First-Order Theorem Proving

Peter Baumgartner

NICTA

Peter.Baumgartner@nicta.com.au
http://users.rsise.anu.edu.au/~baumgart/

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

Purpose of This Lecture

Provide an overview about FOTP:

“What” Part
- Automatically analyzing problems stated in first-order logic
- Context: other disciplines in Automated Deduction

“How” Part - Important Techniques
- Normal forms of formulas
- Herbrand interpretations
- Resolution calculus, unification
- Instance-based method
- Model computation

Context: First-Order Theorem Proving in Relation to ...

… Calculation: Compute function value at given point:

Problem: \( 2^2 = ? \), \( 3^2 = ? \), \( 4^2 = ? \)

“Easy” (often polynomial)

… Constraint Solving: Given:

- Problem: \( x^2 = a \) where \( x \in [1 \ldots b] \) (\( x \) variable, \( a, b \) parameters)
- Instance: \( a = 16, b = 10 \)

Find values for variables such that problem instance is satisfied

“Difficult” (often exponential, but restriction to finite domains)

First-Order Theorem Proving: Given:

- Problem: \( \exists x (x^2 = a \land x \in [1 \ldots b]) \)
- Is it satisfiable? unsatisfiable? valid?
- “Very difficult” (often undecidable)

Logical Analysis 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**

![Graph with nodes and edges]

**Problem Specification**

![Diagram illustrating problem specification]

The Rôle of Theorem Proving?

First-Order Theorem Proving – Peter Baumgartner – p.5

---

Three Coloring Problem - Formalization

*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*

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

*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 - Solving Problem Instances ...

... with a constraint solver:

Let constraint solver find value(s) for variable(s) such that problem instance is satisfied

**Here:** Variables: Colors of nodes in graph
Values: Red, green or blue
Problem instance: Specific graph to be colored

... with a theorem prover

Let the theorem prover prove that the three coloring formula (see previous slide) + specific graph (as a formula) is satisfiable

- To solve problem instances a constraint solver is usually much more efficient than a theorem prover (e.g. use a SAT solver)
- Theorem provers are not even guaranteed to terminate, in general

Other tasks where theorem proving is more appropriate?

Three Coloring Problem: The Rôle of Theorem Proving

**Functional dependency**

- Blue coloring depends functionally on the red and green coloring

![Diagram illustrating dependency]

- Blue coloring does not functionally depend on the red coloring

![Diagram illustrating non-dependency]

Theorem proving: Prove a formula is valid. Here:

Is "the blue coloring is functionally dependent on the red/red and green coloring" (as a formula) valid, i.e. holds for all possible graphs?

I.e. analysis wrt. all instances ⇒ theorem proving is adequate

Theorem Prover Demo

First-Order Theorem Proving – Peter Baumgartner – p.6
How to Build a (First-Order) Theorem Prover

1. Fix an input language for formulas
2. Fix a semantics to define what the formulas mean
   Will be always “classical” here
3. Determine the desired services from the theorem prover
   (The questions we would like the prover be able to answer)
4. Design a calculus for the logic and the services
   Calculus: high-level description of the “logical analysis” algorithm
   This includes redundancy criteria for formulas and inferences
5. Prove the calculus is correct (sound and complete) wrt. the logic and
   the services, if possible
6. Design a proof procedure for the calculus
7. Implement the proof procedure (research topic of its own)

Go through the red issues in the rest of this talk

Languages and Services — Propositional SAT

<table>
<thead>
<tr>
<th>Formula(s)</th>
<th>Theorem Prover</th>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional logic formula $\phi$</td>
<td></td>
<td>Is $\phi$ satisfiable?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Minimal model? Maximal consistent subsets?)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Based on BDD, DPLL, or stochastic local search</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>See lecture by Anbulagan on methods for SAT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Languages and Services — Description Logics

<table>
<thead>
<tr>
<th>Formula(s)</th>
<th>Theorem Prover</th>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description Logic TBox + ABox (restricted FOL)</td>
<td></td>
<td>Is TBox + ABox satisfiable?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professor $\sqcap$ $\exists$ supervises. Student $\sqsubset$ BusyPerson</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p : $ Professor $(p, s) :$ supervises</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>See lecture by Prof. Baader on Description Logics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Languages and Services — Satisfiability Modulo Theories (SMT)**

