Automated Theorem Proving

Peter Baumgartner

NICTA

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

Overview of Automated Theorem Proving (ATP)

- Emphasis on automated proof methods for first-order logic
- More “breadth” than “depth”

Standard techniques covered

- Normal forms of formulas
- Herbrand interpretations
- Resolution calculus, unification
- Instance-based methods
- Model computation
- Theory reasoning: Satisfiability Modulo Theories
Part 1: What is Automated Theorem Proving?
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 \(x \in [1\ldots b]\)

\((x\) \text{ variable, } a, b \text{ 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)
**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

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

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?
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 ⇒ theorem proving is adequate
Part 2: Methods in Automated Theorem Proving
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)
Formula: Propositional logic formula $\phi$

Question: Is $\phi$ satisfiable?
   (Minimal model? Maximal consistent subsets? )

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

Issue: the formula $\phi$ can be BIG
DPLL as a Semantic Tree Method

(1) $A \lor B$   (2) $C \lor \neg A$   (3) $D \lor \neg C \lor \neg A$   (4) $\neg D \lor \neg B$

\langle\text{empty tree}\rangle

\{$\}$ $\not\models A \lor B$
\{$\}$ $\models C \lor \neg A$
\{$\}$ $\models D \lor \neg C \lor \neg A$
\{$\}$ $\models \neg D \lor \neg B$

A Branch stands for an interpretation

**Purpose of splitting:** satisfy a clause that is currently falsified

Close branch if some clause is plainly falsified by it (*)
DPLL as a Semantic Tree Method

(1) $A \lor B$  
(2) $C \lor \neg A$  
(3) $D \lor \neg C \lor \neg A$  
(4) $\neg D \lor \neg B$

A Branch stands for an interpretation

**Purpose of splitting:** satisfy a clause that is currently falsified

Close branch if some clause is plainly falsified by it ($\star$)
DPLL as a Semantic Tree Method

(1) $A \lor B$

(2) $C \lor \neg A$

(3) $D \lor \neg C \lor \neg A$

(4) $\neg D \lor \neg B$

A Branch stands for an interpretation

Purpose of splitting: satisfy a clause that is currently falsified

Close branch if some clause is plainly falsified by it ($\star$)
DPLL as a Semantic Tree Method

(1) \(A \lor B\)  (2) \(C \lor \neg A\)  (3) \(D \lor \neg C \lor \neg A\)  (4) \(\neg D \lor \neg B\)

A Branch stands for an interpretation

**Purpose of splitting:** satisfy a clause that is currently falsified

Close branch if some clause is plainly falsified by it (⋆)

Model \(\{A, C, D\}\) found.
DPLL as a Semantic Tree Method

(1) \(A \lor B\)  
(2) \(C \lor \neg A\)  
(3) \(D \lor \neg C \lor \neg A\)  
(4) \(\neg D \lor \neg B\)

A Branch stands for an interpretation

\textbf{Purpose of splitting:} satisfy a clause that is currently falsified

Close branch if some clause is plainly falsified by it (*)

DPLL is the basis of most efficient SAT solvers today

Model \(\{B\}\) found.
Languages and Services — Description Logics

Formula: Description Logic $\text{TBox} + \text{ABox}$ (restricted FOL)

- **TBox:** Terminology
  
  \[
  \text{Professor} \sqcap \exists \text{supervises} . \text{Student} \sqsubseteq \text{BusyPerson}
  \]

- **ABox:** Assertions
  
  \[
  p : \text{Professor} \quad (p, s) : \text{supervises}
  \]

Question: Is $\text{TBox} + \text{ABox}$ satisfiable?

(Does $C$ subsume $D$?, Concept hierarchy?)

Theorem Prover: Tableaux algorithms (predominantly)

Issue: Push expressivity of DLs while preserving decidability

See overview lecture by Maurice Pagnucco on “Knowledge Representation and Reasoning”
Formula: Usually **variable-free** first-order logic formula $\phi$

Equality $\doteq$, combination of theories, free symbols

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

$$\models_{\mathbb{NUL}} \forall l \ (c = 5 \rightarrow \text{car(cons}(3 + c, l)) \doteq 8)$$

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

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

$$P(0) \land (\forall x \ P(x) \rightarrow P(x + 1)) \models_{\mathbb{N}} \forall x \ P(x)$$

Design a “good” prover anyways (ongoing research)
Languages and Services — “Full” First-Order Logic

Formula: First-order logic formula $\phi$ (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)
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 $0, |., -. , f(.)$

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 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 $\mapsto$ 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 - $\Sigma$-Algebra Example

Let $\Sigma_{PA}$ be the standard signature of Peano Arithmetic.

