Theorem Proving in Relation to ...

Just one perspective to explain what theorem proving is about

**Calculation:** Compute function value at given point:

\[ 2^2 = ? \quad 3^2 = ? \quad 4^2 = ? \]

"Easy" (often polynomial)

**Constraint solving:** Find value(s) for variable(s) such that problem instance is satisfied:

- Is there \( x, y \) such that \( x^2 = 16 \)? \( x^2 = 17 \)? \( x^2 = y \)?

"Difficult" (often exponential)

**Theorem proving:** Prove a formula holds true:

\[ \forall x \text{ even}(x) \rightarrow \text{even}(x^2) \]

In general: (semi-)automatically analyse formula for logical properties

"Very difficult" (often undecidable)
Example: Three Coloring Problem

Problem: Given a map. Can it be colored using only three colors, where neighbouring countries are colored differently?

Three Coloring Problem: The Rôle of Theorem Proving

To apply theorem provers, the domain has to be formalized in logic.

**Every node has at least one color**

∀N (red(N) ∨ green(N) ∨ blue(N))

**Every node has at most one color**

∀N ((red(N) → ¬green(N)) ∧
(red(N) → ¬blue(N)) ∧
(blue(N) → ¬green(N)))

**Adjacent nodes have different color**

∀M, N (edge(M, N) → (¬(red(M) ∧ red(N)) ∧
¬(green(M) ∧ green(N)) ∧
¬(blue(M) ∧ blue(N))))

Constraint Solving: Find value(s) for variable(s) such that problem instance is satisfied

Here: Variables: Nodes in graph
Values: Red, green or blue
Problem instance: Specific graph to be colored

Constraint solving ¬ Theorem proving Prove that
general three coloring formula (see previous slide) + specific graph (as a formula)
is satisfiable

On such problems, a constraint solver is usually more efficient than a theorem prover

Other tasks where theorem proving is more appropriate?
Three Coloring Problem: The Rôle of Theorem Proving

Functional dependency

- Blue coloring depends functionally from the red and green coloring
- Blue coloring does not functionally depend from the red coloring

Theorem proving: Prove a formula holds true. Here:
Does “the blue coloring is functionally dependent from the red/red and green coloring” (as a formula) hold true?

For such general analysis tasks (wrt. all instances) theorem proving is appropriate! See Demo.

Another Application: Compiler Validation

Problem: prove equivalence of source and target program

Example:

```
1: y := 1
2: if z = x*x*x
3: then y := x*x + y
4: endif
5: y := R1+1
```

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

\[
y_1 \approx 1 \land z_0 \approx x_0 \land x_0 \land y_1 \approx x_0 \land x_0 + y_1
\]

\[
y_1' \approx 1 \land R1_2 \approx x_0' \land x_0' \land R2_3 \approx R1_2 \land x_0' \land z_0' \approx R2_3 \land y_0 \approx R1_2 + 1
\]

\[
\land x_0 \approx x_0' \land y_0 \approx y_0' \land z_0 \approx z_0' \quad \models \quad y_3 \approx y_0'
\]

Motivation

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: Specializations for Applications
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 $\delta_f$ and $\delta_g$ 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:

$$ |h(x) - h(a)| = |(f(x) + g(x)) - (f(a) + g(a))| $$

$$ = |(f(x) - f(a)) + (g(x) - g(a))| $$

$$ \leq |f(x) - f(a)| + |g(x) - g(a)| < \varepsilon/2 + \varepsilon/2 = \varepsilon $$

Predicate Logic Syntax

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

Variables $\varepsilon, a, \delta, x$

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 $\wedge, \lor, \rightarrow, \neg$

Quantifiers $\forall, \exists$

The function symbols and predicate symbols, each of given arity, comprise a signature $\Sigma$. 

The Language of Predicate Logic

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

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

Can pass this formula to a theorem prover? What does it “mean” to the prover?
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 connectives $\mapsto$ 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 $\Sigma$-algebra.

