Γ is called **sound** $\quad :\Leftrightarrow$

$$\frac{F_1 \ \ldots \ F_n}{F} \in \Gamma \quad \Rightarrow \quad F_1, \ldots, F_n \models F$$

Γ is called **complete** $\quad :\Leftrightarrow$

$$N \models F \quad \Rightarrow \quad N \vdash_\Gamma F$$

Γ is called **refutationally complete** $\quad :\Leftrightarrow$

$$N \models \bot \quad \Rightarrow \quad N \vdash_\Gamma \bot$$

Soundness and Completeness

Proposition

1. Let Γ be sound. Then $N \vdash_\Gamma F \Rightarrow N \models F$
2. $N \vdash_\Gamma F \Rightarrow$ there exist $F_1, \ldots, F_n \in N$ s.t. $F_1, \ldots, F_n \vdash_\Gamma F$ (resembles compactness).

Proofs as Trees

markings	$\widehat{=}$	formulas			
leaves	$\widehat{=}$	assumptions and axioms			
other nodes	$\widehat{=}$	inferences:	conclusion	$\widehat{=}$	ancestor
			premises	$\widehat{=}$	direct descendants

$$
\cfrac{P(g(a,b)) \qquad \cfrac{\cfrac{\cfrac{P(f(a)) \lor Q(b) \qquad \cfrac{\cfrac{P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b)}{\neg P(f(a)) \lor Q(b) \lor Q(b)}}{\neg P(f(a)) \lor Q(b)}}{Q(b) \lor Q(b)}}{Q(b)} \qquad \neg P(f(a)) \lor \neg Q(b)}{\neg P(g(a,b))}}{\bot}
$$

Proof Systems

The Aunta Agatha puzzle has shown that a proof system has to combine

- instantiation of variables with
- treatment of Boolean connectives.

In the subsequent slides we will concentrate on the second aspect and assume ground clauses, i.e. clauses where all variables have been instantiated by ground terms.

We observe that ground clauses and propositional clauses are the same concept.

Thus, for the time being we only deal with propositional clauses.

The subsequent **Resolution Calculus** *Res* can be used to decide the satisfiability problem of propositional clause logic.

The Resolution Calculus *Res*

Resolution inference rule:

$$\frac{C \lor A \qquad \neg A \lor D}{C \lor D}$$

Terminology: $C \lor D$: **resolvent**; A: **resolved atom**

(Positive) factorisation inference rule:

$$\frac{C \lor A \lor A}{C \lor A}$$

These are **schematic inference rules**; for each substitution of the **schematic variables** C, D, and A, respectively, by ground clauses and ground atoms we obtain an inference rule.

As "$\lor$" is considered associative and commutative, we assume that A and $\neg A$ can occur anywhere in their respective clauses.

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \vee \neg P(f(a)) \vee Q(b)$	(given)
2.	$P(f(a)) \vee Q(b)$	(given)
3.	$\neg P(g(b,a)) \vee \neg Q(b)$	(given)
4.	$P(g(b,a))$	(given)

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \vee \neg P(f(a)) \vee Q(b)$	(given)
2.	$P(f(a)) \vee Q(b)$	(given)
3.	$\neg P(g(b,a)) \vee \neg Q(b)$	(given)
4.	$P(g(b,a))$	(given)
5.	$\neg P(f(a)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \lor \neg P(f(a)) \lor Q(b)$	(given)
2.	$P(f(a)) \lor Q(b)$	(given)
3.	$\neg P(g(b, a)) \lor \neg Q(b)$	(given)
4.	$P(g(b, a))$	(given)
5.	$\neg P(f(a)) \lor Q(b) \lor Q(b)$	(Res. 2. into 1.)
6.	$\neg P(f(a)) \lor Q(b)$	(Fact. 5.)

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \vee \neg P(f(a)) \vee Q(b)$	(given)
2.	$P(f(a)) \vee Q(b)$	(given)
3.	$\neg P(g(b,a)) \vee \neg Q(b)$	(given)
4.	$P(g(b,a))$	(given)
5.	$\neg P(f(a)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)
6.	$\neg P(f(a)) \vee Q(b)$	(Fact. 5.)
7.	$Q(b) \vee Q(b)$	(Res. 2. into 6.)

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \vee \neg P(f(a)) \vee Q(b)$	(given)
2.	$P(f(a)) \vee Q(b)$	(given)
3.	$\neg P(g(b, a)) \vee \neg Q(b)$	(given)
4.	$P(g(b, a))$	(given)
5.	$\neg P(f(a)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)
6.	$\neg P(f(a)) \vee Q(b)$	(Fact. 5.)
7.	$Q(b) \vee Q(b)$	(Res. 2. into 6.)
8.	$Q(b)$	(Fact. 7.)

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \vee \neg P(f(a)) \vee Q(b)$	(given)
2.	$P(f(a)) \vee Q(b)$	(given)
3.	$\neg P(g(b, a)) \vee \neg Q(b)$	(given)
4.	$P(g(b, a))$	(given)
5.	$\neg P(f(a)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)
6.	$\neg P(f(a)) \vee Q(b)$	(Fact. 5.)
7.	$Q(b) \vee Q(b)$	(Res. 2. into 6.)
8.	$Q(b)$	(Fact. 7.)
9.	$\neg P(g(b, a))$	(Res. 8. into 3.)

Sample Refutation

By the just made observation, this is a propositional clause set:

1.	$\neg P(f(a)) \vee \neg P(f(a)) \vee Q(b)$	(given)
2.	$P(f(a)) \vee Q(b)$	(given)
3.	$\neg P(g(b, a)) \vee \neg Q(b)$	(given)
4.	$P(g(b, a))$	(given)
5.	$\neg P(f(a)) \vee Q(b) \vee Q(b)$	(Res. 2. into 1.)
6.	$\neg P(f(a)) \vee Q(b)$	(Fact. 5.)
7.	$Q(b) \vee Q(b)$	(Res. 2. into 6.)
8.	$Q(b)$	(Fact. 7.)
9.	$\neg P(g(b, a))$	(Res. 8. into 3.)
10.	$\perp$	(Res. 4. into 9.)

Soundness of Resolution

Proposition

Propositional resolution is sound.

Proof:

Let $I \in \Sigma$-Alg. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factorization: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Soundness of Resolution

Proposition

Propositional resolution is sound.

Proof:

Let $I \in \Sigma$-Alg. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factorization: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered:
(a) A is valid in I, or (b) $\neg A$ is valid in I.

Soundness of Resolution

Proposition

Propositional resolution is sound.

Proof:

Let $I \in \Sigma$-Alg. To be shown:

1. for resolution: $I \models C \lor A,\ I \models D \lor \neg A \ \Rightarrow\ I \models C \lor D$
2. for factorization: $I \models C \lor A \lor A \ \Rightarrow\ I \models C \lor A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \lor D$

Soundness of Resolution

Proposition

Propositional resolution is sound.

Proof:

Let $I \in \Sigma$-Alg. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factorization: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \vee D$

b) $I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \vee D$

Soundness of Resolution

Proposition

Propositional resolution is sound.

Proof:

Let $I \in \Sigma$-Alg. To be shown:

1. for resolution: $I \models C \vee A,\ I \models D \vee \neg A \ \Rightarrow\ I \models C \vee D$
2. for factorization: $I \models C \vee A \vee A \ \Rightarrow\ I \models C \vee A$

Ad (i): Assume premises are valid in I. Two cases need to be considered: (a) A is valid in I, or (b) $\neg A$ is valid in I.

a) $I \models A \Rightarrow I \models D \Rightarrow I \models C \vee D$

b) $I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \vee D$

Ad (ii): even simpler.