The standard interpretation $\mathbb{N}$ for Peano Arithmetic then is:

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

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

Evaluation of terms and formulas

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

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

Important Basic Notion: Model

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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>What if theorem prover’s native service is only “Is $\phi$ unsatisfiable”?</th>
</tr>
</thead>
</table>
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**
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

![Diagram of translation process]

**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 \( \Rightarrow_P \):

\[
\begin{align*}
(F \iff G) & \Rightarrow_P (F \implies G) \land (G \implies F) \\
\neg Q x F & \Rightarrow_P \overline{Q} x \neg F \quad (\neg Q) \\
(Q x F \, \rho \, G) & \Rightarrow_P \overline{Q} y (F[y/x] \, \rho \, G), \text{ y fresh, } \rho \in \{ \land, \lor \} \\
(Q x F \implies G) & \Rightarrow_P \overline{Q} y (F[y/x] \implies G), \text{ y fresh} \\
(F \, \rho \, Q x G) & \Rightarrow_P \overline{Q} y (F \, \rho \, G[y/x]), \text{ y fresh, } \rho \in \{ \land, \lor, \implies \}
\end{align*}
\]

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

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

\[ \Rightarrow P \]

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

\[ \Rightarrow P \]

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

\[ \Rightarrow P \]

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

\[ \Rightarrow P \]

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

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

$\Rightarrow_S$

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

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

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

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
\[
F \Rightarrow^*_P Q_1 y_1 \ldots Q_n y_n \ G \quad \text{\(G\) quantifier-free)
\]
\[
\Rightarrow^*_S \forall x_1, \ldots, x_m \ H \quad \text{\(m \leq n, \ H\) quantifier-free}
\]
\[
\Rightarrow^*_K \forall x_1, \ldots, x_m \ \left( \bigwedge_{i=1}^{k} \bigvee_{j=1}^{n_i} L_{ij} \right) \quad \text{leave out clauses } C_i
\]

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

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

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 $\Sigma$) 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):

  $$U_A = T_\Sigma$$

- Every function symbol from $\Sigma$ is “mapped to itself”:

  $$f_A : (s_1, \ldots, s_n) \mapsto f(s_1, \ldots, s_n), \text{ where } f \text{ is } n\text{-ary function symbol in } \Sigma$$

**Example**

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

  $$U_A = \{0, s(0), s(s(0)), \ldots, 0 + 0, s(0) + 0, \ldots, s(0 + 0), s(s(0) + 0), \ldots\}$$

  $$0 \mapsto 0, s(0) \mapsto s(0), s(s(0)) \mapsto s(s(0)), \ldots, 0 + 0 \mapsto 0 + 0, \ldots$$
Herbrand Interpretations

Only interpretations \( p_A \) of predicate symbols \( p \in \Sigma \) is undetermined in a Herbrand interpretation

- \( p_A \) represented as the set of ground atoms

\[ \{ p(s_1, \ldots, s_n) \mid (s_1, \ldots, s_n) \in p_A \text{ where } p \in \Sigma \text{ is } n\text{-ary predicate symbol} \} \]

- Whole interpretation represented as \( \bigcup_{p \in \Sigma} p_A \)

Example

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

\[
I = \{ 0 \leq 0, 0 \leq s(0), 0 \leq s(s(0)), \ldots, \\
0 + 0 \leq 0, 0 + 0 \leq s(0), \ldots, \\
\ldots, (s(0) + 0) + s(0) \leq s(0) + (s(0) + s(0)), \ldots \}
\]
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
$\{\phi \gamma_1, \ldots, \phi \gamma_n\}$ 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
\[ \forall x \exists y \ P(y, x) \land \forall z \neg P(z, a) \]

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:**

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

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

**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

\[ P(f(x), x) \land \neg P(z, a) \]

Outer loop:

Grounding

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

Inner loop:

Propositional Method

Sat?

