Automated Reasoning

Peter Baumgartner
NICTA, Canberra
and
RSISE, ANU

Contact
Office: NICTA Bldg, Tower A, 7 London Circuit
Email: Peter.Baumgartner@nicta.com.au
Web: http://users.rsise.anu.edu.au/~baumgart/

Many slides based on material from Scott Sanner

Schedule

• Introduction to Logic (John Slaney)
  – Starting week of March 3
• Automated Reasoning (Peter Baumgartner)
  – Starting week of March 17
• SAT solving (Anbulagan)
  – Starting week of March 31
• Knowledge Compilation (Jinbo Huang)
  – Starting week of April 28
• Temporal Logic (Michael Norrish)
  – Starting week of May 12
• Higher-Order Logic (Jeremy Dawson)
  – Starting week of May 26
Automated Reasoning

• ... vs. calculation:
  – Problem: $2^2 = ? \quad 3^2 = ? \quad 4^2 = ?$
  – "Easy", often polynomial

• ... vs. constraint solving:
  – Problem spec: $x^2 = a$ where $x \in [1 .. b]$
  – Problem instance: fix parameter values $a$ and $b$: $a = 16, b = 10$
  – Find satisfying values then for variable $x$ (from finite domain)
  – "Difficult", often exponential (NP-complete problems)

• ... is, among others, about (first-order logic) theorem proving:
  – Problem: $\exists x \ (x^2 = a \land x \in [1 .. b])$
  – Is is satisfiable? valid?
  – "Very difficult" (often undecidable)

Logical Analysis Example: Three-Coloring Problem

Problem: Given a map
Can it be colored with only three colors?
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 values for variables such that spec is satisfied.
    - Variables: colors of nodes in the graph
    - Values: red, green or blue

- ... with a **first-order logic theorem prover**
  - Let theorem prover prove that the three-colouring 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 first-order theorem prover (e.g. use a propositional SAT solver)
  - Theorem provers are not even guaranteed to terminate on such problems!

---

### Three-Coloring Problem: The Role of Theorem Proving

- Functional Dependencies
  - The blue coloring functionally depends on the red and green coloring
    - ![Diagram of functional dependency]
  - The blue coloring does not functionally depend on the red coloring
    - ![Diagram of non-functional dependency]

- Theorem proving tasks: are the following valid (expressed as formulas)?
  - The blue coloring functionally depends on the red and green coloring
  - The blue coloring functionally depends on the red coloring
- (Learning about functional dep. might be instructive for modeller and solver)
- These are "proper" theorem proving tasks: analysis wrt all instances
- Demo now, files can be downloaded from my web page
Abstracting from the Example

• **AR systems functionality**
  – **Input**: a set of formulas in a specific **logical language**
  – **Run**: analyze these formulas by **logical inference** for a specific **task**
  – **Output**: the result of the analysis (proof, counterexample, solution...)

• **Rationale** - deduction: the "ultimate declarative paradigm"
  – Formulas describe possible worlds
  – Draw conclusions by (sound) logical inference
  – Learn something about the "real world"

• **Logical language and semantics**
  – Propositional, first-order, higher-order, modal, description logic, ...
  – monotonic/non-monotonic, probabilistic, resource-bounded, ...

• **Logical Inference and task**
  – Calculus (Resolution, ...) -> Proof procedure -> Implementation
  – Prove theorem, disprove conjecture, plausible explanation, find a model,...

History

• **Pre-computer era**
  – Early: Aristotle, Leibniz
  – 19th century: Boole, DeMorgan, Peano, Cantor and others
  – 20th century: Hilbert, Skolem, Herbrand, Gödel, Gentzen, Church

• **Computer era** (among many others)
  – 1960s Calculi
    • Davis-Putnam-Logemann-Loveland (DPLL), Resolution, Model Elimination
  – 1970s Logic programming
    • Prolog
  – 1980s Knowledge representation
    • Description Logics
  – 1990s Modern theory of resolution
  – 2000s (Serious) applications
Applications

- **Proofs of Mathematical Conjectures**
  - Graph theory: Four color theorem
  - Boolean algebra: Robbins conjecture

