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: $2^2 = ?$ $\quad 3^2 = ?$ $\quad 4^2 = ?$

"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)

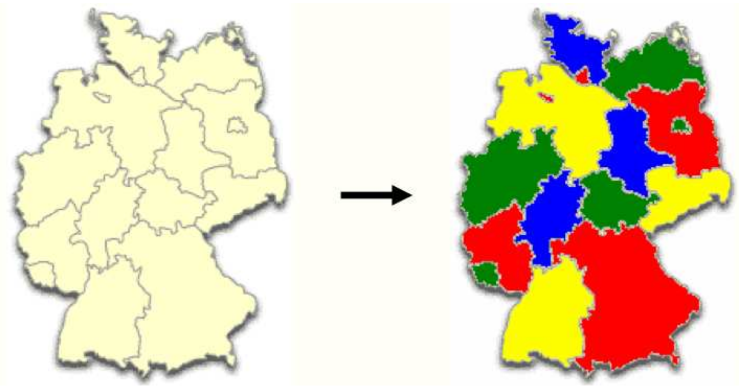
First-Order Theorem Proving: Given:

Problem: $\exists x\, (x^2 = a \land x \in [1 \ldots b])$

Is it satisfiable? unsatisfiable? valid?

"Very difficult" (often undecidable)

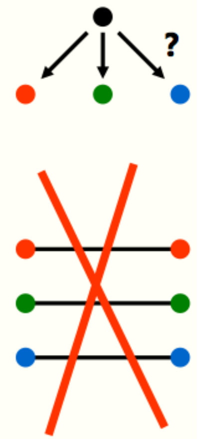
Logical Analysis Example: Three Coloring Problem



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 **Problem Specification**



The Rôle of Theorem Proving?

Three Coloring Problem - Formalization

Every node has at least one color

$$\forall N\, (\mathrm{red}(N) \lor \mathrm{green}(N) \lor \mathrm{blue}(N))$$

Every node has at most one color

$$\begin{aligned}\forall N\, ((\mathrm{red}(N) \rightarrow \neg \mathrm{green}(N)) \land \\ (\mathrm{red}(N) \rightarrow \neg \mathrm{blue}(N)) \land \\ (\mathrm{blue}(N) \rightarrow \neg \mathrm{green}(N)))\end{aligned}$$

Adjacent nodes have different color

$$\begin{aligned}\forall M, N\, (\mathrm{edge}(M, N) \rightarrow (&\neg(\mathrm{red}(M) \land \mathrm{red}(N)) \land \\ &\neg(\mathrm{green}(M) \land \mathrm{green}(N)) \land \\ &\neg(\mathrm{blue}(M) \land \mathrm{blue}(N))))\end{aligned}$$

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

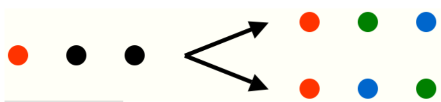
Other tasks where theorem proving is more appropriate?

Three Coloring Problem: The Rôle of Theorem Proving

Functional dependency

- Blue coloring depends functionally on the red and green coloring

- Blue coloring does not functionally depend on the red coloring



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 $\Rightarrow$ theorem proving is adequate

Theorem Prover Demo

How to Build a (First-Order) Theorem Prover

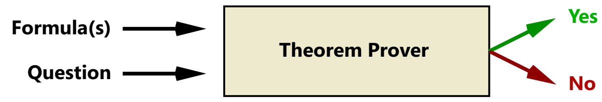
1. Fix an **input language** for formulas
2. Fix a **semantics** to define what the formulas mean
   Will be always "classical" here
3. Determine the desired **services** from the theorem prover
   (The questions we would like the prover be able to answer)
4. Design a **calculus** for the logic and the 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

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)

Languages and Services — Propositional SAT



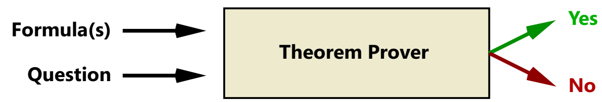
Formula: Propositional logic formula ϕ

Question: Is ϕ satisfiable?
(Minimal model? Maximal consistent subsets?)

Theorem Prover: Based on BDD, DPLL, or stochastic local search

Issue: the formula ϕ can be **BIG**
See lecture by Anbulagan on methods for SAT

Languages and Services — Description Logics



Formula: Description Logic TBox + ABox (restricted FOL)

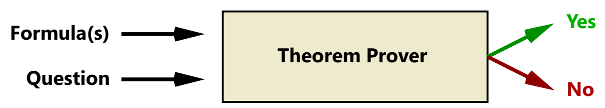
TBox: Terminology	Professor $\sqcap \exists$ supervises . Student $\sqsubseteq$ BusyPerson
ABox: Assertions	p : Professor $\quad (p, s)$: supervises

Question: Is TBox + ABox satisfiable?
(Does C subsume D?, Concept hierarchy?)