Resolution is also refutationally complete.

Methods for First-Order Clause Logic

Treated here:

- Gilmore's method (considered "naive" nowadays)
- The Resolution Calculus

  The Resolution Calculus [Robinson 1965] (for first-order clause logic) is much better suited for automatization on a computer than earlier calculi:

  - Simpler (one single inference rule)
  - Less search space

There are other methods that are not based on Resolution:

- Tableaux and connection methods, Model Elimination (see later)
- Instance Based Methods (separate lecture)

Gilmore's Method

Early method for FOTP, directly based on Herbrand's theorem

Preprocessing:

Given Formula

$$\forall x\, \exists y\, P(y, x) \wedge \forall z\, \neg P(z, a)$$

$\longrightarrow$

Clause Form

$$P(f(x), x)$$
$$\neg P(z, a)$$

Outer loop:
Grounding

Inner loop:
Propositional Method

Gilmore’s Method

Early method for FOTP, directly based on Herbrand’s theorem

Preprocessing:

Given Formula

$$\forall x\, \exists y\, P(y,x) \\ \wedge \forall z\, \neg P(z,a)$$

$\longrightarrow$

Clause Form

$$P(f(x),x) \\ \neg P(z,a)$$

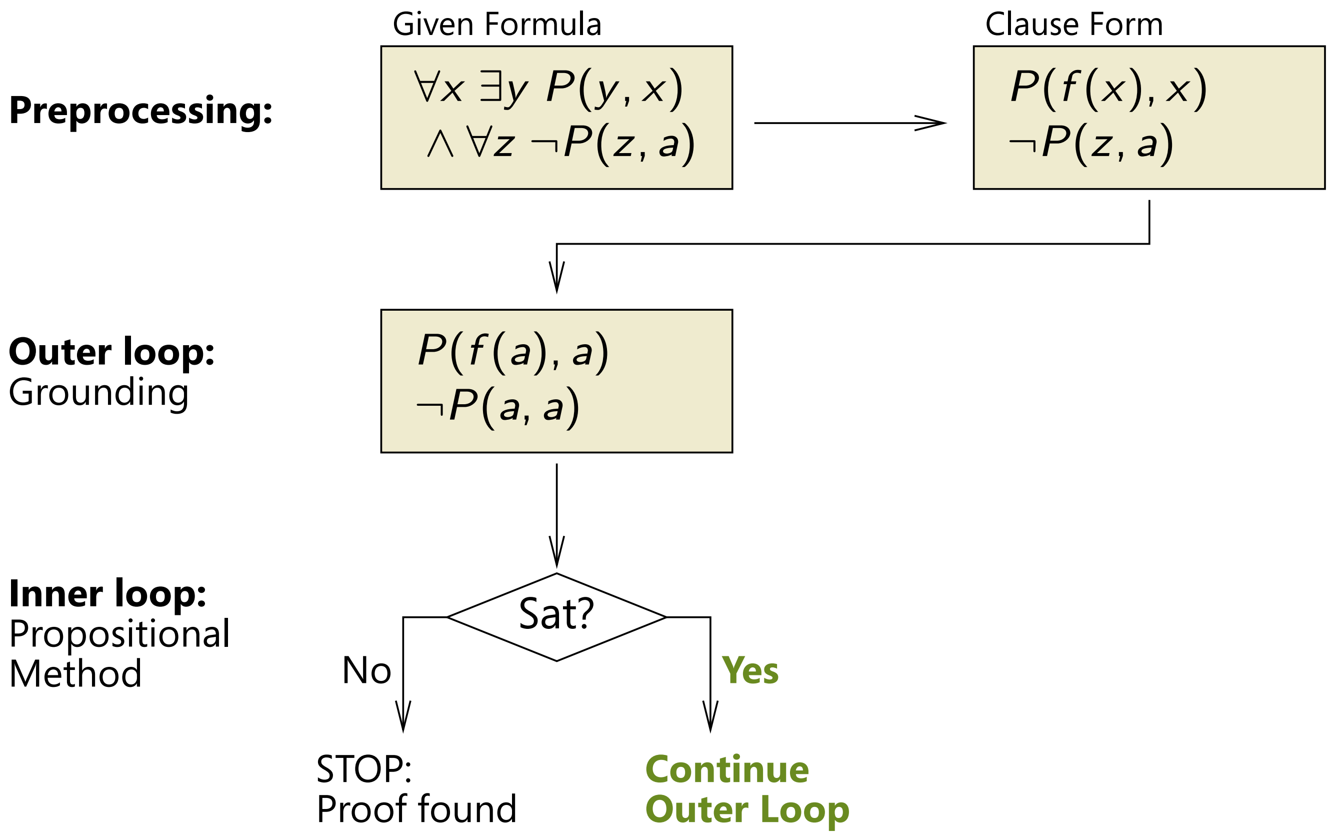
Outer loop:
Grounding

$$P(f(a),a) \\ \neg P(a,a)$$

Inner loop:
Propositional
Method

Gilmore's Method

Early method for FOTP, directly based on Herbrand's theorem



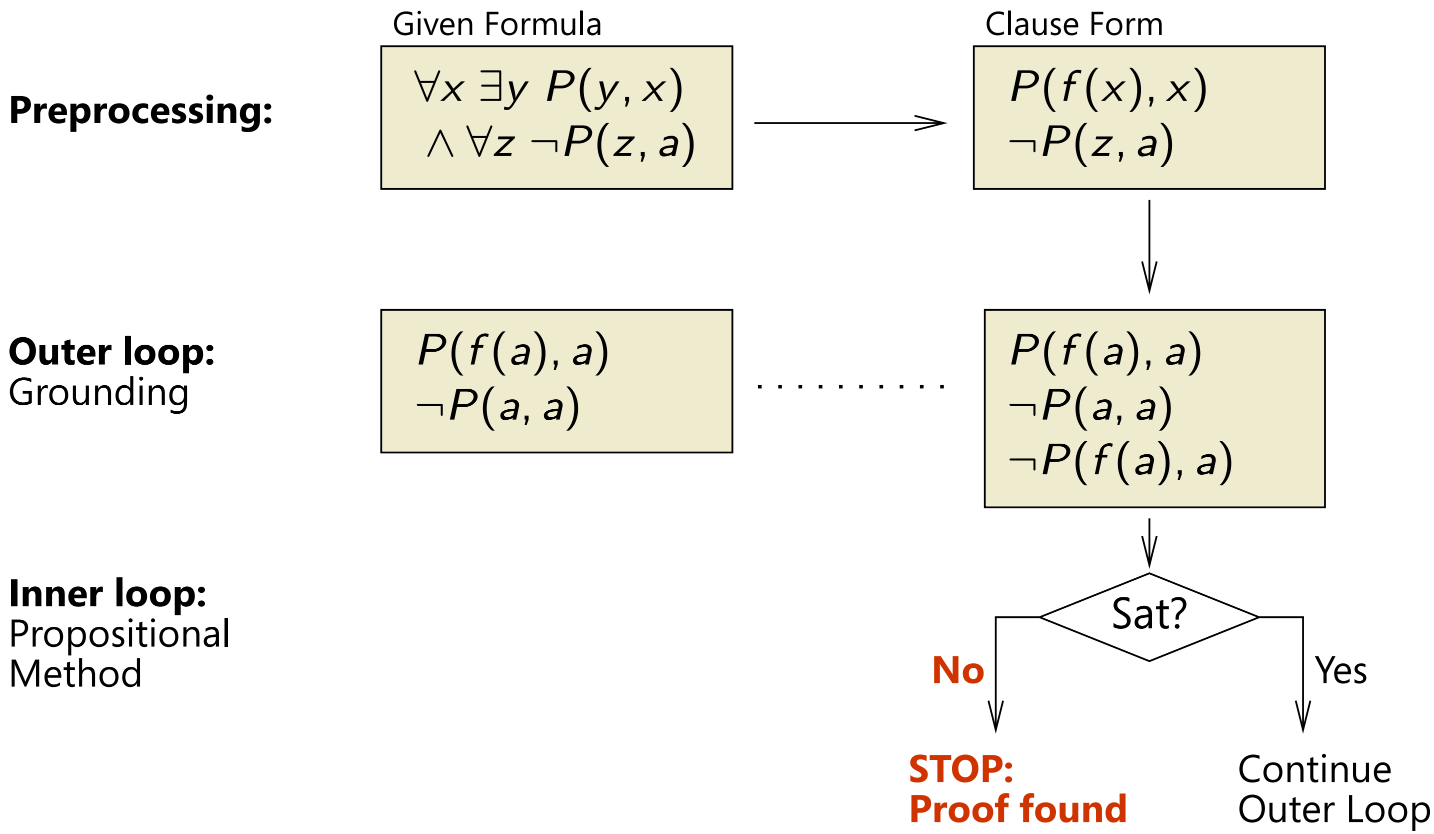
Gilmore's Method

Early method for FOTP, directly based on Herbrand's theorem

	Given Formula		Clause Form
Preprocessing:	$\forall x\, \exists y\, P(y, x)$ $\land \forall z\, \neg P(z, a)$	$\longrightarrow$	$P(f(x), x)$ $\neg P(z, a)$