- **Verification**
  - Hardware: arithmetic units correctness
  - Software: functional correctness, safety properties, static checking

- **Query Answering**
  - Build domain-specific knowledge bases, use theorem proving to answer queries

- **Key to Success**
  - Chose your logic and calculus carefully (e.g. avoid undecidable logic if possible)
  - Need domain-specific optimizations (e.g. avoid successor-arithmetic)
  - Domain-independent optimization (e.g. "subsumption", good data structures)

- **Next**: preview of some logics and calculi

---

Example of Propositional Logic Sequent Proof

- **Given:**
  - **Axioms:** None
  - **Conjecture:** \( A \lor \neg A \)?

- **Inference:**
  - Gentzen
  - Sequent
  - Calculus

- **Direct Proof:**

  \[
  \begin{align*}
  (I) & \quad A \vdash A \\
  (\neg R) & \quad \vdash \neg A, A \\
  (\lor R2) & \quad \vdash A \lor \neg A, A \\
  (PR) & \quad \vdash A, A \lor \neg A \\
  (\lor R1) & \quad \vdash A \lor \neg A, A \lor \neg A \\
  (CR) & \quad \vdash A \lor \neg A
  \end{align*}
  \]

- **Problem:**
  - the Sequent Calculus is deduction-complete - it can derive *every* tautology
  - Calculi for ATP used nowadays are only refutation-complete (they can only derive a contradiction for a given theorem)
Example of First-order Logic Resolution Proof

- **Given:**
- **Axioms:**
  \[ \forall x \text{ Man}(x) \implies \text{Mortal}(x) \]
  \[ \text{Man}(\text{Socrates}) \]
- **Conjecture:**
  \[ \exists y \text{ Mortal}(y) ? \]
- **Inference:**
  Resolution calculus

<table>
<thead>
<tr>
<th>CNF:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \neg \text{Man}(x) \lor \text{Mortal}(x) )</td>
</tr>
<tr>
<td>( \text{Man}(\text{Socrates}) )</td>
</tr>
<tr>
<td>( \neg \text{Mortal}(y) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Proof:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( \neg \text{Mortal}(y) )</td>
</tr>
<tr>
<td>2. ( \neg \text{Man}(x) \lor \text{Mortal}(x) )</td>
</tr>
<tr>
<td>3. ( \text{Man}(\text{Socrates}) )</td>
</tr>
<tr>
<td>4. ( \text{Mortal}(\text{Socrates}) )</td>
</tr>
<tr>
<td>5. ( \bot )</td>
</tr>
<tr>
<td>Contradiction ( \Rightarrow ) Conj. is true</td>
</tr>
</tbody>
</table>

Example of Description Logic Tableaux Proof

- **Given:**
- **Axioms:**
  None
- **Conjecture:**
  \[ \exists \text{Child.}\neg\text{Male} \implies \neg \forall \text{Child.Male} ? \]
- **Inference:**
  Tableaux

<table>
<thead>
<tr>
<th>Proof:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check unsatisfiability of</td>
</tr>
<tr>
<td>[ \exists \text{Child.}\neg\text{Male} \land \forall \text{Child.Male} ]</td>
</tr>
</tbody>
</table>

| x: \[ \exists \text{Child.}\neg\text{Male} \land \forall \text{Child.Male} \] |
| x: \( \forall \text{Child.Male} \) |

| y: \( \exists \text{Child.}\neg\text{Male} \) |
| y: \( \forall \text{Child.Male} \) |

<CLASH>

Contradiction \( \Rightarrow \) Conj. is true
• For each calculus one has to specify:
  – Syntax and semantics of its logic
  – Foundational axioms (if any)
  – Inference rules
  – How to combine inference rules applications into derivations

• Derivability and Entailment
  – Let KB be the conjunction of axioms
  – Let F be a formula (possibly a conjecture)
  – We say $KB \vdash F$ (read: KB derives F)
    if F can be derived from KB through rules of inference
  – We say $KB \models F$ (read: KB entails F, or KB models F)
    if (model-theoretic) semantics hold that F is true whenever KB is true

