First-Order Theorem Proving

Peter Baumgartner

Logic and Computation Program

Peter.Baumdartner@nicta.com.au

Slides partially based on material by Alexander Fuchs, Harald Ganzinger, John Slaney, Viorica Sofronie-Stockermans and Uwe Waldmann
Contents

- Part I: Motivation
- Part II: Predicate Logic, from the viewpoint of First-Order Theorem Proving (FOTP)
- Part III: Proof Systems, in Particular the Resolution Calculus
- Part IV: Model Generation
Part I - Motivation
Theorem Proving in Relation to ...

Just one perspective to explain what theorem proving is about
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)
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 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:

Does \( \forall x \text{ even}(x) \rightarrow \text{even}(x^2) \) hold true?

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 - Graph Theory Abstraction

Problem Instance

Problem Specification
Three Coloring Problem - Graph Theory Abstraction

Problem Instance

Problem Specification

The Rôle of Theorem Proving?
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

\[ \forall N \, (\text{red}(N) \lor \text{green}(N) \lor \text{blue}(N)) \]
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**

\[ \forall N (\text{red}(N) \lor \text{green}(N) \lor \text{blue}(N)) \]

**Every node has at most one color**

\[ \forall N ((\text{red}(N) \rightarrow \neg \text{green}(N)) \land
    (\text{red}(N) \rightarrow \neg \text{blue}(N)) \land
    (\text{blue}(N) \rightarrow \neg \text{green}(N))) \]
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

\[ \forall N \ (\text{red}(N) \lor \text{green}(N) \lor \text{blue}(N)) \]

Every node has at most one color

\[ \forall N \ ((\text{red}(N) \rightarrow \neg \text{green}(N)) \land \neg \text{blue}(N)) \land \neg \text{green}(N)) \]

Adjacent nodes have different color

\[ \forall M, N \ (\text{edge}(M, N) \rightarrow \neg (\text{red}(M) \land \text{red}(N)) \land \neg (\text{green}(M) \land \text{green}(N)) \land \neg (\text{blue}(M) \land \text{blue}(N)))) \]
Three Coloring Problem: The Rôle of Theorem Proving

**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  $\rightsquigarrow$ Theorem proving** Prove that general three coloring formula (see previous slide) + specific graph (as a formula)

is satisfiable
Three Coloring Problem: The Rôle of Theorem Proving

**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** $\rightsquigarrow$ **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 dependence

Blue coloring depends functionally from the red and green coloring
Functional dependancy

- Blue coloring depends functionally from the red and green coloring

- Blue coloring does not functionally depend from the red coloring
Three Coloring Problem: The Rôle of Theorem Proving

Functional dependancy

- 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?
Three Coloring Problem: The Rôle of Theorem Proving

Functional dependancy

- Blue coloring depends functionally from the red and green coloring
  - ![Diagram showing blue coloring depending on red and green]

- Blue coloring does not functionally depend from the red coloring
  - ![Diagram showing blue coloring not depending on red]

**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 \( z = x \times x \times x \)
3: then \( y := x \times x + y \)
4: endif

1: \( y := 1 \)
2: \( R1 := x \times x \)
3: \( R2 := R1 \times x \)
4: jmpNE \((z, R2, 6)\)
5: \( y := R1 + 1 \)

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

\[
\begin{align*}
y_1 & \approx 1 \land z_0 \approx x_0 \times x_0 \times x_0 \land y_3 \approx x_0 \times x_0 + y_1 \\
y_1' & \approx 1 \land R1_2 \approx x_0' \times x_0' \land R2_3 \approx R1_2 \times x_0' \land z_0' \approx R2_3 \land y_5' \approx R1_2 + 1 \\
& \land x_0 \approx x_0' \land y_0 \approx y_0' \land z_0 \approx z_0' \end{align*}
\]

\[\vdash y_3 \approx y_5'\]
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, …
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
Motivation

In this lecture, 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
Part II – Predicate Logic, from the viewpoint of FOTP

- Syntax and semantics of first-order predicate logic: a mathematical example
- 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 \( \delta_f \) 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:

\[
|h(x) - h(a)| = |f(x) + g(x) - f(a) - g(a)| \\
= |(f(x) - f(a)) + (g(x) - g(a))| \\
\leq |f(x) - f(a)| + |g(x) - g(a)| < \varepsilon/2 + \varepsilon/2 = \varepsilon
\]
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 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  \( \land, \lor, \rightarrow, \neg \)