**Formula:** Usually variable-free first-order logic formula \( \phi \)
- Equality \( \equiv \), combination of theories, free symbols

**Question:** Is \( \phi \) valid? (satisfiable? entailed by another formula?)
- \[ \models_{\text{NUL}} \forall c (c = 5 \rightarrow \text{car}(\text{cons}(3 + c, l)) = 8) \]

**Theorem Prover:** DPLL(T), translation into SAT, first-order provers

**Issue:** essentially undecidable for non-variable free fragment

\[ P(0) \land (\forall x P(x) \rightarrow P(x + 1)) \models_{\text{N}} \forall x P(x) \]

Design a “good” prover anyways (ongoing research)

---

**Languages and Services — “Full” First-Order Logic**

**Formula:** First-order logic formula \( \phi \) (e.g. the three-coloring spec above)
- Usually with equality \( \equiv \)

**Question:** Is \( \phi \) formula valid? (satisfiable?, entailed by another formula?)

**Theorem Prover:** Superposition (Resolution), Instance-based methods

**Issues**
- Efficient treatment of equality
- Decision procedure for sub-languages or useful reductions?
  - Can do e.g. DL reasoning? Model checking? Logic programming?
- Built-in inference rules for arrays, lists, arithmetics (still open research)

---

**How to Build a (First-Order) Theorem Prover**

1. Fix an **input language** for formulas
2. Fix a **semantics** to define what the formulas mean
   - Will be always “classical” here
3. Determine the desired **services** from the theorem prover
   - (The questions we would like the prover be able to answer)
4. Design a **calculus** for the logic and the services
   - Calculus: high-level description of the “logical analysis” algorithm
   - This includes redundancy criteria for formulas and inferences
5. Prove the calculus is **correct** (sound and complete) wrt. the logic and the services, if possible
6. Design a **proof procedure** for the calculus
7. Implement the proof procedure (research topic of its own)

---

**Semantics**

“The function \( f \) is continuous”, expressed in (first-order) predicate logic:

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

**Underlying Language**

**Variables** \( \varepsilon, a, \delta, x \)

**Function symbols** \( 0, |\cdot|, \cdot - \cdot, f(\cdot) \)

**Terms** are well-formed expressions over variables and function symbols

**Predicate symbols** \( \cdot < \cdot, \cdot = \cdot \)

**Atoms** are applications of predicate symbols to terms

**Boolean connectives** \( \land, \lor, \rightarrow, \neg \)

**Quantifiers** \( \forall, \exists \)

The function symbols and predicate symbols comprise a signature \( \Sigma \)
Semantics

“The function \( f \) is continuous”, expressed in (first-order) predicate logic:

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

“Meaning” of Language Elements – \( \Sigma \)-Algebras

Universe (aka Domain): Set \( U \)
Variables \( \rightarrow \) values in \( U \) (mapping is called “assignment”)
Function symbols \( \rightarrow \) (total) functions over \( U \)
Predicate symbols \( \rightarrow \) relations over \( U \)
Boolean connectives \( \rightarrow \) the usual boolean functions
Quantifiers \( \rightarrow \) “for all ... holds”, “there is a ..., such that”
Terms \( \rightarrow \) values in \( U \)
Formulas \( \rightarrow \) Boolean (Truth-) values

Semantics - \( \Sigma \)-Algebra Example

Let \( \Sigma_{PA} \) be the standard signature of Peano Arithmetic
The standard interpretation \( \mathbb{N} \) for Peano Arithmetic then is:

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

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

Evaluation of terms and formulas

Under the interpretation \( \mathbb{N} \) and the assignment \( \beta : x \mapsto 1, y \mapsto 3 \) we obtain

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

Important Basic Notion: Model

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

\[
I \models \phi \quad \text{ ("} I \text{ is a model of } \phi \text{")}
\]

E.g. \( I \models \forall x \exists y \ x < y \)

Standard reasoning services can now be expressed semantically

Services Semantically

E.g. “entailment”:

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

Services

Model(\( I, \phi \)):

\[
I \models \phi ? \ (\text{Is } I \text{ a model for } \phi?)
\]

Validity(\( \phi \)):

\[
\models \phi ? \ (I \models \phi \text{ for every interpretation?})
\]

Satisfiability(\( \phi \)):

\[
\phi \text{ satisfiable? } (I \models \phi \text{ for some interpretation?)}
\]

Entailment(\( \phi, \psi \)):

\[
\phi \models \psi \ ? \ (\text{does } \phi \text{ entail } \psi \text{?, i.e. for every interpretation } I: \text{if } I \models \phi \text{ then } I \models \psi?)
\]

Solve(\( I, \phi \)):

\[
\text{find an assignment } \beta \text{ such that } I, \beta \models \phi
\]

Solve(\( \phi \)):

\[
\text{find an interpretation and assignment } \beta \text{ such that } I, \beta \models \phi
\]

Additional complication: fix interpretation of some symbols (as in \( \mathbb{N} \) above)

What if theorem prover’s native service is only “Is \( \phi \) unsatisfiable?”?
Semantics - Reduction to Unsatisfiability

Suppose we want to prove an entailment \( \phi \models \psi \).

Equivalently, prove \( \models \phi \rightarrow \psi \), i.e., that \( \phi \rightarrow \psi \) is valid.

Equivalently, prove that \( \neg(\phi \rightarrow \psi) \) is not satisfiable (unsatisfiable).

Equivalently, prove that \( \phi \land \neg \psi \) is unsatisfiable.

Basis for (predominant) refutational theorem proving

Dual problem, much harder: to disprove an entailment \( \phi \models \psi \) find a model of \( \phi \land \neg \psi \).

One motivation for (finite) model generation procedures

First-Order Theorem Proving – Peter Baumgartner – p.21

How to Build a (First-Order) Theorem Prover

1. Fix an input language for formulas
2. Fix a semantics to define what the formulas mean
   Will be always “classical” here
3. Determine the desired services from the theorem prover
   (The questions we would like the prover be able to answer)
4. Design a calculus for the logic and the services
   Calculus: high-level description of the “logical analysis” algorithm
   This includes redundancy criteria for formulas and inferences
5. Prove the calculus is correct (sound and complete) wrt. the logic and the services, if possible
6. Design a proof procedure for the calculus
7. Implement the proof procedure (research topic of its own)

Calculus - Normal Forms

Most first-order theorem provers take formulas in clause normal form

Why Normal Forms?
- Reduction of logical concepts (operators, quantifiers)
- Reduction of syntactical structure (nesting of subformulas)
- Can be exploited for efficient data structures and control

Translation into Clause Normal Form

Prop: the given formula and its clause normal form are equi-satisfiable

Prenex Normal Form

Prenex formulas have the form
\[ Q_1 x_1 \ldots Q_n x_n F, \]
where \( F \) is quantifier-free and \( Q_i \in \{\forall, \exists\} \).

Computing prenex-free and \( Q_i \in \{\forall, \exists\} \)

Here \( Q \) denotes the quantifier dual to \( Q \), i.e., \( \forall = \exists \) and \( \exists = \forall \).
In the Example

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

\[ \Rightarrow \rho \]

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

\[ \Rightarrow \rho \]

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

\[ \Rightarrow \rho \]

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

\[ \Rightarrow \rho \]

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

Clausal Normal Form (Conjunctive Normal Form)

Rules to convert the matrix of the formula in Skolem normal form into a conjunction of disjunctions:

\[ (F \leftrightarrow G) \Rightarrow K (F \rightarrow G) \land (G \rightarrow F) \]

\[ (F \rightarrow G) \Rightarrow K (\neg F \lor G) \land (\neg F \land G) \]

\[ (F \land G) \lor H \Rightarrow K (F \land H) \lor (G \lor H) \]

\[ (F \lor \top) \Rightarrow K \top \]

\[ (F \lor \bot) \Rightarrow K \bot \]

They are to be applied modulo associativity and commutativity of \( \land \) and \( \lor \)