---

**Model-Theoretic Semantics**

• Model-theoretic semantics for (propositional) logics
  – An interpretation is a truth assignment to atomic elements of a KB:
    $I \langle C, D \rangle \in \{ \langle F, F \rangle, \langle F, T \rangle, \langle T, F \rangle, \langle T, T \rangle \}$
  – A model of a formula is an interpretation where it is true:
    $I \langle C, D \rangle = \langle F, T \rangle$ models $C \lor D$, $C \Rightarrow D$, but not $C \land D$
  – Two important properties of a formula C w.r.t. axioms of KB:
    • Entailment, written as $KB \models C$: C is true in all models of KB
    • Consistency: C is true in $\geq 1$ model of KB

• Think of truth in a set-theoretic manner

\[ KB \models C \]

C

KB

Models of KB
\subseteq Models of C
Calculi and Properties

• Important properties of calculi:
  – Soundness: If KB ⊢ C then KB ⊨ C
  – Completeness: If KB ⊨ C then KB ⊢ C
  – Refutational completeness: KB ∪ { ¬C } ⊢ ⊥
  – Termination: halts on any KB and C after finite time

• These properties may be incompatible, depending on the logic
  – Decidable logics: all three
    • Example: propositional logic
  – Semi-decidable logics: can have sound and (refutationally) complete calculus
    • Thus terminates if KB ⊨ C. Example: first-order logic
  – Non-r.e. logics: can’t have sound and (refutationally) complete calculus
    • Example: second-order logic

Propositional Logic Syntax

• Propositional variables: p, rain, sunny
• Connectives: ⇒ ⇔ ¬ ∧ ∨
• Inductive definition of well-formed formula (wff):
  – Base: All propositional vars are wffs
  – Inductive 1: If A is a wff then ¬A is a wff
  – Inductive 2: If A and B are wffs then A ∧ B, A ∨ B, A ⇒ B, A ⇔ B are wffs
• Examples:
  – rain, rain ⇒ ¬ sunny
  – (rain ⇒ ¬ sunny) ⇔ (sunny ⇒ ¬rain)
Prop. Logic Semantics

- For a formula $F$, the truth $I(F)$ under interpretation $I$ is recursively defined:
  - **Base:**
    - $F$ is prop var $A$ then $I(F)=true$ iff $I(A)=true$
  - **Recursive:**
    - $F$ is $\neg C$ then $I(F)=true$ iff $I(C)=false$
    - $F$ is $C \land D$ then $I(F)=true$ iff $I(C)=true$ and $I(D)=true$
    - $F$ is $C \lor D$ then $I(F)=true$ iff $I(C)=true$ or $I(D)=true$
    - $F$ is $C \Rightarrow D$ then $I(F)=true$ iff $I(\neg C \lor D)=true$
    - $F$ is $C \Leftrightarrow D$ then $I(F)=true$ iff $I(C \Rightarrow D)=true$ and $I(D \Rightarrow C)=true$

- Truth defined recursively from ground up
  - Modal logics don't have this property!

CNF Normalization

- Many theorem proving techniques req. KB to be in clausal normal form (CNF):
  - Rewrite all $C \leftrightarrow D$ as $C \Rightarrow D \land D \Rightarrow C$
  - Rewrite all $C \Rightarrow D$ as $\neg C \lor D$
  - Push negation through connectives:
    - Rewrite $\neg(C \land D)$ as $\neg C \lor \neg D$
    - Rewrite $\neg(C \lor D)$ as $\neg C \land \neg D$
  - Rewrite double negation $\neg \neg C$ as $C$
  - Now NNF, to get CNF, distribute $\lor$ over $\land$:
    - Rewrite $(C \land D) \lor E$ as $(C \lor E) \land (D \lor E)$
  - A clause is a disjunction of literals (pos/neg propositional variables)
  - Can express KB, a set of clauses, as the conjunction of its clauses
CNF Normalization Example

• Given KB with single formula:
  – \( \neg (\text{rain} \Rightarrow \text{wet}) \Rightarrow (\text{inside} \land \text{warm}) \)