No

STOP: Proof found

Yes

Continue Outer Loop

Automated Theorem Proving – Peter Baumgartner – p.42
Gilmore’s Method - Based on Herbrand’s Theorem

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

Given Formula

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

**Outer loop:**

Grounding

Inner loop:

Propositional Method

**Inner loop:**

Grounding

**Outer loop:**

Propositional Method

**Given Formula:**
\[ P(f(a), a) \land \neg P(a, a) \]

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

**Sat?**

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

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

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

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

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

These are **schematic inference rules**:

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

“\( \lor \)” is considered associative and commutative
Sample 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.)
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>${ P(f(a), a), \ldots, P(f(f(a)), f(f(a))), \ldots }$</td>
</tr>
<tr>
<td>$\neg P(z, z)$</td>
<td>${ \neg P(a), \ldots, \neg P(f(f(a)), f(f(a))), \ldots }$</td>
</tr>
</tbody>
</table>

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

$$
\frac{P(f(f(a)), f(f(a))) \quad \neg P(f(f(a)), f(f(a)))}{\bot}
$$

**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**

$$
\frac{P(f(x), y) \quad \neg P(z, z)}{\bot}
$$

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 mgu’s or return “fail”
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

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

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

Given clause → usable list → consequences → filters

Set of support
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”):

\[
S : \quad P(x, y) \lor P(y, x) \\
\neg P(x, x) \\
S \bot : \quad P(\bot, \bot) \lor P(\bot, \bot) \\
\neg P(\bot, \bot)
\]

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 \( /\bot \)?**
**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:

\[
S: \quad P(x) \lor Q(x) \quad \quad S_\bot: \quad P(\bot) \lor Q(\bot) \\
\neg P(a) \quad \quad \quad \quad \quad \neg P(a)
\]

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 selected in $S$ by $I_\bot$ is not a model of $S$

Add “problematic” instance $P(a) \lor Q(a)$ to $S$ to refine $I_S$
Clause set after adding $P(a) \lor Q(a)$

$$S : \quad P(x) \lor Q(x) \quad S \bot : \quad P(\bot) \lor Q(\bot)$$

$$P(a) \lor Q(a) \quad P(a) \lor Q(a)$$

$$\neg P(a) \quad \neg P(a)$$

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 selected 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 $S_{\text{ground}}$ for given clause set $S$

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

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

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 $I_1, \ldots, I_n$ as different readings of discourses:
  $$I_i \models BG-Knowledge \cup Discourse_{so\_far}$$

- Consistency checks (“Mia’s husband loves Sally. She is not married.”)
  $$BG-Knowledge \cup Discourse_{so\_far} \not\models \neg New\_utterance$$
  iff $$BG-Knowledge \cup Discourse_{so\_far} \cup New\_utterance$$ is **satisfiable**

- Informativity checks (“Mia’s husband loves Sally. She is married.”)
  $$BG-Knowledge \cup Discourse_{so\_far} \not\models New\_utterance$$
  iff $$BG-Knowledge \cup Discourse_{so\_far} \cup \neg New\_utterance$$ is **satisfiable**
Example - Group Theory

The following axioms specify a group

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

Does

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

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\}

\[
e : 1
\]

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

\[
* : 
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 1 & 4 & 3 & 6 & 5 \\
3 & 3 & 5 & 1 & 6 & 2 & 4 \\
4 & 4 & 6 & 2 & 5 & 1 & 3 \\
5 & 5 & 3 & 6 & 1 & 4 & 2 \\
6 & 6 & 4 & 5 & 2 & 3 & 1 \\
\end{array}
\]
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:
\begin{align*}
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
\end{align*}
Totality:
\begin{align*}
a &= 1 \lor a = 2 \\
f(1) &= 1 \lor f(1) = 2 \\
f(2) &= 1 \lor f(2) = 2
\end{align*}
Functionality:
\begin{align*}
a &\neq 1 \lor a \neq 2 \\
f(1) &\neq 1 \lor f(1) \neq 2 \\
f(2) &\neq 1 \lor f(2) \neq 2
\end{align*}

A model is obtained by setting the blue literals true
Theory Reasoning

Let $T$ be a first-order theory of signature $\Sigma$
Let $L$ be a class of $\Sigma$-formulas

The $T$-validity Problem

- Given $\phi$ in $L$, is it the case that $T \models \phi$? More accurately:
- Given $\phi$ in $L$, is it the case that $T \models \forall \phi$?