Quantifiers  \( \forall, \exists \)

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

\[
\begin{align*}
U_N &= \{0, 1, 2, \ldots\} \\
0_N &= 0 \\
\text{s}_N &: \quad n \mapsto n + 1 \\
+_{N} &: \quad (n, m) \mapsto n + m \\
\times_{N} &: \quad (n, m) \mapsto n \times m \\
\leq_{N} &= \{(n, m) \mid n \text{ less than or equal to } m\} \\
<_{N} &= \{(n, m) \mid n \text{ less than } m\}
\end{align*}
\]

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

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

If $\phi$ is a closed formula, then, instead of $I(\phi) = True$ 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:

\[ \neg x < x \]

\[ x < y \land y < z \implies x < z \]

\[ x \leq x \]

\[ x \leq y \iff x < y \lor x = y \]

\[ x \leq y \lor y < x \]

divide by 2 and absolute values:

\[ x/2 \leq 0 \implies x \leq 0 \]

\[ x < z/2 \land y < z/2 \implies x + y < z \]

\[ |x + y| \leq |x| + |y| \]
Now one can prove:

\[
\text{Axioms over } \mathbb{R} \land \text{continuous}(f) \land \text{continuous}(g) \models \text{continuous}(f + g)
\]
Now one can prove:

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

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 $\beta$ 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 $H \models G$. 
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. $H \land \neg G$ is unsatisfiable.
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 $\Rightarrow_P$:

\[
(F \leftrightarrow G) \Rightarrow_P (F \rightarrow G) \land (G \rightarrow F)
\]

\[
\neg Q \times F \Rightarrow_P \overline{Q} \times \neg F
\]

\[
(Q \times F \rho G) \Rightarrow_P Qy(F[y/x] \rho G), \ y \text{ fresh}, \ \rho \in \{\land, \lor\}
\]

\[
(Q \times F \rightarrow G) \Rightarrow_P \overline{Q}y(F[y/x] \rightarrow G), \ y \text{ fresh}
\]

\[
(F \rho Q \times G) \Rightarrow_P Qy(F \rho G[y/x]), \ y \text{ fresh}, \ \rho \in \{\land, \lor, \rightarrow\}
\]

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

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

\[ \Rightarrow P \]

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

\[ \Rightarrow P \]

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

\[ \Rightarrow P \]

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

\[ \Rightarrow P \]

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

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

Transformation $\Rightarrow_S$

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

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

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

$\Rightarrow_S$

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

Together: \( F \Rightarrow_P G \Rightarrow_S H \)

\( \text{prenex} \quad \text{prenex, no } \exists \)

Theorem: The given and the final formula are equi-satisfiable.
Clausal Normal Form (Conjunctive Normal Form)

\[
(F \leftrightarrow G) \Rightarrow_K (F \rightarrow G) \land (G \rightarrow F)
\]
\[
(F \rightarrow G) \Rightarrow_K (\neg F \lor G)
\]
\[
\neg (F \lor G) \Rightarrow_K (\neg F \land \neg G)
\]
\[
\neg (F \land G) \Rightarrow_K (\neg F \lor \neg G)
\]
\[
\neg \neg F \Rightarrow_K F
\]
\[
(F \land G) \lor H \Rightarrow_K (F \lor H) \land (G \lor H)
\]
\[
(F \land \top) \Rightarrow_K F
\]
\[
(F \land \bot) \Rightarrow_K \bot
\]
\[
(F \lor \top) \Rightarrow_K \top
\]
\[
(F \lor \bot) \Rightarrow_K F
\]

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 \textbf{negation normal form} (NNF) of a formula.
In the Example

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

\[ \Rightarrow K \]

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

**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

\[ F \Rightarrow^*_P Q_1 y_1 \ldots Q_n y_n G \]  
\[ \Rightarrow^*_S \forall x_1, \ldots, x_m H \]  
\[ \Rightarrow^*_K \forall x_1, \ldots, x_m \bigwedge_{i=1}^{k} \bigvee_{j=1}^{n_i} L_{ij} \]

leave out \( i \leq n \), \( H \) quantifier-free

clauses \( C_i \)

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

**Note:** the variables in the clauses are implicitly universally quantified.
The Complete Picture

\[ F \Rightarrow^*_P Q_1 y_1 \ldots Q_n y_n G \quad (G \text{ quantifier-free}) \]

\[ \Rightarrow^*_S \forall x_1, \ldots, x_m H \quad (m \leq n, H \text{ quantifier-free}) \]

\[ \Rightarrow^*_K \forall x_1, \ldots, x_m \left( \bigwedge_{i=1}^{k} \bigvee_{j=1}^{n_i} L_{ij} \right) \quad \text{(leaves out clauses } C_i) \]

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

**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 \( F \).
Some thoughts

Suppose we want to prove $H \models G$. 
Some thoughts

Suppose we want to prove \( H \models G \).