			$\downarrow$
Outer loop: Grounding	$P(f(a), a)$ $\neg P(a, a)$	$\cdots\cdots\cdots$	$P(f(a), a)$ $\neg P(a, a)$ $\neg P(f(a), a)$
Inner loop: Propositional Method			

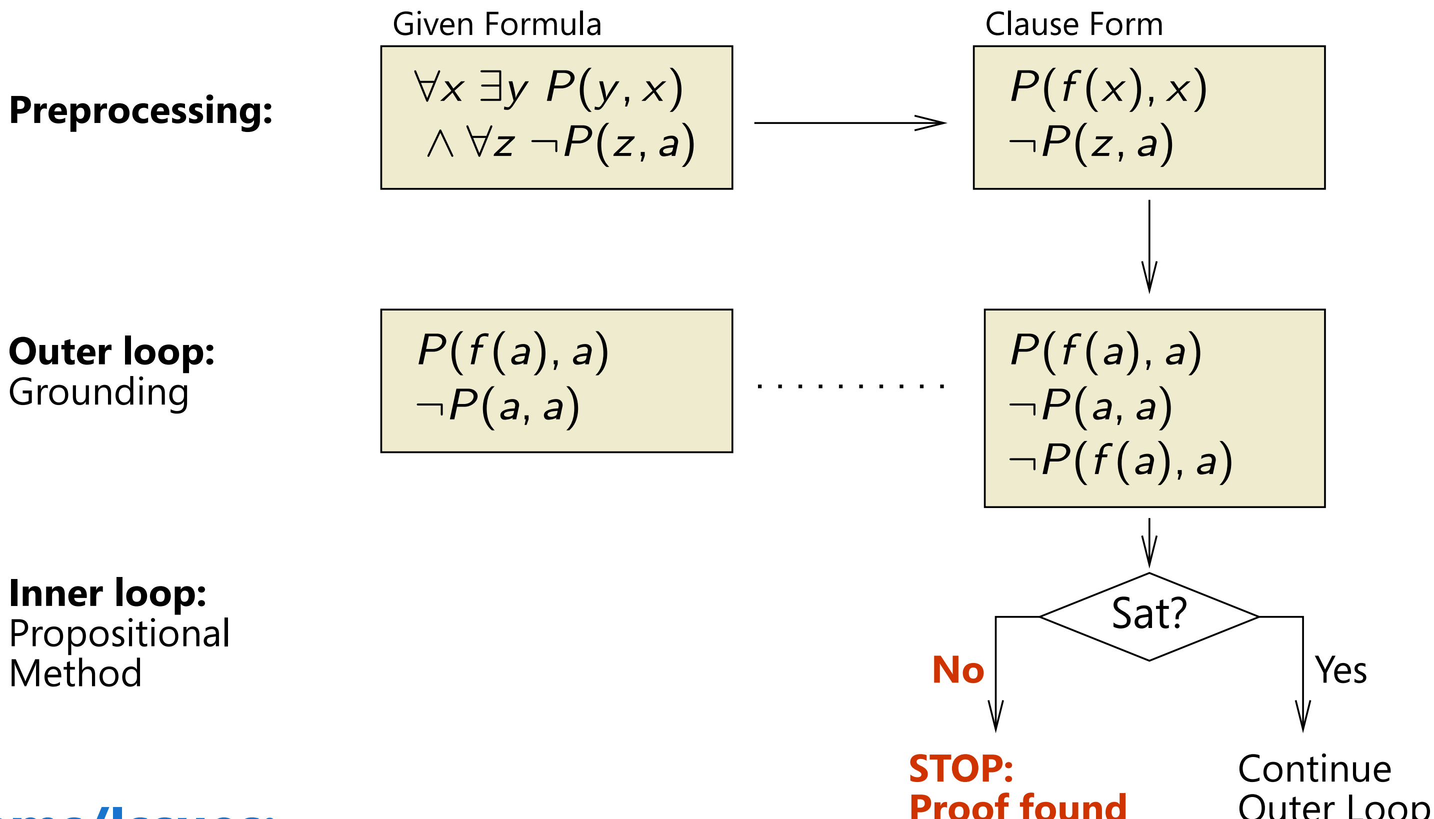
Gilmore's Method

Early method for FOTP, directly based on Herbrand's theorem



Gilmore’s Method

Early method for FOTP, directly based on Herbrand’s theorem



Problems/Issues:

- Controlling the grounding process in **outer loop** (irrelevant instances)
- Repeat work **across** inner loops
- Weak redundancy criterion **within** inner loop

. . . Versus Resolution

Central Point: Resolution performs **intrinsic first-order reasoning**

. . . Versus Resolution

Central Point: Resolution performs **intrinsic first-order reasoning**

Resolution inferences on first-order clauses (clauses with variables):

$$\frac{P(f(x), x) \qquad \neg P(y, z) \lor Q(y, z)}{Q(f(x), x)}$$

. . . Versus Resolution

Central Point: Resolution performs **intrinsic first-order reasoning**

Resolution inferences on first-order clauses (clauses with variables):

$$\frac{P(f(x), x) \qquad \neg P(y, z) \vee Q(y, z)}{Q(f(x), x)}$$

One inference may represent infinitely many propositional resolution inferences (“lifting principle”)

. . . Versus Resolution

Central Point: Resolution performs **intrinsic first-order reasoning**

Resolution inferences on first-order clauses (clauses with variables):

$$\begin{array}{c} P(f(x),x) \qquad \neg P(y,z) \vee Q(y,z) \\ \hline Q(f(x),x) \end{array}$$

One inference may represent infinitely many propositional resolution inferences ("lifting principle")