Example

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

\[
U_N = \{0, 1, 2, \ldots\} \\
0_N = 0 \\
s_N : n \mapsto n + 1 \\
+_N : (n, m) \mapsto n + m \\
* _N : (n, m) \mapsto n \times m \\
\leq_N = \{(n, m) \mid n \text{ less than or equal to } m\} \\
<_N = \{(n, m) \mid n \text{ less than } m\}
\]

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

Example

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

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

\[
\begin{align*}
\mathbb{N}(\beta)(s(x) + s(0)) &= 3 \\
\mathbb{N}(\beta)(x + y \approx s(y)) &= \text{True} \\
\mathbb{N}(\beta)(\forall x, y(x + y \approx y + x)) &= \text{True} \\
\mathbb{N}(\beta)(\forall z \ z \leq y) &= \text{False} \\
\mathbb{N}(\beta)(\forall x \exists y \ x < y) &= \text{True}
\end{align*}
\]

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

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:

\[
\begin{align*}
x + y &= y + x \\
(x + y) + z &= x + (y + z) \\
x - y &= x + (-y) \\
-(x + y) &= (-x) + (-y)
\end{align*}
\]
Axiomatizing the Real Numbers

Ordering:

\( \neg x < x \)

\( x < y \land y < z \rightarrow x < z \)

\( x \leq x \)

\( x \leq y \leftrightarrow x < y \lor x = y \)

\( x \leq y \lor y < x \)

divide by 2 and absolute values:

\( x/2 \leq 0 \rightarrow x \leq 0 \)

\( x < z/2 \land y < z/2 \rightarrow x + y < z \)

\( |x + y| \leq |x| + |y| \)

Now one can prove:

Axioms over \( \mathbb{R} \) \( \land \) continuous\((f) \land \) continuous\((g)\) \( \models \) 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 \( \beta \) such that \( A, \beta \models F \)

Solve\((F)\): find a substitution \( \sigma \) 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 \( H \models G \).

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 \( \Rightarrow \): \( \Rightarrow_p \):

\[
(F \leftrightarrow G) \Rightarrow_p (F \to G) \land (G \to F) \\
\neg QxF \Rightarrow_p Qx\neg F \\
(QxF \rho G) \Rightarrow_p Qy(F[y/x] \rho G), \ y \text{ fresh}, \ \rho \in \{\land, \lor\} \\
(QxF \to G) \Rightarrow_p Qy(F[y/x] \to G), \ y \text{ fresh} \\
(F \rho QxG) \Rightarrow_p Qy(F \rho G[y/x]), \ y \text{ fresh}, \ \rho \in \{\land, \lor, \to\} \\
\]

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

In the Example

\[
\forall \varepsilon (0 < \varepsilon \to \forall a \exists \delta (0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon))) \\
\Rightarrow_p \\
\forall \varepsilon \forall a(0 < \varepsilon \to \exists \delta (0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon))) \\
\Rightarrow_p \\
\forall \varepsilon \forall a \exists \delta (0 < \varepsilon \to 0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon)) \\
\Rightarrow_p \\
\forall \varepsilon \forall a \exists \delta (0 < \varepsilon \to \forall x (0 < \delta \land |x - a| < \delta \to |f(x) - f(a)| < \varepsilon)) \\
\Rightarrow_p \\
\forall \varepsilon \forall a \exists \delta \forall x (0 < \varepsilon \to (0 < \delta \land \forall x(|x - a| < \delta \to |f(x) - f(a)| < \varepsilon)))
\]
Skolemization

**Intuition:** replacement of $\exists y$ by a concrete choice function computing $y$ from all the arguments $y$ depends on.

**Transformation** $\Rightarrow S$

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

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

**In the Example**

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

$$\Rightarrow S$$

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

**Clausal Normal Form (Conjunctive Normal Form)**

$$(F \leftrightarrow G) \Rightarrow K (F \rightarrow G) \land (G \rightarrow F)$$

$$(F \rightarrow G) \Rightarrow K (\neg F \lor G)$$

$$\neg(F \lor G) \Rightarrow K (\neg F \land \neg G)$$

$$\neg(F \land G) \Rightarrow K (\neg F \lor \neg G)$$

$$\neg \neg F \Rightarrow K F$$

$$(F \land H) \lor (G \lor H) \Rightarrow K (F \land G)$$

$$(F \land \bot) \Rightarrow K \bot$$

$$(F \lor \bot) \Rightarrow K \top$$

$$(F \lor \bot) \Rightarrow K F$$

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.