Equivalently, we can prove that \( F := H \land \neg G \) is unsatisfiable.
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.
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 $\Sigma$) is a $\Sigma$-algebra $\mathcal{A}$ such that

- $U_{\mathcal{A}} = T_\Sigma$ (= the set of ground terms over $\Sigma$, where a **ground term** is a term without any variables)
- $f_{\mathcal{A}} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n), \ f/n \in \Omega$

![Diagram](attachment:image.png)
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 $p_A \subseteq T^m$. 

**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: $\Sigma_{\text{Pres}} = \{0/0, s/1, +/2\}, \{</2, \leq/2\}$

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

$I = \{0 \leq 0, 0 \leq s(0), 0 \leq s(s(0)), \ldots,$

$0 + 0 \leq 0, 0 + 0 \leq s(0), \ldots,$

$\ldots, (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0))$

$\ldots$

$s(0) + 0 < s(0) + 0 + 0 + s(0)$

$\ldots\}$
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 \( \Sigma_{Pres} \) 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)
∀φ is unsatisfiable iff some finite set of ground instances \(\{\phi_1, \ldots, \phi_n\}\) 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 \( \mathcal{N} \) be a set of \( \Sigma \)-clauses.

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

\( \iff \) there is a \textbf{finite} subset of \( G_\Sigma(\mathcal{N}) \)

that has no Herbrand model (over \( \Sigma \))
Herbrand’s Theorem

**Theorem (Skolem-Herbrand-Theorem)**
∀φ 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
\[ P(f(x), x) \land \neg P(z, a) \]

Outer loop:
Grounding

Inner loop:
Propositional Method
Gilmore’s Method - Based on Herbrand’s Theorem

Preprocessing:
\[
\forall x \ \exists y \ P(y, x) \\
\land \forall z \ \neg P(z, a)
\]

Clause Form:
\[
P(f(x), x) \\
\neg P(z, a)
\]

Outer loop: Grounding

Inner loop: Propositional Method

First-Order Theorem Proving – Peter Baumgartner – p.41
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
\[ P(f(x), x) \land \neg P(z, a) \]

Outer loop:
Grounding

Inner loop:
Propositional Method

Sat?

No
STOP: Proof found

Yes
Continue Outer Loop

First-Order Theorem Proving – Peter Baumgartner – p.41
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
\[ P(f(a), a) \]
\[ \neg P(a, a) \]

**Inner loop:**
Propositional Method
\[ P(f(a), a) \]
\[ \neg P(a, a) \]
\[ \neg P(f(a), a) \]
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

\[ P(f(a), a) \]
\[ \neg P(a, a) \]

**Inner loop:**

Propositional Method

\[ P(f(a), a) \]
\[ \neg P(a, a) \]
\[ \neg P(f(a), a) \]

**Sat?**

- **No**
  - STOP: Proof found
  - Continue Outer Loop

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

Outer loop:
- Grounding
  $P(f(a), a) \land \neg P(a, a)$
  $\vdots$
  $P(f(a), a) \land \neg P(a, a) \land \neg P(f(a), a)$

Inner loop:
- Propositional Method

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

---

First-Order Theorem Proving – Peter Baumgartner – p.41
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.
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.)
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

\[
\begin{array}{c}
\text{premises} \\
F_1 \ldots F_n \\
\hline
F_{n+1}
\end{array}
\]

\text{conclusion}

Clausal inference system: premises and conclusions are clauses. One also considers inference systems over other data structures.
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 $1 \leq j \leq n_i$. 

---

**First-Order Theorem Proving** – Peter Baumgartner – p.44
Soundness and Completeness

**Provability** $\vdash_{\Gamma}$ of $F$ from $N$ in $\Gamma$:

$N \vdash_{\Gamma} F :\iff$ there exists a proof $\Gamma$ of $F$ from $N$.