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

Herbrand interpretation

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

- The universe is the set $T_\Sigma$ of ground terms over $\Sigma$ (a ground term is a term without any variables):
  $$U_\mathcal{A} = T_\Sigma$$
- Every function symbol from $\Sigma$ is "mapped to itself":
  $$f_\mathcal{A} : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n), \text{ where } f \text{ is } n\text{-ary function symbol in } \Sigma$$

Example

- $\Sigma_{\text{Pres}} = \{0/0, s/1, +/2\}$
- $U_\mathcal{A} = \{0, s(0), s(s(0)), \ldots, 0 + 0, s(0) + 0, \ldots, s(0) + 0, 0 + 0\}$
- $0 \mapsto 0, s(0) \mapsto s(0), s(s(0)) \mapsto s(s(0)), \ldots, 0 + 0 \mapsto 0 + 0, \ldots$

Herbrand Interpretations

Only interpretations $\rho_\mathcal{A}$ of predicate symbols $\rho \in \Sigma$ is undetermined in a Herbrand interpretation

- $\rho_\mathcal{A}$ represented as the set of ground atoms

$$\{p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \in \rho_\mathcal{A} \text{ where } p \in \Sigma \text{ is } n\text{-ary predicate symbol}\}$$

Example

- $\Sigma_{\text{Pres}} = \{0/0, s/1, +/2\}$ (from above)
- $N$ as Herbrand interpretation over $\Sigma_{\text{Pres}}$
- $I = \{0 \leq 0, 0 \leq s(0), 0 \leq s(s(0)), \ldots, 0 + 0 \leq 0, 0 + 0 \leq s(0), \ldots, \ldots, (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0)), \ldots\}$

Herbrand's Theorem

Proposition

A Skolem normal form $\forall \phi$ is unsatisfiable iff it has no Herbrand model

Theorem (Skolem-Herbrand-Theorem)

$\forall \phi$ has no Herbrand model iff some finite set of ground instances

$$\{\phi \gamma_1, \ldots, \phi \gamma_n\} \text{ is unsatisfiable}$$

Applied to clause logic:

Theorem (Skolem-Herbrand-Theorem)

A set $N$ of $\Sigma$-clauses is unsatisfiable iff some finite set of ground instances of clauses from $N$ is unsatisfiable

Leads immediately to theorem prover "Gilmore's Method"
Gilmore’s Method - Based on Herbrand’s Theorem

**Preprocessing:**

Given Formula: $\forall x \exists y P(y, x)$

Clause Form: $P(f(x), x)$

**Outer loop:**

Grounding

**Inner loop:**

Propositional Method

**Given Formula:**

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

$\neg P(a, a)$

$P(f(x), x)$

$P(f(a), a)$

$\neg P(a, a)$

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

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

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

First-Order Theorem Proving – Peter Baumgartner – p.33

Gilmore’s Method - Based on Herbrand’s Theorem

**Preprocessing:**

Given Formula: $\forall x \exists y P(y, x)$

Clause Form: $P(f(x), x)$

**Outer loop:**

Grounding

**Inner loop:**

Propositional Method

**Given Formula:**

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

$\neg P(a, a)$

$P(f(x), x)$

$P(f(a), a)$

$\neg P(a, a)$

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

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

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

First-Order Theorem Proving – Peter Baumgartner – p.35
Gilmore’s Method - Based on Herbrand’s Theorem

Outer Loop Proof found

Grounding Propositional Method

STOP:

¬ \( P(f(a), a) \)

∧ ∀ \( z \) ¬ \( P(z, a) \)

Given Formula

\( \forall x \exists y P(y, x) \)

\( \land \forall z \neg P(z, a) \)

\( P(f(x), x) \)

\( \neg P(z, a) \)

\( P(f(a), a) \)

\( \neg P(a, a) \)

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

Clause Form

Preprocessing:

Outer loop: Grounding

Inner loop: Propositional Method

Sat?

No

Yes

STOP:

Proof found

Continue Outer Loop

First-Order Theorem Proving – Peter Baumgartner – p.37

Calculi for First-Order Logic Theorem Proving

- Gilmore’s method reduces proof search in first-order logic to propositional logic unsatisfiability problems
- Main problem is the unguided generation of (very many) ground clauses
- All modern calculi address this problem in one way or another, e.g.
  - Guidance: Instance-Based Methods are similar to Gilmore’s method but generate ground instances in a guided way
  - Avoidance: Resolution calculi need not generate the ground instances at all
    Resolution inferences operate directly on clauses, not on their ground instances

Next: propositional Resolution, lifting, first-order Resolution

The Propositional Resolution Calculus \( Res \)

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:

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

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 Proof

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.)