• Rewrite all \( C \Rightarrow D \) as \( \neg C \lor D \)
  – \( \neg \neg (\neg \text{rain} \lor \neg \text{wet}) \lor (\text{inside} \land \text{warm}) \)

• Push negation through connectives:
  – \( (\neg \neg \neg \text{rain} \lor \neg \neg \text{wet}) \lor (\text{inside} \land \text{warm}) \)

• Rewrite double negation \( \neg \neg C \) as \( C \)
  – \( (\neg \text{rain} \lor \neg \text{wet}) \lor (\text{inside} \land \text{warm}) \)

• Distribute \( \lor \) over \( \land \):
  – \( (\neg \text{rain} \lor \neg \text{wet} \lor \text{inside}) \land (\neg \text{rain} \lor \neg \text{wet} \lor \text{warm}) \)

• CNF KB: \{\neg \text{rain} \lor \neg \text{wet} \lor \text{inside}, \neg \text{rain} \lor \neg \text{wet} \lor \text{warm}\}

Prop. Theorem Proving

• \( A \Rightarrow B \) iff \( A \land \neg B \) is unsatisfiable

• Propositional logic is decidable, but NP-complete (reduction to 3-SAT)

• State-of-the-art prop. unsatisfiability methods are DPLL-based

• Many optimizations, more in lecture on SAT solving by Anbulagan

Instantiate prop vars until all clauses falsified, backtrack and do for all instantiations \( \Rightarrow \) unsat!
Prop. Tableaux Methods

- Given negated query F (in NNF), use rules to recursively break down:
  - \( \alpha \)-Rule: Given \( A \land B \) add A and B
  - \( \beta \)-Rule: Given \( A \lor B \) branch on A and B
  - Clash: If A and \( \neg A \) occur on same branch
  - Clash on all branches indicates unsat!

\[
\begin{array}{c}
A \land \neg A \land \neg B \land B \\
\mid \quad (\beta \text{-Rule}) \quad B \land B \quad (\beta \text{-Rule}) \\
A \quad (\alpha \text{-Rule}) \quad \neg B \quad (\alpha \text{-Rule}) \\
\neg A \quad (\alpha \text{-Rule}) \quad B \quad (\alpha \text{-Rule}) \\
(\text{Clash}) \quad (\text{Clash})
\end{array}
\]

Propositional Resolution

- One (!) inference rule

**Resolution:**

\[
\begin{array}{c}
A \lor B \\
\mid \quad \neg B \lor C \\
A \lor C
\end{array}
\]

**Example application:**

\[
\begin{array}{c}
\neg \text{precip} \lor \neg \text{freezing} \lor \text{snow} \\
\mid \quad \neg \text{snow} \lor \text{slippery} \\
\neg \text{precip} \lor \neg \text{freezing} \lor \text{slippery}
\end{array}
\]

- The resolution calculus is sound and (refutationally) complete:
  \( KB \models C \) if and only if \( KB \cup \{\neg C\} \vdash \bot \)
  - Simple strategy for completeness is to close clause set under Resolution
- NB: "One inference rule" calculus treats clauses as sets, otherwise need factoring:

**Factoring:**

\[
\begin{array}{c}
A \lor A \lor B \\
\mid \quad A \lor B
\end{array}
\]

Soundness and completeness proof: see blackboard
Resolution Strategies

- Need strategies to restrict search:
  - **Unit resolution**
    - Only resolve with unit clauses
    - Complete for Horn KB (gives a "bottom-up flavour")
    - Intuition: Decrease clause size
  - **Set of support** (see also next two slides)
    - SOS starts with query clauses
    - Only resolve SOS clauses with non-SOS clauses and put resolvents in SOS
    - Intuition: KB should be satisfiable so refutation should derive from query
  - **Linear resolution**
    - Only resolve query clause with KB clauses, resolvent is new query clause
    - Complete for Horn KB (gives a "top-down flavour"), basis for Prolog
    - Together with ancestor resolution ⇒ complete for non-Horn, too
  - **Ordered resolution** resolve on maximal literals in clause only

The "Given Clause Loop"