**Skolemization**

**Together:** $F \Rightarrow_p G \Rightarrow_s H$

**prenex** **prenex, no $\exists$**

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

**In the Example**

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

$$\Rightarrow_K$$

$$0 < d(\varepsilon, a) \lor \neg(0 < \varepsilon)$$

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

**Note:** The universal quantifiers for the variables $\varepsilon, a$ and $x$, as well as the conjunction symbol $\land$ between the clauses are not written, for convenience.
The Complete Picture

\[ F \implies_{P} Q_{1}y_{1} \ldots Q_{n}y_{n} G \quad \text{(G quantifier-free)} \]
\[ \implies_{S} \forall x_{1}, \ldots, x_{m} H \quad \text{(m \leq n, H quantifier-free)} \]
\[ \implies_{K} \forall x_{1}, \ldots, x_{m} \bigwedge_{i=1}^{k} \bigvee_{j=1}^{n} L_{ij} \text{ clauses } C \]

\[ N = \{ C_{1}, \ldots, C_{k} \} \text{ is called the clausal (normal) form (CNF) of } F. \]

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

Now we arrived at clause logic and the proof problem to show that the CNF of \( F \) is unsatisfiable. That is, to show there is no interpretation that satisfies the CNF of \( F \).

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 \( \Sigma \)) is a \( \Sigma \)-algebra \( A \) such that

\[ U_{A} = T_{\Sigma} (= \text{ the set of ground terms over } \Sigma, \text{ where a ground term is a term without any variables }) \]

\[ f_{A} : (s_{1}, \ldots, s_{n}) \mapsto f(s_{1}, \ldots, s_{n}), \ f / n \in \Omega \]

\[ f_{A}(\Delta, \ldots, \Delta) = f \]

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 \( p_{A} \subseteq T_{\Sigma}^{m} \).

Proposition

Every set of ground atoms \( I \) uniquely determines a Herbrand interpretation \( A \) via

\[ (s_{1}, \ldots, s_{n}) \in p_{A} \iff p(s_{1}, \ldots, s_{n}) \in I \]

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

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

\( N \) as Herbrand interpretation over \( \Sigma_{\text{Pres}} \):

\[ I = \{ 0 \leq 0, 0 \leq s(0), 0 \leq s(s(0)), \ldots, \]

\[ 0 + 0 \leq 0, 0 + 0 \leq s(0), \ldots, \]

\[ (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0)) \]

\[ \ldots, s(0) + 0 < s(0) + 0 + 0 + s(0) \]

\[ \ldots \} \]

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 \( \Sigma \)-clauses.

\( N \) is satisfiable \( \iff \) \( N \) has a Herbrand model (over \( \Sigma \))

\( \iff \) \( G_\Sigma (N) \) has a Herbrand model (over \( \Sigma \))

where

\[ G_\Sigma (N) = \{ C \sigma \text{ ground clause} \mid C \in N, \sigma : X \rightarrow T_\Sigma \} \]

is the set of ground instances of \( N \).

Example of a \( G_\Sigma \)

For \( \Sigma_{\text{Pres}} \) one obtains for

\[ C = (x < y) \lor (y \leq s(x)) \]

the following ground instances:

\[ (0 < 0) \lor (0 \leq s(0)) \]

\[ (s(0) < 0) \lor (0 \leq s(s(0))) \]

\[ \ldots \]

\[ (s(0) + s(0) < s(0) + 0) \lor (s(0) + 0 \leq s(s(0) + s(0))) \]

\[ \ldots \]

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

Applied to clause logic:

Theorem

Let \( N \) be a set of \( \Sigma \)-clauses.

\( N \) is unsatisfiable \( \iff \) \( G_\Sigma (N) \) has no Herbrand model (over \( \Sigma \))

\( \iff \) there is a finite subset of \( G_\Sigma (N) \) that has no Herbrand model (over \( \Sigma \))

Significance: It's the core argument to show that there are complete (and sound) proof systems for first-order logic.