Theorem Prover: Tableaux algorithms (predominantly)

Issue: Push expressivity of DLs while preserving decidability
See lecture by Prof. Baader on Description Logics

Languages and Services — Satisfiability Modulo Theories (SMT)



Formula: Usually **variable-free** first-order logic formula ϕ
Equality $\doteq$, combination of theories, free symbols

Question: Is ϕ valid? (satisfiable? entailed by another formula?)

$$\models_{\mathbb{N}\cup\mathbb{L}} \forall l\ (c = 5 \to \mathrm{car}(\mathrm{cons}(3 + c, l)) \doteq 8)$$

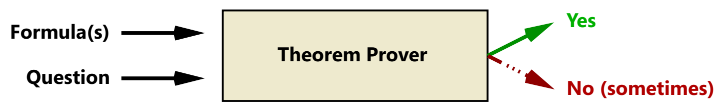
Theorem Prover: DPLL(T), translation into SAT, first-order provers

Issue: essentially undecidable for non-variable free fragment

$$P(0) \wedge (\forall x\ P(x) \to P(x + 1)) \models_{\mathbb{N}} \forall x\ P(x)$$

Design a "good" prover anyways (ongoing research)

Languages and Services — "Full" First-Order Logic



Formula: First-order logic formula ϕ (e.g. the three-coloring spec above)
Usually with equality $\doteq$

Question: Is ϕ 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:

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

Underlying Language

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 $\wedge, \vee, \to, \neg$

Quantifiers $\forall, \exists$

The function symbols and predicate symbols comprise a signature Σ

Semantics

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

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

"Meaning" of Language Elements – Σ-Algebras

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

Semantics - Σ-Algebra Example

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

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

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

Semantics - Σ-Algebra Example

Evaluation of terms and formulas

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

$$\begin{array}{lcll} (\mathbb{N}, \beta)(s(x) + s(0)) & = & 3 & \\ (\mathbb{N}, \beta)(x + y \doteq s(y)) & = & \mathit{True} & \\ (\mathbb{N}, \beta)(\forall z\, z \leq y) & = & \mathit{False} & \\ (\mathbb{N}, \beta)(\forall x \exists y\, x < y) & = & \mathit{True} & \\ \mathbb{N}(\forall x \exists y\, x < y) & = & \mathit{True} & \text{(Short notation when } \beta \text{ irrelevant)} \end{array}$$

Important Basic Notion: Model

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

$$I \models \phi \qquad \text{("}I\text{ is a model of }\phi\text{")}$$

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

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,ϕ): $I \models \phi$? (Is I a model for ϕ?)

Validity(ϕ): $\models \phi$? ($I \models \phi$ for every interpretation?)

Satisfiability(ϕ): ϕ satisfiable? ($I \models \phi$ for some interpretation?)

Entailment(ϕ,ψ): $\phi \models \psi$? (does ϕ entail ψ?, i.e.
for every interpretation I: if $I \models \phi$ then $I \models \psi$?)

Solve(I,ϕ): find an assignment β such that $I, \beta \models \phi$

Solve(ϕ): find an interpretation and assignment β 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 ϕ 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 $\phi \wedge \neg\psi$ is unsatisfiable

Basis for (predominant) refutational theorem proving

Dual problem, much harder: to disprove an entailment $\phi \models \psi$ find a model of $\phi \wedge \neg\psi$

One motivation for (finite) model generation procedures

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

Formula → **Prenex normal form** → **Skolem normal form** → **Clause normal form** → **Clausal Theorem Prover**

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 normal form by the rewrite relation $\Rightarrow_P$:

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

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

In the Example

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

$$\Rightarrow_P$$

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

$$\Rightarrow_P$$

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

$$\Rightarrow_P$$

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

$$\Rightarrow_P$$

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

Skolem Normal Form

Formula → **Prenex normal form** → **Skolem normal form** → **Clause normal form** → **Clausal Theorem Prover**

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

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

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

In the Example

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

$$\Rightarrow_S$$

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

Clausal Normal Form (Conjunctive Normal Form)

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

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

They are to be applied modulo associativity and commutativity of $\wedge$ and $\vee$

In the Example

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

$$\Rightarrow_K$$

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

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

The Complete Picture

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

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

Note: the variables in the clauses are implicitly universally quantified

Instead of showing that F is unsatisfiable, the proof problem from now is to show that N is unsatisfiable

Can do better than "searching through all interpretations"

Theorem: N is satisfiable iff it has a Herbrand model

Herbrand Interpretations

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

- The universe is the set T_Σ of ground terms over Σ
(a **ground term** is a term without any variables):

$$U_{\mathcal{A}} = T_\Sigma$$

- Every function symbol from Σ 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

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

Herbrand Interpretations

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

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

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

- Whole interpretation represented as $\bigcup_{p \in \Sigma} p_{\mathcal{A}}$

Example

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

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

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

Applied to clause logic:
Theorem (Skolem-Herbrand-Theorem)
A set N of Σ-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

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

Clause Form

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

Outer loop:
Grounding

Inner loop:
Propositional
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

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

Inner loop:
Propositional
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

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

Inner loop:
Propositional
Method

Sat?