- In Otter theorem prover [http://en.wikipedia.org/wiki/Otter_(theorem_prover)]
- Lists of clauses maintained by the algorithm: USABLE and SOS.
- Initialize SOS with the input clauses, USABLE empty.
- Algorithm

```
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 here: define clause weight e.g. as `depth + length` of clause
- Important property: no clause is delayed infinitely long in (1)
**First-order logic**

- Refer to objects and relations between them
- Propositional logic requires all relations to be propositionalized
  - Peter-at-home, Peter-at-work, Jim-at-subway, etc...
- Really want a compact relational form:
  - at(Peter, home), at(Peter, work), at(Jim, subway), etc...
- Then can use variables and quantify over all objects:
  - $\forall x \ (\text{person}(x) \Rightarrow \exists y \ at(x,y) \land \text{place}(y))$
From Propositional Logic to First-Order Logic

• Generalize Syntax
• Generalize Semantics
  – Work with Herbrand interpretations
• Clause normal form generation
  – Involves Skolemization now
• Calculi for first-order clause logic
  – Involve substitutions and unification now

First-order Logic Syntax

• Terms (technical definition is inductive because of function symbols)
  – Variables: w, x, y, z
  – Constants: a, b, c, d
  – Functions over terms: f(a), f(x,y), f(x,c,f(f(z)))
• Atoms: P(x), Q(f(x,y)), R(x, f(x,f(c,z),c))
• Connectives: ⇒ ⇔ ¬ ∧ ∨
• Quantifiers: ∀ ∃

• Inductive definition of wff:
  – Same as propositional logic but with following modifications
  – Base: All atoms over terms are wffs
  – Inductive: If A is a wff and x is a variable term
    then ∀x A and ∃x A are wffs
First-order Logic Semantics

- Interpretation $I = (\Delta I, \bullet I)$
  - $\Delta I$ is a non-empty domain
  - $\bullet I$ maps each function symbol $f$ of arity $n$ to a total function $(\Delta I)^n \rightarrow \Delta I$
  - $\bullet I$ maps each predicate symbol $P$ of arity $n$ into a set of $n$-tuples over $\Delta I$

- **Herbrand** interpretations (Th: KB is satisfiable iff it has a Herbrand model)
  - $\Delta I$ is set of ground terms $\{ \text{Peter}, \text{Jim}, \text{loc}(\text{Peter}), \text{loc}(\text{Jim}), \text{loc}(\text{loc}(\text{Peter})), \ldots \}$
  - $\bullet I$ maps each $f$ to the identity function. Thus, $I(\text{loc}(\text{Peter})) = \text{loc}(\text{Peter})$
  - $\bullet I$ maps each predicate symbol $P$ of arity $n$ into a set of $n$-tuples over $\Delta I$
  - Logical connectives interpreted as in propositional logic; new: $I \models \forall x A$ iff $I \models A[x/t]$, for all ground terms $t \in \Delta I$

- Example
  - $\bullet I$ may map $\text{at}(\bullet, \bullet)$ into $\{ \langle \text{Peter}, \text{loc}(\text{Peter}) \rangle, \langle \text{Jim}, \text{loc}(\text{Jim}) \rangle \}$
  - All other ground predicates are false in $I$, e.g. $\text{at}(\text{Jim}, \text{Jim})$

Skolemization

- Skolemization is the process of getting rid of all $\exists$ quantifiers from a formula while preserving (un)satisfiability:
  - If $\exists x$ quantifier is the outermost quantifier, remove the $\exists$ quantifier and substitute a new constant for $x$
  - If $\exists x$ quantifier occurs inside of $\forall$ quantifiers, remove the $\exists$ quantifier and substitute a new function of all $\forall$ quantified variables for $x$

- Examples:
  - $\text{Skolemize}(\exists w \exists x \forall y \forall z P(w,x,y,z)) =$
    $$\forall y \forall z P(c,d,y,z)$$
  - $\text{Skolemize}(\forall w \exists x \forall y \exists z P(w,x,y,z)) =$
    $$\forall w \forall y P(w,f(w),y,g(w,y))$$
### CNF Conversion

- CNF conversion is the same as the propositional case up to NNF, then do:
  - Standardize apart variables (all quantified variables get different names)