For instance, "Gilmore's Method".
Gilmore’s Method - Based on Herbrand’s Theorem

**Preprocessing:**

- **Given Formula:** $\forall x \exists y P(y, x) \land \forall z \neg P(z, a)$
- **Clause Form:** $P(f(x), x)$, $\neg P(z, a)$

**Outer Loop:**

- **Grounding**

**Inner Loop:**

- **Propositional Method**

**Clause Form**

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

**Given Formula**

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

**Preprocessing:**

- $\forall x \exists y P(y, x) \land \forall z \neg P(z, a)$
- $P(f(x), x)$, $\neg P(z, a)$

**Outer Loop:**

- **Grounding**

**Inner Loop:**

- **Propositional Method**

**Clause Form**

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

**Given Formula**

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

**Preprocessing:**

- $\forall x \exists y P(y, x) \land \forall z \neg P(z, a)$
- $P(f(x), x)$, $\neg P(z, a)$

**Outer Loop:**

- **Grounding**

**Inner Loop:**

- **Propositional Method**

**Clause Form**

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

**Given Formula**

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

**Preprocessing:**

- $\forall x \exists y P(y, x) \land \forall z \neg P(z, a)$
- $P(f(x), x)$, $\neg P(z, a)$

**Outer Loop:**

- **Grounding**

**Inner Loop:**

- **Propositional Method**

**Clause Form**

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

**Given Formula**

- $P(f(x), x)$
- $\neg P(z, a)$
Part III: Proof Systems - In Particular the Resolution Calculus

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. But “model computation” is gaining increasingly importance.

Inference Systems and Proofs

**Inference systems** \( \Gamma \) (proof calculi) are sets of tuples
\[
(F_1, \ldots, F_n, F_{n+1}), \ n \geq 0,
\]
called **inferences** or **inference rules**, and written

\[
\frac{F_1 \ldots F_n}{F_{n+1}}.
\]

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

Problems:
- Controlling the grounding process in **outer loop** (irrelevant instances)
- Repeat work **across** inner loops
- Weak redundancy criterion **within** inner loop
A proof in $\Gamma$ of a formula $F$ from 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 $\Gamma$, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness and Completeness

**Proposition**

1. Let $\Gamma$ be sound. Then $N \vdash F \Rightarrow N \models F$
2. $N \vdash F \Rightarrow$ there exist $F_1, \ldots, F_n \in N$ s.t. $F_1, \ldots, F_n \vdash F$

(remembers compactness).

Proofs as Trees

- **markings** $\equiv$ formulas
- **leaves** $\equiv$ assumptions and axioms
- **other nodes** $\equiv$ inferences: conclusion $\equiv$ ancestor
- **premises** $\equiv$ direct descendants

$$\begin{align*}
&\frac{P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b)}{P(f(a)) \lor Q(b) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b)}
\end{align*}$$

$$\begin{align*}
&\frac{\neg P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor Q(b)}{Q(b) \lor Q(b) \quad \neg P(f(a)) \lor \neg Q(b) \quad \neg P(f(a)) \lor \neg Q(b)}
\end{align*}$$

$$\begin{align*}
&\frac{\neg P(g(a, b)) \quad \neg P(g(a, b)) \quad \neg P(g(a, b))}{\neg P(g(a, b)) \lor \neg P(g(a, b)) \quad \neg P(g(a, b)) \lor \neg P(g(a, b)) \quad \neg P(g(a, b)) \lor \neg P(g(a, b))}
\end{align*}$$
The Resolution Calculus

Modern versions of the first-order version of the resolution calculus [Robinson 1965] are (still) the most important calculi for FOTP today.

**Propositional resolution inference rule:**

\[
\frac{C \lor A \land \neg A \lor D}{C \lor D}
\]

Terminology: \(C \lor D\): resolvent; \(A\): resolved atom

**Propositional (positive) factorisation inference rule:**

\[
\frac{C \lor A \lor A}{C \lor A}
\]

These are schematic inference rules:

\(C\) and \(D\) – propositional clauses
\(A\) – propositional atom

“\(\lor\)” is considered associative and commutative

**Sample Refutation**

1. \(\neg A \lor \neg A \lor B\) (given)
2. \(A \lor B\) (given)
3. \(\neg C \lor \neg B\) (given)
4. \(C\) (given)
5. \(\neg A \lor B \lor B\) (Res. 2. into 1.)
6. \(\neg A \lor B\) (Fact. 5.)
7. \(B \lor B\) (Res. 2. into 6.)
8. \(B\) (Fact. 7.)
9. \(\neg C\) (Res. 8. into 3.)
10. \(\bot\) (Res. 4. into 9.)