No

STOP:
Proof found

Yes

Continue
Outer Loop

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

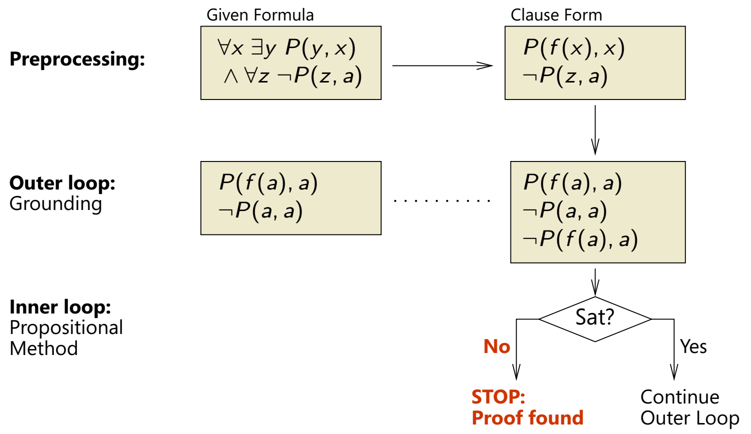
$P(f(a),a)$
$\neg P(a,a)$

. . . .

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

Inner loop:
Propositional
Method

Gilmore's Method - Based on Herbrand's Theorem



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:

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

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

Propositional (positive) factorisation inference rule:

$$\frac{C \vee A \vee A}{C \vee A}$$

These are **schematic inference rules**:

C and D – propositional clauses

A – propositional atom

"$\vee$" is considered associative and commutative

Sample Proof

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

Soundness of Propositional Resolution

Proposition
Propositional resolution is sound

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

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

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

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

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

Ad (ii): even simpler

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

Clauses	Ground instances
$P(f(x), y)$	$\{P(f(a), a), \ldots, P(f(f(a)), f(f(a))), \ldots\}$
$\neg P(z, z)$	$\{\neg P(a), \ldots, \neg P(f(f(a)), f(f(a))), \ldots\}$

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

$$\frac{P(f(f(a)), f(f(a))) \qquad \neg P(f(f(a)), f(f(a)))}{\perp}$$

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

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

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

Can represent infinitely many propositional resolution inferences

Substitutions and Unifiers

- A **substitution** σ 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 σ is from above: $P(f(x), y)\sigma = P(f(x), f(x))$
- A substitution γ 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 σ of s is **most general** iff for every unifier γ of s and t there is a substitution δ 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 mgu's or return "fail"

Resolution for First-Order Clauses

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

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

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

Example

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

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

Completeness of First-Order Resolution

Theorem: Resolution is **refutationally complete**

- That is, if a clause set is unsatisfiable, then Resolution will derive the empty clause $\perp$ eventually
- More precisely: If a clause set is unsatisfiable and closed under the application of the Resolution and Factorization inference rules, then it contains the empty clause $\perp$
- Perhaps easiest proof: Herbrand Theorem + completeness of propositional resolution + Lifting Theorem (see blackboard)

  Lifting Theorem: the conclusion of any propositional inference on ground instances of first-order clauses can be obtained by instantiating the conclusion of a first-order inference on the first-order clauses
- Closure can be achieved by the "Given Clause Loop"

The "Given Clause Loop"

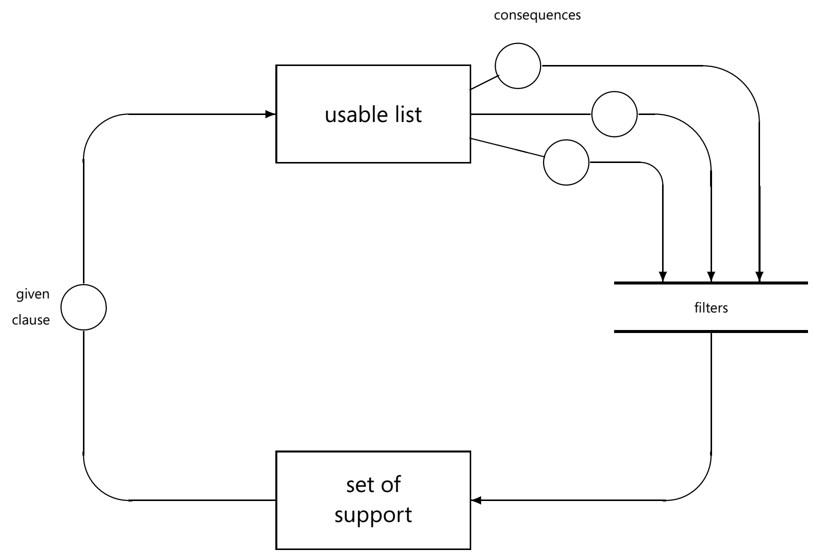
As used in the Otter theorem prover:
Lists of clauses maintained by the algorithm: `usable` and `sos`.
Initialize `sos` with the input clauses, `usable` empty.