$\Gamma$ is called **sound** : $\iff$

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

$\Gamma$ is called **complete** : $\iff$

$N \models F \Rightarrow N \vdash_{\Gamma} F$

$\Gamma$ is called **refutationally complete** : $\iff$

$N \models \bot \Rightarrow N \vdash_{\Gamma} \bot$
Soundness and Completeness

Proposition

1. Let $\Gamma$ 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. $F_1, \ldots, F_n \vdash_{\Gamma} F$

(reminds compactness).
Proofs as Trees

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

\[
P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b) \\
\hline
\neg P(f(a)) \lor Q(b) \lor Q(b) \\
P(f(a)) \lor Q(b) \\
\hline
\neg P(f(a)) \lor Q(b) \\
Q(b) \lor Q(b) \\
\hline
Q(b) \\
\hline
\neg P(g(a, b)) \\
\hline
\bot
\]
The Resolution Calculus $\text{Res}$

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:**

\[
\begin{array}{c}
C \lor A \\
\neg A \lor D
\end{array}
\Rightarrow
\begin{array}{c}
C \lor D
\end{array}
\]

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

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

\[
\begin{array}{c}
C \lor A \lor A
\end{array}
\Rightarrow
\begin{array}{c}
C \lor A
\end{array}
\]

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)
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.)
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.)
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.)
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.)
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.)
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: $I \models C \lor A \lor A \Rightarrow I \models C \lor A$
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 $I$. 
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 \( I \in \Sigma\text{-Alg.} \). 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) \( I \models \neg A \Rightarrow I \models C \Rightarrow I \models C \lor D \)
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*
Gilmore’s Method vs. Versus Resolution

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

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

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

\[ Q(f(x), x) \]
**Gilmore’s Method vs. Versus Resolution**

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

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

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

One inference may represent infinitely many propositional resolution inferences ("lifting principle")
Gilmore’s Method vs. Versus Resolution

Central Point: Resolution performs intrinsic first-order reasoning

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

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

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

Redundancy concepts, e.g. subsumption deletion:

\[
P(y, z) \quad \text{subsumes} \quad P(y, y) \lor Q(y, y)
\]
Gilmore’s Method vs. Versus Resolution

Central Point: Resolution performs intrinsic first-order reasoning

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

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

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

Redundancy concepts, e.g. subsumption deletion:

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

Not available in Gilmore’s method
First-Order Resolution through Instantiation

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

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

\[
[\text{[a/z', f(a, b)/z]}] \quad [\text{[a/y]}] \quad [\text{[b/y]}] \quad [\text{[a/x', b/x]}]
\]

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

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

\[
\bot
\]

Bears resemblance with Gilmore’s method - not optimal.
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:

\[
\begin{align*}
P(z', z') & \lor \neg Q(z) & \neg P(a, y) & P(x', b) \lor Q(f(x', x)) \\
[a/z'] & & [a/y] & [b/y] & [a/x'] \\
P(a, a) & \lor \neg Q(z) & \neg P(a, a) & \neg P(a, b) & P(a, b) \lor Q(f(a, x)) \\
& & \neg Q(z) & & Q(f(a, x)) \\
& [f(a, x)/z] & & & \\
& \neg Q(f(a, x)) & & & \\
& & Q(f(a, x)) & & \\
& & & & \bot
\end{align*}
\]
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

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

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

In both cases, \(A\) and \(B\) have to be renamed apart (made variable disjoint).
Resolution for First-Order Clauses

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

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

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

**Example**

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

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

A **substitution** $\sigma$ is a mapping from variables to terms which is the identity almost everywhere.
Example: $\sigma = [f(a, x)/z, b/y]$
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: $g(x, y, z)\sigma = g(x, b, f(a, x))$. 
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 $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ ($s_i, t_i$ 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 \( \text{mgu}(E) \).
Unification after Martelli/Montanari

\[
\begin{align*}
t & \doteq t, E \quad \Rightarrow_{MM} \quad E \\
f(s_1, \ldots, s_n) & \doteq f(t_1, \ldots, t_n), E \quad \Rightarrow_{MM} \quad s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) & \doteq g(\ldots), E \quad \Rightarrow_{MM} \quad \bot \\
x & \doteq t, E \quad \Rightarrow_{MM} \quad x \doteq t, E[t/x] \\
& \quad \text{if } x \in \text{var}(E), x \not\in \text{var}(t) \\
x & \doteq t, E \quad \Rightarrow_{MM} \quad \bot \\
& \quad \text{if } x \neq t, x \in \text{var}(t) \\
t & \doteq x, E \quad \Rightarrow_{MM} \quad x \doteq t, E \\
& \quad \text{if } t \not\in X
\end{align*}
\]
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 am mgu of $E$. 
Main Properties