Redundancy concepts, e.g. **subsumption deletion**:

$$P(y,z) \quad \text{subsumes} \quad P(y,y) \vee Q(y,y)$$

. . . Versus Resolution

Central Point: Resolution performs **intrinsic first-order reasoning**

Resolution inferences on first-order clauses (clauses with variables):

$$\frac{P(f(x), x) \qquad \neg P(y, z) \vee Q(y, z)}{Q(f(x), x)}$$

One inference may represent infinitely many propositional resolution inferences ("lifting principle")

Redundancy concepts, e.g. **subsumption deletion**:

$$P(y, z) \quad \text{subsumes} \quad P(y, y) \vee Q(y, y)$$

Not available in Gilmore's method

First-Order Resolution through Instantiation

Idea: instantiate clauses to ground clauses:

$$
\begin{tikzcd}
P(z',z') \lor \neg Q(z) \arrow[d, "{[a/z', f(a,b)/z]}"'] & & & \neg P(a,y) \arrow[dl, "{[a/y]}"'] \arrow[dr, "{[b/y]}"] & & & P(x',b) \lor Q(f(x',x)) \arrow[d, "{[a/x', b/x]}"] \\
P(a,a) \lor \neg Q(f(a,b)) \arrow[dr] & & \neg P(a,a) \arrow[dl] & & \neg P(a,b) \arrow[dr] & & P(a,b) \lor Q(f(a,b)) \arrow[dl] \\
& \neg Q(f(a,b)) \arrow[d] & & & & Q(f(a,b)) \arrow[d] & \\
& {} \arrow[drr] & & & & {} \arrow[dll] & \\
& & & \perp & & &
\end{tikzcd}
$$

Bears ressemblance with Gilmore's method.

First-Order Resolution through Instantiation

Problems

- More than one instance of a clause can participate in a proof.
- Even worse: There are infinitely many possible instances.

First-Order Resolution through Instantiation

Problems

- More than one instance of a clause can participate in a proof.
- Even worse: There are infinitely many possible instances.

Observation

- Instantiation must produce complementary literals
  (so that inferences become possible).

First-Order Resolution through Instantiation

Problems

- More than one instance of a clause can participate in a proof.
- Even worse: There are infinitely many possible instances.

Observation

- Instantiation must produce complementary literals
  (so that inferences become possible).

Idea

- Do not instantiate more than necessary to get complementary literals.

First-Order Resolution

Idea: do not instantiate more than necessary:

$$
\begin{tikzcd}
P(z',z') \lor \neg Q(z) \arrow[d, swap, "{[a/z']}"] & & & \neg P(a,y) \arrow[dl, swap, "{[a/y]}"] \arrow[dr, "{[b/y]}"] & & & P(x',b) \lor Q(f(x',x)) \arrow[d, "{[a/x']}"] \\
P(a,a) \lor \neg Q(z) \arrow[dr] & & \neg P(a,a) \arrow[dl] & & \neg P(a,b) \arrow[dr] & & P(a,b) \lor Q(f(a,x)) \arrow[dl] \\
& \neg Q(z) \arrow[d, swap, "{[f(a,x)/z]}"] & & & & Q(f(a,x)) \arrow[d] & \\
& \neg Q(f(a,x)) \arrow[drr] & & & & Q(f(a,x)) \arrow[dll] & \\
& & & \bot & & &
\end{tikzcd}
$$

Lifting Principle

Problem: Make closure under Resolution and Factorization of infinite sets of clauses as they arise from taking the (ground) instances of finitely many **first-order** clauses (with variables) effective and efficient.

Idea (Robinson 65):

- Resolution for first-order clauses:
- **Equality** of ground atoms is generalized to **unifiability** of general atoms;
- Only compute **most general** (minimal) unifiers.

Lifting Principle

Significance: The advantage of the method in (Robinson 65) compared with (Gilmore 60) is that unification enumerates only those instances of clauses that participate in an inference.

Moreover, clauses are not right away instantiated into ground clauses. Rather they are instantiated only as far as required for an inference. Inferences with non-ground clauses in general represent infinite sets of ground inferences which are computed simultaneously in a single step.

Resolution for First-Order Clauses

$$\frac{C \lor A \qquad D \lor \neg B}{(C \lor D)\sigma} \quad \text{if } \sigma = \text{mgu}(A, B) \qquad \text{[resolution]}$$

$$\frac{C \lor A \lor B}{(C \lor A)\sigma} \quad \text{if } \sigma = \text{mgu}(A, B) \quad \text{[factorization]}$$

In both cases, A and B have to be renamed apart (made variable disjoint).

Resolution for First-Order Clauses

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma} \quad \text{if } \sigma = \text{mgu}(A, B) \qquad \text{[resolution]}$$

$$\frac{C \vee A \vee B}{(C \vee A)\sigma} \quad \text{if } \sigma = \text{mgu}(A, B) \quad \text{[factorization]}$$

In both cases, A and B have to be renamed apart (made variable disjoint).

Example

$$\frac{Q(z) \vee P(z, z) \qquad \neg P(x, y)}{Q(x)} \quad \text{where } \sigma = [x/z,\ x/y] \qquad \text{[resolution]}$$

$$\frac{Q(z) \vee P(z, a) \vee P(a, y)}{Q(a) \vee P(a, a)} \quad \text{where } \sigma = [a/z,\ a/y] \quad \text{[factorization]}$$

Unification

A **substitution** σ is a mapping from variables to terms which is the identity almost everywhere.

Example: $\sigma = [f(a, x)/z,\ b/y]$

Unification

A **substitution** σ is a mapping from variables to terms which is the identity almost everywhere.

Example: $\sigma = [f(a, x)/z,\ b/y]$

A substitutions can be **applied** to a term t, written as $t\sigma$.

Example, where σ is from above: $g(x, y, z)\sigma = g(x, b, f(a, x))$.

Unification

A **substitution** σ is a mapping from variables to terms which is the identity almost everywhere.
Example: $\sigma = [f(a, x)/z,\ b/y]$

A substitutions can be **applied** to a term t, written as $t\sigma$.
Example, where σ is from above: $g(x, y, z)\sigma = g(x, b, f(a, x))$.

Let $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ (s_i, t_i terms or atoms) a multi-set of **equality problems.**

A substitution σ is called a **unifier** of E if $s_i\sigma = t_i\sigma$ for all $1 \leq i \leq n$.

If a unifier of E exists, then E is called **unifiable.**

Unification

A substitution σ is called **more general** than a substitution τ, denoted by $\sigma \leq \tau$, if there exists a substitution ρ such that $\rho \circ \sigma = \tau$, where $(\rho \circ \sigma)(x) := (x\sigma)\rho$ is the composition of σ and ρ as mappings.

If a unifier of E is more general than any other unifier of E, then we speak of a **most general unifier** of E, denoted by $\mathrm{mgu}(E)$.

Unification after Martelli/Montanari

$$
\begin{array}{rcl}
t \doteq t, E & \Rightarrow_{MM} & E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E & \Rightarrow_{MM} & s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E & \Rightarrow_{MM} & \bot \\
x \doteq t, E & \Rightarrow_{MM} & x \doteq t, E[t/x] \\
 & & \text{if } x \in var(E), x \notin var(t) \\
x \doteq t, E & \Rightarrow_{MM} & \bot \\
 & & \text{if } x \neq t, x \in var(t) \\
t \doteq x, E & \Rightarrow_{MM} & x \doteq t, E \\
 & & \text{if } t \notin X
\end{array}
$$

MM: Main Properties

If $E = x_1 \doteq u_1, \ldots, x_k \doteq u_k$, with x_i pairwise distinct, $x_i \notin var(u_j)$, then E is called (an equational problem) in
solved form representing the solution $\sigma_E = [u_1/x_1, \ldots, u_k/x_k]$.