First-Order Theorem Proving – Peter Baumgartner – p.39
Soundness of Propositional 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

Completeness of Propositional Resolution

Theorem:
Propositional Resolution is refutationally complete

That is, if a propositional 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: semantic tree proof technique (see blackboard)

This result can be considerably strengthened, some strengthenings come for free from the proof

Propositional resolution is not suitable for first-order clause sets

Lifting Propositional Resolution to First-Order Resolution

Propositional resolution

<table>
<thead>
<tr>
<th>Clauses</th>
<th>Ground instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P(f(x), y) )</td>
<td>( {P(f(a), \ldots, P(f(f(a)), f(f(a))), \ldots} )</td>
</tr>
<tr>
<td>( \neg P(z, z) )</td>
<td>( {\neg P(a), \ldots, \neg P(f(a)), f(f(a))}\ldots)</td>
</tr>
</tbody>
</table>

Only common instances of \( P(f(x), y) \) and \( P(z, z) \) give rise to inference:

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

Unification

All common instances of \( P(f(x), y) \) and \( P(z, z) \) are instances of \( P(f(x), f(x)) \)

\( P(f(x), f(x)) \) is computed deterministically by unification

First-order resolution

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

Justified by existence of \( P(f(x), f(x)) \)

Can represent infinitely many propositional resolution inferences

Substitutions and Unifiers

A substitution \( \sigma \) is a mapping from variables to terms which is the identity almost everywhere

Example: \( \sigma = [y \mapsto f(x), \ z \mapsto f(x)] \)

A substitution can be applied to a term or atom \( t \), written as \( t\sigma \)

Example, where \( \sigma \) is from above: \( P(f(x), y)\sigma = P(f(x), f(x)) \)

A substitution \( \gamma \) is a unifier of \( s \) and \( t \) iff \( s\gamma = t\gamma \)

Example: \( \gamma = [x \mapsto a, y \mapsto f(a), \ z \mapsto f(a)] \) is a unifier of \( P(f(x), y) \) and \( P(z, z) \)

A unifier \( \sigma \) of \( s \) is most general iff for every unifier \( \gamma \) of \( s \) and \( t \) there is a substitution \( \delta \) such that \( \gamma = \sigma \circ \delta \); notation: \( \sigma = \text{mgu}(s, t) \)

Example: \( \sigma = [y \mapsto f(x), \ z \mapsto f(x)] = \text{mgu}(P(f(x), y), P(z, z)) \)

There are (linear) algorithms to compute mgus' or return “fail”
Resolution for First-Order Clauses

\[
\begin{align*}
C \lor A & \quad D \lor \neg B \\
(C \lor D) & \quad \text{if } \sigma = \text{mgu}(A, B) \quad \text{[resolution]} \\
C \lor A \lor B & \quad (C \lor A) \quad \text{if } \sigma = \text{mgu}(A, B) \quad \text{[factorization]}
\end{align*}
\]

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

Example

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

\[
Q(z) \lor P(z, a) \lor P(a, y) \\
Q(a) \lor P(a, a) \quad \text{where } \sigma = [z \mapsto a, y \mapsto a] \quad \text{[factorization]}
\]
Calculi for First-Order Logic Theorem Proving

Recall:
- Gilmore’s method reduces proof search in first-order logic to propositional logic unsatisfiability problems
- Main problem is the unguided generation of (very many) ground clauses
- All modern calculi address this problem in one way or another, e.g.
  - **Guidance:** Instance-Based Methods are similar to Gilmore’s method but generate ground instances in a guided way
  - **Avoidance:** Resolution calculi need not generate the ground instances at all
- Resolution inferences operate directly on clauses, not on their ground instances

Next: Instance-Based Method “Inst-Gen”

Inst-Gen [Ganzinger&Korovin 2003]