Examples

- “$0/0, s/1, +/2, =/2, \leq /2$” $\models \exists y. y > x$
- The theory of equality $E \models \phi$ (\(\phi\) arbitrary formula)
- “An equational theory” $\models \exists s_1 = t_1 \land \cdots \land s_n = t_n$
  (E-Unification problem)
- “Some group theory” $\models s = t$ (Word problem)

The $T$-validity problem is decidable only for restricted $L$ and $T$. 
Approaches to Theory Reasoning

Theory-Reasoning in Automated First-Order Theorem Proving

- Semi-decide the $T$-validity problem, $T \models \phi$?
- $\phi$ arbitrary first-order formula, $T$ universal theory
- Generality is strength and weakness at the same time
- Really successful only for specific instance:
  $T = \text{equality}$, inference rules like paramodulation

Satisfiability Modulo Theories (SMT)

- Decide the $T$-validity problem, $T \models \phi$?
- Usual restriction: $\phi$ is quantifier-free, i.e. all variables implicitly universally quantified
- Applications in particular to formal verification
Checking Satisfiability Modulo Theories

**Given:** A quantifier-free formula $\phi$ (implicitly existentially quantified)

**Task:** Decide whether $\phi$ is T-satisfiable

($T$-validity via “$T \models \forall \phi$” iff “$\exists \neg \phi$ is not $T$-satisfiable”)

**Approach:** eager translation into SAT

- Encode problem into a $T$-equisatisfiable propositional formula
- Feed formula to a SAT-solver
- Example: $T =$ equality (Ackermann encoding)

**Approach:** lazy translation into SAT

- Couple a SAT solver with a given decision procedure for T-satisfiability of ground literals
- For instance if $T$ is “equality” then the Nelson-Oppen congruence closure method can be used
Lazy Translation into SAT

\[ g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \]

Theory: Equality
Lazy Translation into SAT

\[
\begin{align*}
g(a) &= c & \land & & \frac{f(g(a)) \neq f(c)}{2} & \lor & & \frac{g(a) = d}{3} & \land & & \frac{c \neq d}{4}
\end{align*}
\]
Lazy Translation into SAT

\[
\begin{align*}
g(a) &= c & \quad & 1 \\
f(g(a)) \neq f(c) & \quad & \lor & \quad & 2 \\
g(a) &= d & \quad & \land & \quad & 3 \\
c \neq d & \quad & \land & \quad & 4
\end{align*}
\]

- Send \{1, 2 \lor 3, 4\} to SAT solver.
Lazy Translation into SAT

\[
g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d
\]

- Send \(\{1, \overline{2} \lor 3, \overline{4}\}\) to SAT solver.
- SAT solver returns model \(\{1, \overline{2}, \overline{4}\}\).
  Theory solver finds \(\{1, \overline{2}\}\) \textit{E-unsatisfiable}. 
Lazy Translation into SAT

\[
\begin{align*}
g(a) &= c & \land & \quad f(g(a)) & \neq f(c) & \lor & \quad g(a) &= d & \land & \quad c & \neq d \\
& & & \text{(1)} & \quad & \text{(2)} & \quad & \text{(3)} & \quad & \text{(4)}
\end{align*}
\]

- Send \{1, \overline{2} \lor 3, \overline{4}\} to SAT solver.

- SAT solver returns model \{1, \overline{2}, \overline{4}\}.
  Theory solver finds \{1, \overline{2}\} \text{ \textit{E-unsatisfiable.}}

- Send \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2\} to SAT solver.
Lazy Translation into SAT

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \]

- Send \( \{1, \overline{2} \lor 3, \overline{4}\} \) to SAT solver.

- SAT solver returns model \( \{1, \overline{2}, \overline{4}\} \).
  Theory solver finds \( \{1, \overline{2}\} \) \( E\)-unsatisfiable.

- Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2\} \) to SAT solver.

- SAT solver returns model \( \{1, 2, 3, \overline{4}\} \).
  Theory solver finds \( \{1, 3, \overline{4}\} \) \( E\)-unsatisfiable.
Lazy Translation into SAT

\[ g(a) = c \quad \land \quad \frac{f(g(a)) \neq f(c)}{2} \quad \lor \quad \frac{g(a) = d}{3} \quad \land \quad \frac{c \neq d}{4} \]

- Send \( \{1, 2 \lor 3, 4\} \) to SAT solver.
- SAT solver returns model \( \{1, 2, 4\} \).
  Theory solver finds \( \{1, 2\} \) \( E \)-unsatisfiable.
- Send \( \{1, 2 \lor 3, 4, 1 \lor 2\} \) to SAT solver.
- SAT solver returns model \( \{1, 2, 3, 4\} \).
  Theory solver finds \( \{1, 3, 4\} \) \( E \)-unsatisfiable.
- Send \( \{1, 2 \lor 3, 4, 1 \lor 2, 1 \lor 3 \lor 4\} \) to SAT solver.
  SAT solver finds \( \{1, 2 \lor 3, 4, 1 \lor 2, 1 \lor 3 \lor 4\} \) unsatisfiable.
Lazy Translation into SAT: Summary

- Abstract $T$-atoms as propositional variables
- SAT solver computes a model, i.e. satisfying boolean assignment for propositional abstraction (or fails)
- Solution from SAT solver may not be a $T$-model. If so,
  - Refine (strengthen) propositional formula by incorporating reason for false solution
  - Start again with computing a model
Optimizations

Theory Consequences

The theory solver may return consequences (typically literals) to guide the SAT solver.

Online SAT solving

The SAT solver continues its search after accepting additional clauses (rather than restarting from scratch).

Preprocessing atoms

Atoms are rewritten into normal form, using theory-specific atoms (e.g. associativity, commutativity).

Several layers of decision procedures

“Cheaper” ones are applied first.
Combining Theories

Theories:

- $\mathcal{R}$: theory of rationals
  $\Sigma_\mathcal{R} = \{\leq, +, -, 0, 1\}$

- $\mathcal{L}$: theory of lists
  $\Sigma_\mathcal{L} = \{=, \text{hd}, \text{tl}, \text{nil}, \text{cons}\}$

- $\mathcal{E}$: theory of equality
  $\Sigma$: free function and predicate symbols

Problem: Is

\[
x \leq y \land y \leq x + \text{hd} \left( \text{cons} (0, \text{nil}) \right) \land P(h(x) - h(y)) \land \neg P(0)
\]

satisfiable in $\mathcal{R} \cup \mathcal{L} \cup \mathcal{E}$?
Nelson-Oppen Combination Method


Given:

- $\mathcal{T}_1, \mathcal{T}_2$ first-order theories with signatures $\Sigma_1, \Sigma_2$
- $\Sigma_1 \cap \Sigma_2 = \emptyset$
- $\phi$ quantifier-free formula over $\Sigma_1 \cup \Sigma_2$

Obtain a decision procedure for satisfiability in $\mathcal{T}_1 \cup \mathcal{T}_2$ from decision procedures for satisfiability in $\mathcal{T}_1$ and $\mathcal{T}_2$. 
Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd}(\text{cons}(0, \text{nil})) \land P(h(x) - h(y)) \land \neg P(0) \]
Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd}([0, \text{nil}]) \land P(h(x) - h(y)) \land \neg P(\text{0}) \]
Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd(cons(0, nil))} \land P(h(x) - h(y)) \land \neg P(0) \]
Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

\[
\begin{align*}
  x \leq y & \land y \leq x + \text{hd(cons(0, nil))} \land P(h(x) - h(y)) \land \neg P(0) \\
  v_1 & \quad \quad \quad v_2 \quad \quad \quad v_3 \quad \quad \quad v_4 \quad \quad \quad \quad \quad v_5
\end{align*}
\]