Proposition

If E is a solved form then σ_E is am mgu of E.

MM: Main Properties

Theorem

1. If $E \Rightarrow_{MM} E'$ then σ is a (most general) unifier of E iff σ is a (most general) unifier of E'

MM: Main Properties

Theorem

1. If $E \Rightarrow_{MM} E'$ then σ is a (most general) unifier of E iff σ is a (most general) unifier of E'
2. If $E \Rightarrow^*_{MM} \perp$ then E is not unifiable.

MM: Main Properties

Theorem

1. If $E \Rightarrow_{MM} E'$ then σ is a (most general) unifier of E iff σ is a (most general) unifier of E'
2. If $E \Rightarrow^*_{MM} \bot$ then E is not unifiable.
3. If $E \Rightarrow^*_{MM} E'$ with E' in solved form, then $\sigma_{E'}$ is an mgu of E.

MM: Main Properties

Theorem

1. If $E \Rightarrow_{MM} E'$ then σ is a (most general) unifier of E iff σ is a (most general) unifier of E'
2. If $E \Rightarrow^*_{MM} \bot$ then E is not unifiable.
3. If $E \Rightarrow^*_{MM} E'$ with E' in solved form, then $\sigma_{E'}$ is an mgu of E.

Theorem

E is unifiable if and only if there is a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.

Problem: **exponential growth** of terms possible

Properties of Resolution

Theorem: Resolution is **sound**. That is, all derived formulas are logical consequences of the given ones

Properties of Resolution

Theorem: Resolution is **sound**. That is, all derived formulas are logical consequences of the given ones

Theorem: Resolution is **refutationally complete**. That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\perp$ eventually.

Properties of Resolution

Theorem: Resolution is **sound**. That is, all derived formulas are logical consequences of the given ones

Theorem: Resolution is **refutationally complete**. That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\perp$ eventually.

More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factorization inference rules, then it contains the empty clause $\perp$.

Properties of Resolution

Theorem: Resolution is **sound**. That is, all derived formulas are logical consequences of the given ones

Theorem: Resolution is **refutationally complete**. That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\bot$ eventually.

More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factorization inference rules, then it contains the empty clause $\bot$.

Perhaps easiest proof: Herbrand Theorem + Semantic Tree proof technique + Lifting Theorem

(This result can be considerably strengthened using other techniques)

Properties of Resolution

Theorem: Resolution is **sound**. That is, all derived formulas are logical consequences of the given ones

Theorem: Resolution is **refutationally complete**. That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\perp$ eventually.

More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factorization inference rules, then it contains the empty clause $\perp$.

Perhaps easiest proof: Herbrand Theorem + Semantic Tree proof technique + Lifting Theorem

(This result can be considerably strengthened using other techniques)

Closure can be achieved by the “Given Clause Loop” on next slide.

The “Given Clause Loop”

As used in the Otter theorem prover:

Lists of clauses maintained by the algorithm: `usable` and `sos`.

Initialize `sos` with the input clauses, `usable` empty.

Algorithm (straight from the Otter manual):

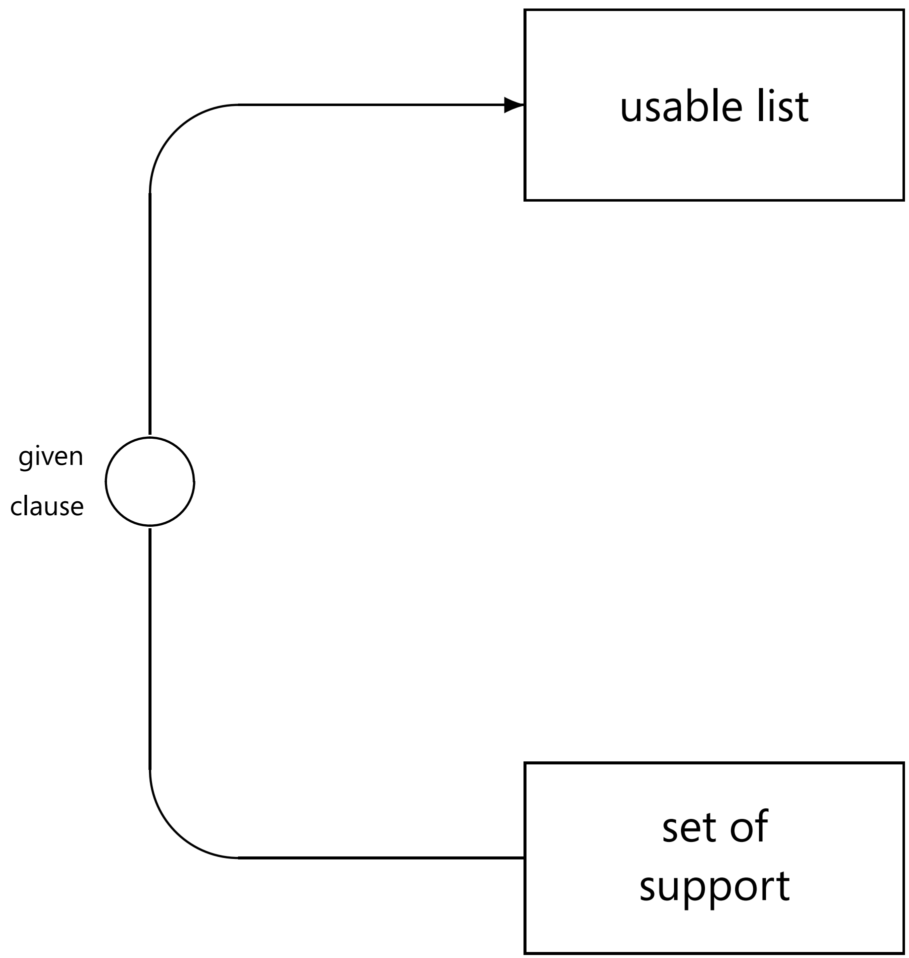
```
While (sos is not empty and no refutation has been found)
  1. Let given_clause be the ‘lightest’ clause in sos;
  2. Move given_clause from sos to usable;
  3. Infer and process new clauses using the inference rules in
     effect; each new clause must have the given_clause as
     one of its parents and members of usable as its other
     parents;  new clauses that pass the retention tests
     are appended to sos;
End of while loop.
```

Fairness: define clause weight e.g. as “depth + length” of clause.