**Idea:** “semantic” guidance: add only instances that are falsified by a “candidate model”

Eventually, all repairs will be made or there is no more candidate model

**Important notation:** \( \perp \) denotes both a unique constant and a substitution that maps every variable to \( \perp \)

Example (\( S \) is “current clause set”):

\[
S : \quad \begin{align*}
P(x, y) \lor P(y, x) \\
\neg P(x, x)
\end{align*}
\]

\[
S \perp : \quad \begin{align*}
P(\perp, \perp) \lor P(\perp, \perp) \\
\neg P(\perp, \perp)
\end{align*}
\]

Analyze \( S \perp \):

Case 1: SAT detects unsatisfiability of \( S \perp \)
Then Conclude \( S \) is unsatisfiable

**But what if \( S \perp \) is satisfied by some model, denoted by \( I_S \)?
Inst-Gen - Model Construction

It provides (partial) interpretation for \( S_{\text{ground}} \) for given clause set \( S \)

\[
S : \quad P(x) \lor Q(x) \quad \Sigma = \{a, b\}, \quad S_{\text{ground}} : \quad P(b) \lor Q(b) \\
Q(a) \lor P(a) \quad P(a) \lor Q(a) \\
\neg P(a) \quad \neg P(a)
\]

For each \( C_{\text{ground}} \in S_{\text{ground}} \) find most specific \( C \in S \) that can be instantiated to \( C_{\text{ground}} \)

Select literal in \( C_{\text{ground}} \) corresponding to selected literal in that \( C \)

Add selected literal of that \( C_{\text{ground}} \) to \( I_S \) if not in conflict with \( I_S \)

Thus, \( I_S = \{P(b), Q(a), \neg P(a)\} \)

Model Generation

Why compute models (cont’d)?

Natural Language Processing:
- Maintain models \( J_1, \ldots, J_n \) as different readings of discourses:
  \( J_i \models BG\text{-Knowledge} \cup \text{Discourse so far} \)

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\}

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

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 \( p(a, g(2)) \).

- 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

<table>
<thead>
<tr>
<th>Domain: {1, 2}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clauses: {p(a) \lor f(x) = a}</td>
</tr>
<tr>
<td>Flattened: ( p(y) \lor f(x) = y \lor a \neq y )</td>
</tr>
<tr>
<td>Instances:</td>
</tr>
<tr>
<td>( p(1) \lor f(1) = 1 \lor a \neq 1 )</td>
</tr>
<tr>
<td>( p(2) \lor f(1) = 1 \lor a \neq 2 )</td>
</tr>
<tr>
<td>( p(1) \lor f(2) = 1 \lor a \neq 1 )</td>
</tr>
<tr>
<td>( p(2) \lor f(2) = 1 \lor a \neq 2 )</td>
</tr>
<tr>
<td>Totality:</td>
</tr>
<tr>
<td>( a = 1 \lor a = 2 )</td>
</tr>
<tr>
<td>( f(1) = 1 \lor f(1) = 2 )</td>
</tr>
<tr>
<td>( f(2) = 1 \lor f(2) = 2 )</td>
</tr>
<tr>
<td>Functionality:</td>
</tr>
<tr>
<td>( a \neq 1 \lor a \neq 2 )</td>
</tr>
<tr>
<td>( f(1) \neq 1 \lor f(1) \neq 2 )</td>
</tr>
<tr>
<td>( f(2) \neq 1 \lor f(2) \neq 2 )</td>
</tr>
</tbody>
</table>

A model is obtained by setting the blue literals true

Conclusions

- Talked about the role of First-Order Theorem proving
- Talked about some standard techniques (Normal forms of formulas, Resolution calculus, unification, Instance-based method, Model computation)

Further Topics

- Resolution variants and other calculi, e.g. Model Elimination
- Redundancy elimination, efficient equality reasoning, adding arithmetics (open problem)
- FOTP methods as decision procedures in special cases
  E.g. reducing planning problems and temporal logic model checking problems to function-free clause logic and using an instance-based method as a decision procedure
- Implementation techniques
- Competition CASC and TPTP problem library
- Instance-based methods (a lot to do here, cf. my home page)
  Attractive because of complementary features to more established methods