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



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: $\bot$ denotes both a unique constant and a substitution that maps every variable to $\bot$

Example (S is "current clause set"):

$$S: \quad P(x,y) \lor P(y,x) \qquad\qquad S\bot: \quad P(\bot,\bot) \lor P(\bot,\bot)$$
$$\neg P(x,x) \qquad\qquad\qquad\qquad \neg P(\bot,\bot)$$

Analyze $S\bot$:

Case 1: SAT detects unsatisfiability of $S\bot$
Then Conclude S is unsatisfiable

But what if $S\bot$ is satisfied by some model, denoted by $I_\bot$?

Inst-Gen

Main idea: associate to model $I_\bot$ of $S\bot$ a **candidate model** I_S of S.
Calculus goal: add instances to S so that I_S becomes a model of S

Example:

$$S: \quad \underline{P(x)} \lor Q(x) \qquad\qquad S\bot: \quad \underline{P(\bot)} \lor Q(\bot)$$
$$\underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\bot$:

Case 2: SAT detects model $I_\bot = \{P(\bot), \neg P(a)\}$ of $S\bot$

Case 2.1: candidate model $I_S = \{\neg P(a)\}$ derived from
literals <u>selected</u> in S by $I_\bot$ is not a model of S

Add "problematic" instance $P(a) \lor Q(a)$ to S to refine I_S

Inst-Gen

Clause set after adding $P(a) \lor Q(a)$

$$S: \quad \underline{P(x)} \lor Q(x) \qquad\qquad S\bot: \quad \underline{P(\bot)} \lor Q(\bot)$$
$$P(a) \lor \underline{Q(a)} \qquad\qquad\qquad P(a) \lor \underline{Q(a)}$$
$$\underline{\neg P(a)} \qquad\qquad\qquad\qquad \underline{\neg P(a)}$$

Analyze $S\bot$:

Case 2: SAT detects model $I_\bot = \{P(\bot), Q(a), \neg P(a)\}$ of $S\bot$

Case 2.2: candidate model $I_S = \{Q(a), \neg P(a)\}$ derived from
literals <u>selected</u> in S by $I_\bot$ **is** a model of S

Then conclude S is satisfiable

How to derive candidate model I_S?

Inst-Gen - Model Construction

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

$$S: \quad \underline{P(x)} \vee Q(x) \qquad\qquad \Sigma = \{a, b\},\ S_{\text{ground}}: \quad \underline{P(b)} \vee Q(b)$$
$$P(a) \vee \underline{Q(a)} \qquad\qquad P(a) \vee \underline{Q(a)}$$
$$\underline{\neg P(a)} \qquad\qquad \underline{\neg P(a)}$$

- For each $C_{\text{ground}} \in S_{\text{ground}}$ find most specific $C \in S$ that can be instantiated to C_{ground}
- Select literal in C_{ground} corresponding to selected literal in that C
- Add selected literal of that C_{ground} to I_S if not in conflict with I_S

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

Model Generation

Scenario: no "theorem" to prove, or disprove a "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]

Model Generation

Why compute models (cont'd)?

Natural Language Processing:

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

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

Example - Group Theory

The following axioms specify a group

$$\begin{array}{llcll}
\forall x, y, z & : (x * y) * z & = & x * (y * z) & \text{(associativity)} \\
\forall x & : e * x & = & x & \text{(left} - \text{identity)} \\
\forall x & : i(x) * x & = & e & \text{(left} - \text{inverse)}
\end{array}$$

Does

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

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: $\{1, 2, 3, 4, 5, 6\}$

$e : 1$

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

$$* : \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 2 & 1 & 4 & 3 & 6 & 5 \\ 3 & 3 & 5 & 1 & 6 & 2 & 4 \\ 4 & 4 & 6 & 2 & 5 & 1 & 3 \\ 5 & 5 & 3 & 6 & 1 & 4 & 2 \\ 6 & 6 & 4 & 5 & 2 & 3 & 1 \end{array}$$

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

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

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