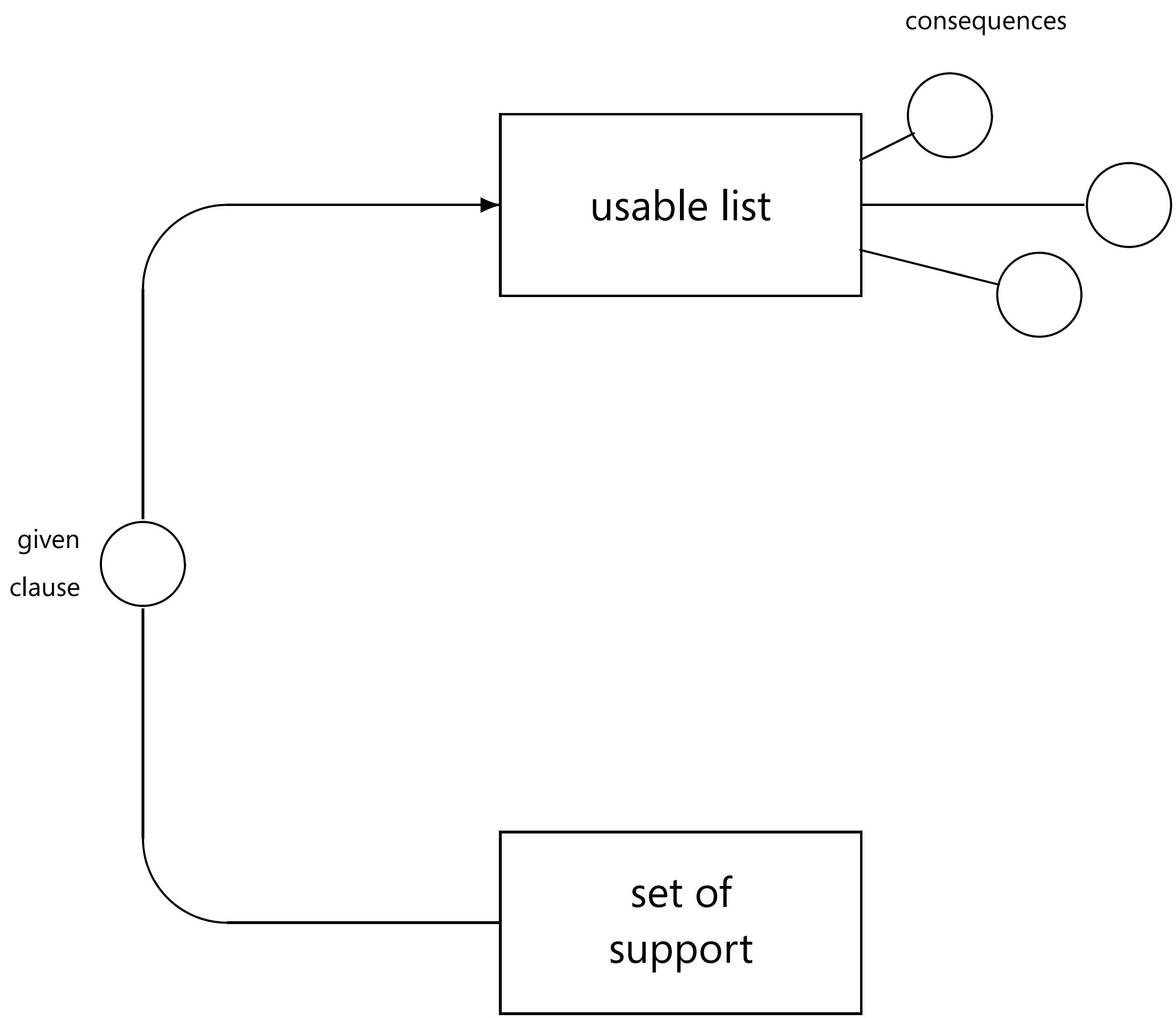
The “Given Clause Loop” - Graphically

The "Given Clause Loop" - Graphically

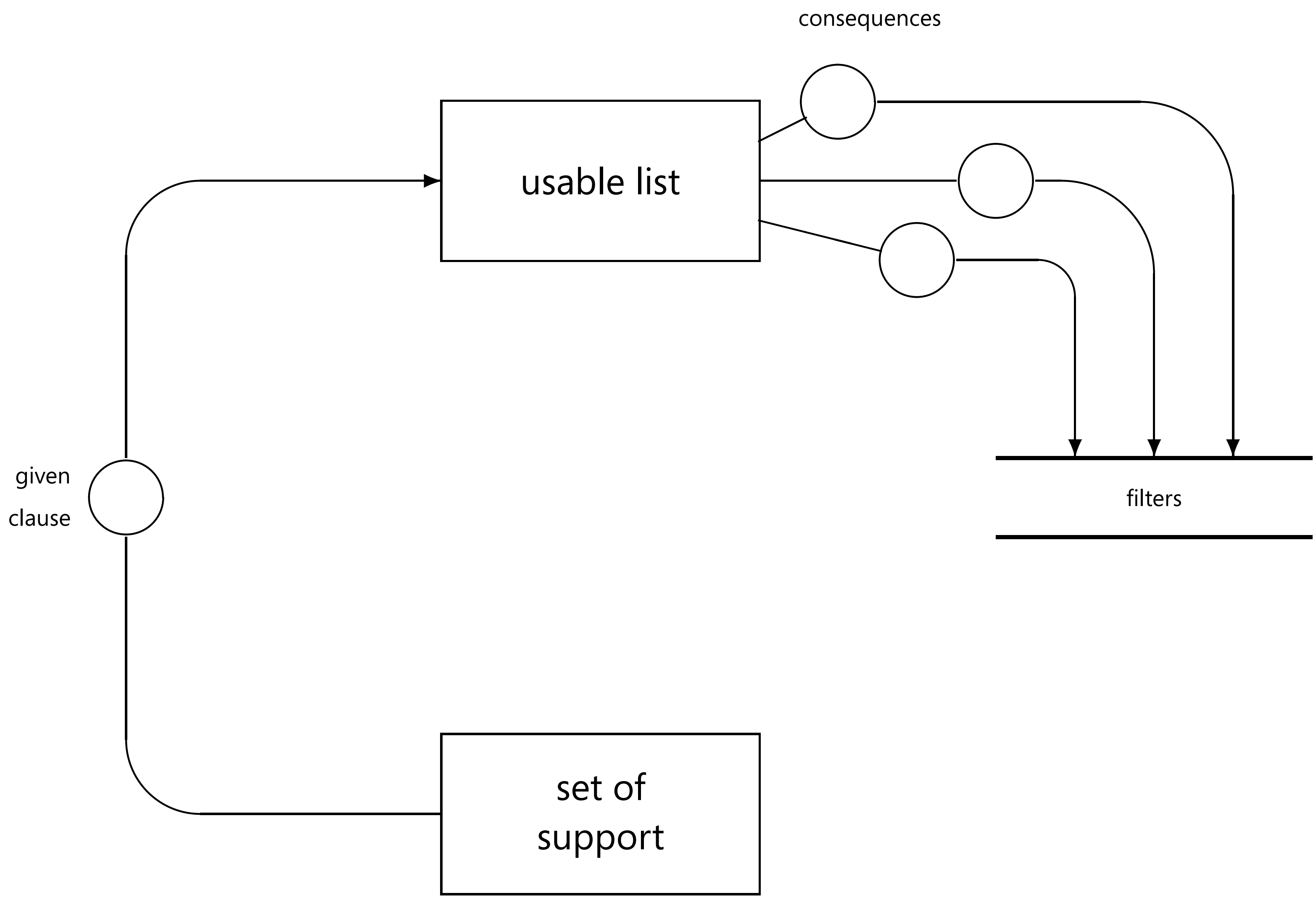
The "Given Clause Loop" - Graphically



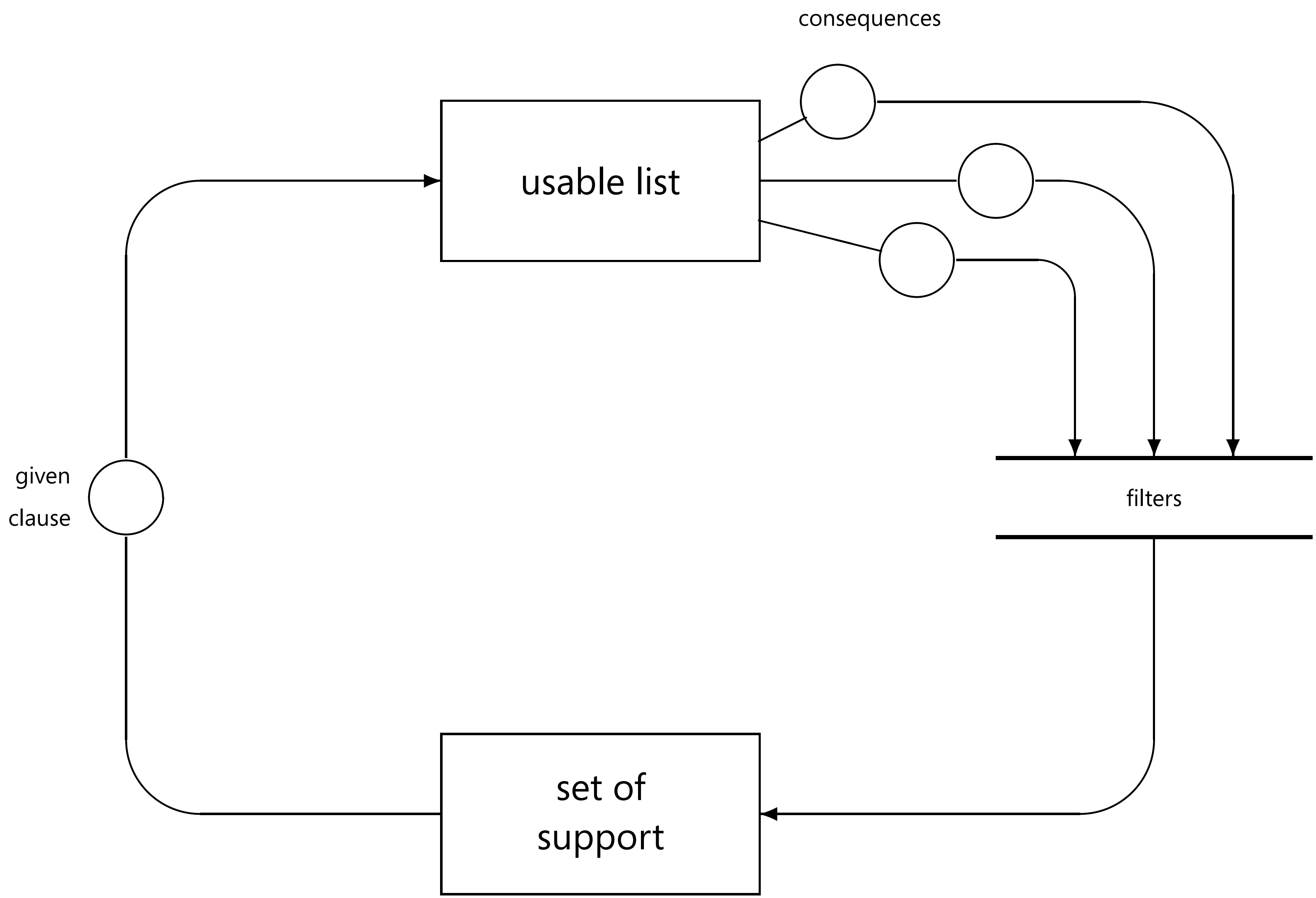
The “Given Clause Loop” - Graphically



The “Given Clause Loop” - Graphically



The "Given Clause Loop" - Graphically



Part IV: Model Generation and Tableaux

No "theorem" clause, cannot use Resolution to derived a contradiction.

Ideally, can detect satisfiability by computing a model.

Why compute models?

Planning: Can be formalised as propositional satisfiability problem.

[Kautz& Selman, AAAI96; Dimopolous et al, ECP97]

Diagnosis: Minimal models of *abnormal* literals (circumscription).

[Reiter, AI87]

Databases: View materialisation, View Updates, Integrity Constraints.

Nonmonotonic reasoning: Various semantics (GCWA, Well-founded, Perfect, Stable,...), all based on minimal models. [Inoue et al, CADE 92]

Software Verification: Counterexamples to conjectured theorems.

Theorem proving: Counterexamples to conjectured theorems. Finite models of quasigroups, (MGTP/G). [Fujita et al, IJCAI 93]

Part IV: Model Generation and Tableaux

Why compute models (cont'd)?

Natural Language Processing:

- Maintain models $\mathfrak{I}_1, \ldots, \mathfrak{I}_n$ as different readings of discourses:

$$\mathfrak{I}_i \models \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far}$$

Part IV: Model Generation and Tableaux

Why compute models (cont'd)?

Natural Language Processing:

- Maintain models $\mathfrak{I}_1, \ldots, \mathfrak{I}_n$ as different readings of discourses:

$$\mathfrak{I}_i \models \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far}$$

- Consistency checks ("Mia's husband loves Sally. She is not married.")

$$\begin{array}{ll} & \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far} \not\models \neg \mathit{New\_utterance} \\ \mathit{iff} & \mathit{BG\text{-}Knowledge} \cup \mathit{Discourse\_so\_far} \cup \mathit{New\_utterance} \text{ is } \textbf{satisfiable} \end{array}$$

Part IV: Model Generation and Tableaux

Why compute models (cont'd)?

Natural Language Processing:

- Maintain models $\mathfrak{I}_1, \ldots, \mathfrak{I}_n$ as different readings of discourses:

$$\mathfrak{I}_i \models \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far}$$