Theorem

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

Theorem

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

2. If $E \Rightarrow_{MM}^* \perp$ then $E$ is not unifiable.
Main Properties

Theorem

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

Theorem

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

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

usable list

set of support
The “Given Clause Loop” - Graphically

- Given clause
- Usable list
- Set of support
The “Given Clause Loop” - Graphically

First-Order Theorem Proving – Peter Baumgartner – p.66
The “Given Clause Loop” - Graphically

- given clause
- usable list
- set of support
- consequences
- filters

First-Order Theorem Proving – Peter Baumgartner – p.66
The “Given Clause Loop” - Graphically
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 $J_1, \ldots, J_n$ as different readings of discourses:

$$J_i \models BG\text{-}Knowledge \cup Discourse_{so\_far}$$
Part IV: Model Generation

Why compute models (cont’d)?

Natural Language Processing:

- Maintain models $J_1, \ldots, J_n$ as different readings of discourses:
  
  $$J_i \models BG\text{-Knowledge} \cup Discourse\_so\_far$$

- Consistency checks (“Mia’s husband loves Sally. She is not married.”)
  
  $$BG\text{-Knowledge} \cup Discourse\_so\_far \not\models \neg New\_utterance$$

  iff  
  
  $$BG\text{-Knowledge} \cup Discourse\_so\_far \cup New\_utterance \text{ is satisfiable}$$
Part IV: Model Generation

Why compute models (cont’d)?

Natural Language Processing:

- Maintain models $J_1, \ldots, J_n$ as different readings of discourses:
  \[ J_i \models BG\text{-Knowledge} \cup Discourse\_so\_far \]

- Consistency checks ("Mia’s husband loves Sally. She is not married.")
  \[ BG\text{-Knowledge} \cup Discourse\_so\_far \not\models \neg New\_utterance \]
  iff \[ BG\text{-Knowledge} \cup Discourse\_so\_far \cup \neg New\_utterance \text{ is satisfiable} \]

- Informativity checks ("Mia’s husband loves Sally. She is married.")
  \[ BG\text{-Knowledge} \cup Discourse\_so\_far \not\models New\_utterance \]
  iff \[ BG\text{-Knowledge} \cup Discourse\_so\_far \cup \neg New\_utterance \text{ is satisfiable} \]
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?
Example - Group Theory

The following axioms specify a group

\[ \forall x, y, z : (x * y) * z = x * (y * z) \quad \text{(associativity)} \]
\[ \forall x : e * x = x \quad \text{(left – identity)} \]
\[ \forall x : i(x) * x = e \quad \text{(left – inverse)} \]

Does

\[ \forall x, y : x * y = y * x \quad \text{(commutat.)} \]

follow?

No, it does not
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\}

\[e : 1\]

\[i : \]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 5 & 4 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

\[\ast : \]

\[
\begin{array}{cccccc}
3 & 3 & 5 & 1 & 6 & 2 \\
4 & 4 & 6 & 2 & 5 & 1 \\
5 & 5 & 3 & 6 & 1 & 4 \\
6 & 6 & 4 & 5 & 2 & 3 \\
\end{array}
\]
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 $\{1, \ldots, n\}$.
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 $p(a, g(2))$. 
1. Approach: SEM-style

- Set up multiplication tables for all symbols with the whole domain as cell values.

- Example: For domain size 2 and function symbol $g$ with arity 1 the cells are $g(1) = \{1, 2\}$ and $g(2) = \{1, 2\}$.

- Try to restrict each cell to exactly 1 value.

- The clauses are the constraints guiding the search and propagation.

- Example: if the cell of $a$ contains $\{1\}$, the clause $a = b$ forces the cell of $b$ to be $\{1\}$ as well.
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:
- \( p(1) \lor f(1) = 1 \lor a \neq 1 \)
- \( p(2) \lor f(1) = 1 \lor a \neq 2 \)
- \( p(1) \lor f(2) = 1 \lor a \neq 1 \)
- \( p(2) \lor f(2) = 1 \lor a \neq 2 \)

Totality:
- \( a = 1 \lor a = 2 \)
- \( f(1) = 1 \lor f(1) = 2 \)
- \( f(2) = 1 \lor f(2) = 2 \)

Functionality:
- \( a \neq 1 \lor a \neq 2 \)
- \( f(1) \neq 1 \lor f(1) \neq 2 \)
- \( f(2) \neq 1 \lor f(2) \neq 2 \)

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?
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 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 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.)