    - e.g. $(\forall x \ A(x)) \land (\exists x \ \neg A(x))$ becomes $(\forall x \ A(x)) \land (\exists y \ \neg A(y))$
  - Shift all quantifiers in front of formula (obtain, ultimately, prenex normal form)
    - $(\forall x \ A(x)) \land (\exists y \ \neg A(y))$ becomes $\exists y \ \forall x \ (A(x) \land \neg A(y))$
  - Skolemize formula
    - e.g. $\exists y \ \forall x \ (A(x) \land \neg A(y))$ becomes $\forall x \ (A(x) \land \neg A(c))$
  - Drop universals
    - e.g. $\forall x \ (A(x) \land \neg A(c))$ becomes $A(x) \land \neg A(c)$
  - Distribute $\lor$ over $\land$
  - Write result as a clause set (trivial)
    - e.g. $A(x) \land \neg A(c)$ becomes \{ $A(x), \neg A(c)$ \}

### Herbrand's Theorem

- Refutational theorem proving calculi are based on the following chain of reasoning
  - Any FO-formula is unsatisfiable iff its clause form is unsatisfiable
    (Non-trivial part is Skolemization)
  - A clause set is unsatisfiable iff it has no satisfying Herbrand interpretation
    (i.e. no Herbrand model)
  - A clause set has no Herbrand model iff some finite set of ground instances of its clauses is unsatisfiable
    (Herbrand’s theorem)
  - A naive application of the chain leads to **Gilmore’s method**
    - It searches for this unsatisfiable set of ground instances in a direct way
### Gilmore's Method

**Preprocessing:**

<table>
<thead>
<tr>
<th>Given Formula</th>
<th>Clause Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀x ∃y P(y, x) ∧ ∀z ¬P(z, a)</td>
<td>P(f(x), x) ≤ ¬P(z, a)</td>
</tr>
</tbody>
</table>

**Outer loop:**

Grounding

**Inner loop:**

Propositional Method

**Preprocessing:**

<table>
<thead>
<tr>
<th>Given Formula</th>
<th>Clause Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀x ∃y P(y, x) ∧ ∀z ¬P(z, a)</td>
<td>P(f(x), x) ≤ ¬P(z, a)</td>
</tr>
</tbody>
</table>

**Outer loop:**

Grounding

**Inner loop:**

Propositional Method
Gilmore's Method

Preprocessing:

Given Formula: \( \forall x \exists y P(y, x) \land \forall z \neg P(z, a) \)

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

Outer loop:

Grounding:

Inner loop:

Propositional Method

Sat? No Yes

STOP: Proof found Continue Outer Loop

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

Clause Form: \( P(f(a), a) \land \neg P(a, a) \)

Inner loop:

Propositional Method
Problems with Gilmore's Method

- 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, they work directly on first-order clauses, not on their ground instances. This way, infinitely many ground resolution steps can be represented compactly with one first-order resolution step (sometimes)
- They use the **unification** operation to enable this
Better Methods for First-order Theorem Proving

- **Tableaux methods**
  - Highly successful for description and modal logics, which conform to certain (syntactically restricted) fragments of FOL
  - Not treated here
- **Resolution Methods**
  - Most successful technique for a variety of KBs
  - But... search space grows very quickly
  - Need a variety of optimizations in practice
    - strategies, ordering, redundancy elimination
- **Instance Based Methods**
  - Reduce proof search in FOL to proof search in propositional logic
  - Comparably new and interesting paradigm
- All methods are based on Herbrand interpretations
  - which justifies the use of **unification**

Substitution and Unification

- **Substitution**
  - A substitution list $\theta$ is a list of variable-term pairs
    - e.g., $\theta = \{x/3, y/f(z)\}$
  - When $\theta$ is applied to an FOL formula, every free occurrence of a variable in the list is replaced with the given term
    - e.g. $(P(x,y) \land \exists x P(x,y))\theta = P(3,f(z)) \land \exists x P(x,f(z))$
- **Unification / Most General Unifier**