- Consistency checks ("Mia's husband loves Sally. She is not married.")

$$\begin{array}{ll} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \not\models \neg\textit{New\_utterance} \\ \textit{iff} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \cup \textit{New\_utterance} \text{ is } \textbf{satisfiable} \end{array}$$

- Informativity checks ("Mia's husband loves Sally. She is married.")

$$\begin{array}{ll} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \not\models \textit{New\_utterance} \\ \textit{iff} & \textit{BG-Knowledge} \cup \textit{Discourse\_so\_far} \cup \neg\textit{New\_utterance} \text{ is } \textbf{satisfiable} \end{array}$$

Tableaux

Calculi with a long history

- Beth 1955, Hintikka 1955, Schütte 1956: Calculi without meta-language constructs, such as sequents.
  Nodes in derivation tree labeled by formulae.
- Lis 1960, Smullyan 1968: Analytic tabaleaux

Tableaux

Calculi with a long history

- Beth 1955, Hintikka 1955, Schütte 1956: Calculi without meta-language constructs, such as sequents.
  Nodes in derivation tree labeled by formulae.
- Lis 1960, Smullyan 1968: Analytic tabaleaux

Some later FOTP Calculi can be rephrased as Tableaux

- Loveland 1968 Model elimination, Kowalski, Kuehner 1971 SL-resolution
- Bibel 1975, Andrews 1976 Connection or matings methods.

Tableaux

Calculi with a long history

- Beth 1955, Hintikka 1955, Schütte 1956: Calculi without meta-language constructs, such as sequents.
  Nodes in derivation tree labeled by formulae.
- Lis 1960, Smullyan 1968: Analytic tabaleaux

Some later FOTP Calculi can be rephrased as Tableaux

- Loveland 1968 Model elimination, Kowalski, Kuehner 1971 SL-resolution
- Bibel 1975, Andrews 1976 Connection or matings methods.

Significance

- Various non-classical logics (modal, sub-structural, . . .)
- ATP in Description Logics (cf. Knowledge Representation lectures)

Tableaux

Calculi with a long history

- Beth 1955, Hintikka 1955, Schütte 1956: Calculi without meta-language constructs, such as sequents.
  Nodes in derivation tree labeled by formulae.
- Lis 1960, Smullyan 1968: Analytic tabaleaux

Some later FOTP Calculi can be rephrased as Tableaux

- Loveland 1968 Model elimination, Kowalski, Kuehner 1971 SL-resolution
- Bibel 1975, Andrews 1976 Connection or matings methods.