<table>
<thead>
<tr>
<th>$\mathcal{R}$</th>
<th>$\mathcal{L}$</th>
<th>$\mathcal{E}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leq y$</td>
<td></td>
<td>$P(v_2)$</td>
</tr>
<tr>
<td>$y \leq x + v_1$</td>
<td></td>
<td>$\neg P(v_5)$</td>
</tr>
<tr>
<td>$v_2 = v_3 - v_4$</td>
<td>$v_1 = \text{hd(cons(v_5, nil))}$</td>
<td>$v_3 = h(x)$</td>
</tr>
<tr>
<td>$v_5 = 0$</td>
<td></td>
<td>$v_4 = h(y)$</td>
</tr>
</tbody>
</table>
Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd}(\text{cons}(0, \text{nil})) \land P(h(x) - h(y)) \land \neg P(0) \]

\[ v_1 \]

\[ v_2 = v_3 - v_4 \]

\[ v_5 = 0 \]

\[ v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) \]

\[ v_3 = h(x) \]

\[ v_4 = h(y) \]

\[ v_1 = v_5 \]
**Nelson-Oppen Combination Method**

Variable abstraction + equality propagation:

\[
x \leq y \land y \leq x + \text{hd}(\text{cons}(0, \text{nil})) \land P(h(x) - h(y)) \land \neg P(0)
\]

### \( R \)

- \( x \leq y \)
- \( y \leq x + v_1 \)
- \( v_2 = v_3 - v_4 \)
- \( v_5 = 0 \)
- \( x = y \)

### \( L \)

- \( v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) \)

### \( E \)

- \( P(v_2) \)
- \( \neg P(v_5) \)
- \( v_3 = h(x) \)
- \( v_4 = h(y) \)
- \( v_1 = v_5 \)
Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd}(\text{cons}(0, \text{nil})) \land P(h(x) - h(y)) \land \neg P(0) \]

<table>
<thead>
<tr>
<th>( \mathcal{R} )</th>
<th>( \mathcal{L} )</th>
<th>( \mathcal{E} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leq y )</td>
<td>( v_1 )</td>
<td>( P(v_2) )</td>
</tr>
<tr>
<td>( y \leq x + v_1 )</td>
<td></td>
<td>( \neg P(v_5) )</td>
</tr>
<tr>
<td>( v_2 = v_3 - v_4 )</td>
<td>( v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) )</td>
<td></td>
</tr>
<tr>
<td>( v_5 = 0 )</td>
<td></td>
<td>( v_3 = h(x) )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( v_1 = v_5 )</td>
<td>( v_3 = v_4 )</td>
</tr>
</tbody>
</table>

Automated Theorem Proving – Peter Baumgartner – p.90
Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd}(\text{cons}(0, \text{nil})) \land P(h(x) - h(y)) \land \neg P(0) \]

<table>
<thead>
<tr>
<th>R</th>
<th>L</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leq y )</td>
<td></td>
<td>( P(v_2) )</td>
</tr>
<tr>
<td>( y \leq x + v_1 )</td>
<td></td>
<td>( \neg P(v_5) )</td>
</tr>
<tr>
<td>( v_2 = v_3 - v_4 )</td>
<td>( v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) )</td>
<td>( v_3 = h(x) )</td>
</tr>
<tr>
<td>( v_5 = 0 )</td>
<td></td>
<td>( v_4 = h(y) )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( v_1 = v_5 )</td>
<td>( v_3 = v_4 )</td>
</tr>
<tr>
<td>( v_2 = v_5 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

\[ x \leq y \land y \leq x + \text{hd}(\text{cons}(0, \text{nil})) \land P(h(x) - h(y)) \land \neg P(0) \]

\[
\begin{array}{c|c|c}
\mathcal{R} & \mathcal{L} & \mathcal{E} \\
\hline
x \leq y & P(v_2) & P(v_2) \\
y \leq x + v_1 & v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) & v_3 = h(x) \\
v_2 = v_3 - v_4 & v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) & v_3 = h(y) \\
v_5 = 0 & v_1 = v_5 & v_4 = h(y) \\
x = y & v_2 = v_5 & v_3 = v_4 \\
v_2 = v_5 & & \bot \\
\end{array}
\]
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)
- Talked about DPLL and Satisfiability Modulo Theories (SMT)

Further Topics

- Redundancy elimination, efficient equality reasoning, adding arithmetics to first-order theorem provers
- 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