  - The unifier $\text{UNIF}(x,y)$ of two atoms/terms is a substitution that makes both arguments identical
    - e.g. $\text{UNIF}( P(x,f(x)), P(y, f(f(z))) ) = \{x/f(1), y/f(1), z/1\}$
  - The most general unifier $\text{MGU}(x,y)$ is just that... all other unifiers can be obtained from the MGU by additional substitution (MGU exists for unifiable args)
    - e.g. $\text{MGU}( P(x,f(x)), P(y, f(f(z))) ) = \{x/f(z), y/f(z)\}$
An Instance-Based Method ("InstGen")

Current clauses

\[
\begin{align*}
P(f(x), x) \lor Q(x) & \quad \text{ground} \\
\neg P(z, a) \lor \neg Q(z) & \quad x, z \rightarrow \$
\end{align*}
\]

Model: \{P(f($), $), \neg P($, a)\}

Model determines literals selection in current clauses for InstGen inference:

\[
\begin{array}{c}
\text{InstGen} \\
P(f(x), x) \lor Q(x) & \quad \neg P(z, a) \lor \neg Q(z) \\
P(f(a), a) \lor Q(a) & \quad \neg P(f(a), a) \lor \neg Q(f(a))
\end{array}
\]

Conclusions are obtained by unifying selected literals
Add conclusions to "current clauses" and start over

Lifting Propositional Resolution to First-Order Resolution

- **Propositional Resolution**

  Clauses \quad \text{Ground instances}

  \[
  \begin{array}{c}
P(f(x), y) & \quad \{P(f(a), a), \ldots, P(f(f(a)), f(f(a))), \ldots\} \\
\neg P(z, z) & \quad \{-P(a), \ldots, \neg P(f(f(a)), f(f(a))), \ldots\}
\end{array}
  \]

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

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

- **Observation (leading to "lifting lemma of resolution inferences")**

  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**

  \[
  \begin{array}{c}
P(f(x), y) & \quad \neg P(z, z)
\end{array}
  \]

  Justified by existence of \(P(f(x), f(x))\) via unification;
observation above tells us that these are the only inferences neccessary
Resolution for First-Order Clauses

- **Inference rules**

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

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

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

- **Example**

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

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

---

**Example of First-Order Resolution Proof**

**Given:**

**Axioms:**
\[
\forall x \text{ Man}(x) \Rightarrow \text{Mortal}(x)
\]

\[
\text{Man}(\text{Socrates})
\]

**Conjecture:**

\[
\exists y \text{ Mortal}(y) \ ?
\]

**Inference:**

Refutation
Resolution

**CNF:**

\[
\neg \text{Man}(x) \lor \text{Mortal}(x)
\]

\[
\text{Man}(\text{Socrates})
\]

\[
\neg \text{Mortal}(y) \quad \text{[Neg. conj.]}\]

**Proof:**

1. \( \neg \text{Mortal}(y) \) [Neg. conj.]
2. \( \neg \text{Man}(x) \lor \text{Mortal}(x) \) [Given]
3. \( \text{Man}(\text{Socrates}) \) [Given]
4. \( \text{Mortal}(\text{Socrates}) \) [Res. 2,3]
5. \( \bot \) [Res. 1,4]

Contradiction \( \Rightarrow \) Conj. is true
Importance of Factoring

• Without the factoring rule, resolution is incomplete

• For example, take the following refutable clause set:
  \{ A(w) \lor A(z), \neg A(y) \lor \neg A(z) \}

• All binary resolutions yield clauses of the same form

• Clause set is only refutable if one of the clauses is first factored

Search Control

• Goal-directed / bottom-up search, as in propositional logic
  
  – SLD Resolution
    • KB of definite clauses (i.e. Horn rules), e.g.
      Uncle(x,y) :- Father(x,z) \land Brother(z,y)
      
    • Resolution backward chains from goal of rules
    • With negation-as-failure semantics, SLD- resolution is logic programming, i.e. Prolog
  
  – Negative and Positive Hyperresolution
    • All negative (positive) literals in nucleus clause are simultaneously resolved with completely positive (negative) satellite clauses
    • Positive Hyperresolution yields backward chaining
    • Negative Hyperresolution yields forward chaining
  