Significance

- Various non-classical logics (modal, sub-structural, . . .)
- ATP in Description Logics (cf. Knowledge Representation lectures)
- **Model generation**

Analytic Tableaux

Given set of propositional formulae, e.g. $\{\neg P \land \neg(Q \lor R), \neg(Q \land \neg R)\}$

Construct a tree by using Tableau extension rules:

$$\neg P \land \neg(Q \lor R)$$

$$\neg(Q \land \neg R)$$

$$\begin{array}{ccc}
\neg Q & & \neg\neg R \\
| & & | \\
\neg P & & R \\
| & & \vdots \\
\neg(Q \lor R) & & \\
| & & \star \\
\neg Q & & \\
| & & \\
\neg R & &
\end{array}$$

Analytic Tableaux

Given set of propositional formulae, e.g. $\{\neg P \land \neg(Q \lor R), \neg(Q \land \neg R)\}$

Construct a tree by using Tableau extension rules:

$$\neg P \land \neg(Q \lor R)$$

$$\neg(Q \land \neg R)$$

$$\neg Q \qquad\qquad\qquad \neg\neg R$$

$$\neg P \qquad\qquad\qquad R$$

$$\neg(Q \lor R) \qquad\qquad\qquad \star$$

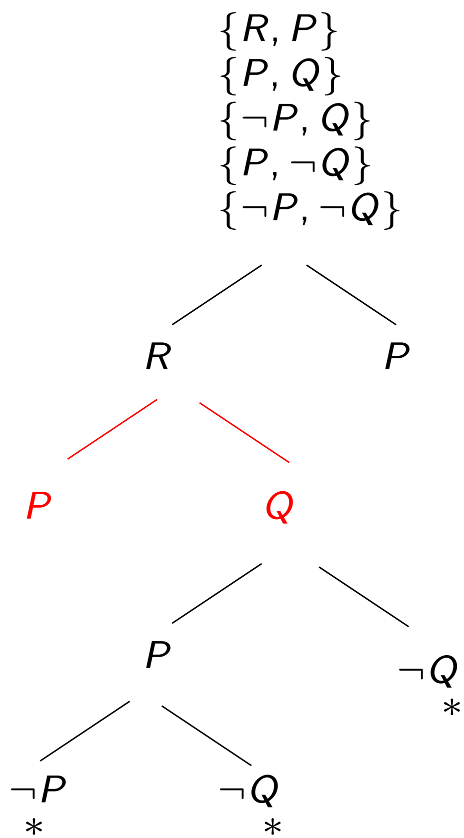
$$\neg Q$$

$$\neg R$$

Left branch is open (non-contradictory) and fully expanded: model

Clause Normalform Tableaux – Ground Case

Given a set of clauses, e.g. $\{\{R, P\}, \{P, Q\}, \{\neg P, Q\}, \{\neg Q, P\}, \{\neg P \neg Q\}\}$. From a one-path tree, consisting of a node for each clause, construct a tree by using the β-rule:



"Link condition" not satisfied
Can be demanded (or not)

First-Order Tableaux: The $P(x) \lor Q(x)$ problem

No Problem:

$$\forall x, y\ (P(x) \lor Q(y))$$

$$\leftrightarrow$$

$$\forall x\ P(x) \lor \forall y\ Q(y)$$

$$\begin{array}{cc} \forall x P(x) & \forall y Q(y) \\ | & | \\ \neg P(a) & \neg Q(b) \\ * & * \end{array}$$

x, y branch-local

universal variables

Problem:

$$\forall x\ (P(x) \lor Q(x))$$

$$\not\leftrightarrow$$

$$\forall x\ P(x) \lor \forall x\ Q(x)$$

$$\begin{array}{cc} P(X) & Q(X) \\ \uparrow ? & \uparrow ? \\ \neg P(a) & \neg Q(b) \end{array}$$

X split variable

rigid variable, stands for **one** ground term

Clause Normalform Tableaux – First Order Case

Allow max number *n* of γ-rule applications, arbitrary β-rule applications

Try **simultaneously** closing all branches by unifying literals;
increase *n* if unsuccessful and restart

$$\forall x(P(x) \vee Q(x))$$
$$\neg Q(b)$$
$$\neg P(a)$$
$$\neg Q(a) \vee r$$

$P(X) \vee Q(X)$ — γ-rule: copy of clause with rigid variables

$P(X)$ $\quad$ $Q(X)$ — β-rule: splitting

Branch closure candidate subst: $\sigma = [a/X]$

Clause Normalform Tableaux – First Order Case

Allow max number n of γ-rule applications, arbitrary β-rule applications

Try **simultaneously** closing all branches by unifying literals;
increase n if unsuccessful and restart

$$\forall x (P(x) \lor Q(x))$$
$$\neg Q(b)$$
$$\neg P(a)$$
$$\neg Q(a) \lor r$$

$P(X) \lor Q(X)$ $\qquad$ γ-rule: copy of clause with rigid variables

$P(X) \qquad Q(X)$ $\qquad$ β-rule: splitting

Branch closure candidate subst: $\sigma = [a/X]$

This formalism can be used to describe Prolog's SLD Resolution, Model Elimination, Connection Methods, Hyper Tableaux, ...

SATCHMO

[Manthey&Bry, CADE 88; Loveland et al, JAR 95]
[Bry&Yahya TABLEAUX 96; Geisler et al, JAR 97; Bry&Torge JELIA 98]

Significance: an early and simple method for model computation, can also be described as a tableaux method (without rigid variables)

1. Convert clauses to range-restricted form:

   $q(x) \vee p(x, y) \leftarrow q(x) \quad \leadsto \quad$ `q(X) ; p(X,Y) <- q(X), dom(Y)`

2. `assert` range-restricted clauses and dom clauses in Prolog database.

3. Call `satisfiable`:

```
satisfy :-
    (Head <- Body),
    Body, not Head, !,
    component(HLit, Head),
    assume(HLit),
    not false,
    satisfy.
satisfy.
```

```
assume(X) :- asserta(X).
assume(X) :-
retract(X), !, fail.

component(E, (E ; _)).
component(E, (_ ; R)) :-
    !, component(E, R).
component(E, E).
```

Further Considerations

Choice. There have been many inference systems developed. Which one is best suited for my application?

Further Considerations

Choice. There have been many inference systems developed. Which one is best suited for my application?

Local search space. Design small inference systems that allow for as little as inferences as possible.

Further Considerations

Choice. There have been many inference systems developed. Which one is best suited for my application?

Local search space. Design small inference systems that allow for as little as inferences as possible.

Global redundancy elimination. In general there are many proofs of a given formula. Proof attempts that are “subsumed” by previous attempts should be pruned.

Further Considerations

Choice. There have been many inference systems developed. Which one is best suited for my application?

Local search space. Design small inference systems that allow for as little as inferences as possible.

Global redundancy elimination. In general there are many proofs of a given formula. Proof attempts that are "subsumed" by previous attempts should be pruned.

Efficient data structures. Determine as fast as possible the possible inferences.

Further Considerations

Choice. There have been many inference systems developed. Which one is best suited for my application?

Local search space. Design small inference systems that allow for as little as inferences as possible.

Global redundancy elimination. In general there are many proofs of a given formula. Proof attempts that are "subsumed" by previous attempts should be pruned.

Efficient data structures. Determine as fast as possible the possible inferences.

Building-in theories. Specialized reasoning procedures for "data structures", like $\mathbb{R}$, $\mathbb{Z}$, lists, arrays, sets, etc.
(These can be axiomatized, but in general this leads to nowhere.)