Soundness of Resolution

**Proposition**

Propositional resolution is sound.

**Proof:**

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

1. for resolution: \(I \models C \lor A, I \models D \lor \neg A \Rightarrow I \models C \lor D\)
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) \(I \models A \Rightarrow I \models D \Rightarrow I \models C \lor D\)

b) \(I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \lor D\)

Ad (ii): even simpler.

Resolution is also refutationally complete.

Methods for First-Order Clause Logic

- Gilmore’s method, see above (considered “naive” nowadays)
- The Resolution Calculus, see below

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 (not treated here)

- Tableaux and connection methods, Model Elimination
- Instance Based Methods (not here - see my home page for tutorial)
**Gilmore’s Method vs. Versus Resolution**

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

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

\[ P(f(x), x) \rightarrow \neg P(y, z) \lor Q(y, z) \]
\[ Q(f(x), x) \]

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

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

\[ P(y, z) \text{ subsumes } P(y, y) \lor Q(y, y) \]

Not available in Gilmore’s method

---

**First-Order Resolution through Instantiation**

**Idea:** instantiate clauses to ground clauses:

\[ P(z', z') \lor \neg Q(z) \]
\[ \neg P(a, y) \]
\[ P(x', b) \lor Q(f(x', x)) \]

\[ [a/z', f(a, b)/z] \]
\[ [a/y] \]
\[ [b/y] \]
\[ [a/x', b/x] \]

\[ P(a, a) \lor \neg Q(f(a, b)) \]
\[ \neg P(a, a) \]
\[ \neg P(a, b) \]
\[ P(a, b) \lor Q(f(a, b)) \]

\[ \neg Q(f(a, b)) \]
\[ Q(f(a, b)) \]

\[ \bot \]

---

**First-Order Resolution**

**Idea:** do not instantiate more than necessary:

\[ P(z', z') \lor \neg Q(z) \]
\[ \neg P(a, y) \]
\[ P(x', b) \lor Q(f(x', x)) \]

\[ [a/z'] \]
\[ [a/y] \]
\[ [b/y] \]
\[ [a/x'] \]

\[ P(a, a) \lor \neg Q(z) \]
\[ \neg P(a, a) \]
\[ \neg P(a, b) \]
\[ P(a, b) \lor Q(f(a, x)) \]

\[ \neg Q(z) \]
\[ Q(f(a, x)) \]

\[ [f(a, x)/z] \]
\[ \neg Q(f(a, x)) \]
\[ Q(f(a, x)) \]

\[ \bot \]

Bears ressemblance with Gilmore’s method - not optimal.
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.

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

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

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

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

Example

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

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

Unification

A substitution \( \sigma \) 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 \( \sigma \) is from above: \( g(x, y, z)\sigma = g(x, b, f(a, x)) \).

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

A substitution \( \sigma \) 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 $\sigma$ is called more general than a substitution $\tau$, denoted by $\sigma \leq \tau$, if there exists a substitution $\rho$ such that $\rho \circ \sigma = \tau$, where $(\rho \circ \sigma)(x) := (x\sigma)\rho$ is the composition of $\sigma$ and $\rho$ 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 $\text{mgu}(E)$.

Main Properties

If $E = x_1 \doteq u_1, \ldots, x_k \doteq u_k$, with $x_i$ pairwise distinct, $x_i \not\in \text{var}(u_j)$, 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 $\sigma_E$ is an mgu of $E$.

Theorem

1. If $E \Rightarrow_{\text{MM}} E'$ then $\sigma$ is a (most general) unifier of $E$ iff
   $\sigma$ is a (most general) unifier of $E'$
2. If $E \Rightarrow_{\text{MM}} \bot$ then $E$ is not unifiable.
3. If $E \Rightarrow_{\text{MM}} E'$ 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 $\sigma$ of $E$, such that $\sigma$ is idempotent and $\text{dom}(\sigma) \cup \text{codom}(\sigma) \subseteq \text{var}(E)$.

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.

**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.)

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):

1. 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.

---

**Part IV: Model Generation**

Scenario: no “theorem” to prove, or a non-theorem.

A model provides further information then

**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

Why compute models (cont’d)?

Natural Language Processing:

Maintain models $I_1, \ldots, I_n$ as different readings of discourses:

$\exists I \models BG\text{-Knowledge} \cup \text{Discourse so far}$

Consistency checks ("Mia’s husband loves Sally. She is not married.”)

BG-Knowledge $\cup$ Discourse so far $\not\models \neg$New utterance

eff BG-Knowledge $\cup$ Discourse so far $\cup$ New utterance is satisfiable

Informativity checks ("Mia’s husband loves Sally. She is married.”)

BG-Knowledge $\cup$ Discourse so far $\not\models$ New utterance

eff BG-Knowledge $\cup$ Discourse so far $\cup$ $\neg$New utterance is satisfiable

Example - Group Theory

The following axioms specify a group

$\forall x, y, z : (x \ast y) \ast z = x \ast (y \ast z)$ (associativity)

$\forall x : e \ast x = x$ (left − identity)

$\forall x : i(x) \ast x = e$ (left − inverse)

Does

$\forall x, y : x \ast y = y \ast x$ (commutat.)

follow?

No, it does not

Finite Model Finders - Idea

Assume a fixed domain size $n$.

Use a tool to decide if there exists a model with domain size $n$ for a given problem.

Do this starting with $n = 1$ with increasing $n$ until a model is found.

Note: domain of size $n$ will consist of $\{1, \ldots, n\}$. 

Example - Group Theory

Counterexample: a group with finite domain of size 6, where the elements 2 and 3 are not commutative: Domain: \{1, 2, 3, 4, 5, 6\}

$e : 1$

$i : \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}$

$*: \begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 2 \\
1 & 2 & 3 & 4 & 5 & 6 & 2 \\
1 & 2 & 3 & 4 & 5 & 6 & 2 \\
1 & 2 & 3 & 4 & 5 & 6 & 2 \\
1 & 2 & 3 & 4 & 5 & 6 & 2 \\
1 & 2 & 3 & 4 & 5 & 6 & 2 \\
\end{array}$

First-Order Theorem Proving – Peter Baumgartner – p.68
1. Approach: SEM-style

Tools: SEM, Finder, Mace4

Specialized constraint solvers.

For a given domain generate all ground instances of the clause.

Example: For domain size 2 and clause \( p(a, g(x)) \) the instances are \( p(a, g(1)) \) and \( p(a, g(2)) \).

2. Approach: Mace-style

Tools: Mace2, Paradox

For given domain size \( n \) transform first-order clause set into equisatisfiable propositional clause set.

Original problem has a model of domain size \( n \) iff the transformed problem is satisfiable.

Run SAT solver on transformed problem and translate model back.

Paradox - Example

Domain: \{1, 2\}

Clauses: \( \{p(a) \lor f(x) = a\} \)

Flattened: \( p(y) \lor f(x) = y \lor a \neq y \)

Instances:

\[
\begin{align*}
\text{p(1)} & \lor f(1) = 1 \lor a \neq 1 \\
\text{p(2)} & \lor f(1) = 1 \lor a \neq 2 \\
\text{p(1)} & \lor f(2) = 1 \lor a \neq 1 \\
\text{p(2)} & \lor f(2) = 1 \lor a \neq 2
\end{align*}
\]

Totality:

\[
\begin{align*}
a = 1 & \lor a = 2 \\
f(1) = 1 & \lor f(1) = 2 \\
f(2) = 1 & \lor f(2) = 2
\end{align*}
\]

Functionality:

\[
\begin{align*}
a \neq 1 & \lor a \neq 2 \\
f(1) \neq 1 & \lor f(1) \neq 2 \\
f(2) \neq 1 & \lor f(2) \neq 2
\end{align*}
\]

A model is obtained by setting the blue literals true.
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.)