  • Such search strategies prevent the generation of resolvents, they don’t explain when clauses can be deleted (redundancy control)
Redundancy Control

- Redundancy of clauses is a huge problem in FOL resolution
  - For clauses C & D, C is redundant if \( \exists \theta \text{ s.t. } C\theta \subseteq D \) as a multiset, a.k.a. \( \theta \)-subsumption
  - If true, D is redundant and can be removed
    - Intuition: If D used in a refutation, C\( \theta \) could be substituted leading to even shorter refutation

- Two types of subsumption where N is a new resolvent and A is a current clause:
  - Forward subsumption: A \( \theta \)-subsumes N, delete N
  - Backward subsumption: N \( \theta \)-subsumes A, delete A

- Forward / backward subsumption expensive but saves many redundant inferences
- Leads to saturation-based theorem proving (with orderings, in general)

Saturation Theorem Proving

- Given a set of clauses S:
  - S is saturated if all possible inferences from clauses in S generate forward subsumed clauses
  - All new inferences are "redundant" then and need not be carried out, without sacrificing completeness
  - If S does not contain the empty clause then S is satisfiable

- Saturation without deriving the empty clause implies no proof possible! And the clause set is satisfiable then.

- Usually need ordering restrictions to reach finite saturation.
Term Indexing

- Term indexing is an implementation technique for fast retrieval of sets of terms / clauses matching criteria

- Common uses in modern theorem provers:
  - Term q (query) is unifiable with term t (in index), i.e., \( \exists \theta \text{ s.t. } q\theta = t \theta \)
  - Term t is an instance of q, i.e., \( \exists \theta \text{ s.t. } q\theta = t \)
  - Term t is a generalization of q, i.e., \( \exists \theta \text{ s.t. } q = t\theta \)
  - Clause q subsumes clause t, i.e., \( \exists \theta \text{ s.t. } q\theta \subseteq t \)
  - Clause q is subsumed by clause t, i.e., \( \exists \theta \text{ s.t. } t\theta \subseteq q \)

- Techniques: (Google for “term indexing”)
  - Path indexing
  - Substitution tree indexing, **discrimination trees**

Discrimination Tree Indexing

A discrimination tree

Stores \( P(a,a), P(a,b), P(b,a), \ldots -P(c,c) \)

- Tree structure to look up terms or literals from a (large) database
  - Branches store terms as written down (from left to right)
  - Doesn't distinguish different variables \( P(x,y) \) becomes \( P(?,?) \)
    (Can overretrieve)
  - More efficient for common uses than linear search
  - Can be combined with hashing of symbols, if branching is high
Equality

- A predicate w/ special interpretation
- Could axiomatize:
  - \( x = x \) (reflexive)
  - \( x = y \, \Rightarrow \, y = x \) (symmetric)
  - \( x = y \wedge y = z \, \Rightarrow \, x = z \) (transitive)
  - For each function symbol \( f \):
    - \( x_1 = y_1 \wedge \ldots \wedge x_n = y_n \, \Rightarrow \, f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \) (congruence)
  - For each predicate symbol \( P \):
    - \( x_1 = y_1 \wedge \ldots \wedge x_n = y_n \wedge P(x_1, \ldots, x_n) \, \Rightarrow \, P(y_1, \ldots, y_n) \) (congruence)
- Lead to bad search space
- Better to use dedicated inference rules (Paramodulation)

Inference Rules for Equality

- Demodulation (incomplete, based on matching)
  
  \[
  \frac{l = r \wedge L[t] \lor D}{L[r\theta] \lor D} \quad l\theta = t
  \]

  Example application:
  
  \[
  f(x) = x \quad P(f(a)) \lor Q \quad \theta = \{x/a\}
  \]

- Paramodulation (complete, based on unification)
  
  \[
  \frac{l = r \lor C \quad L[t] \lor D}{(L[r] \lor C \lor D)\theta} \quad \theta = \text{MGU}(l, t)
  \]

  Example application:
  
  \[
  f(x, a) = x \lor C(x) \quad P(f(b, y)) \lor Q(y) \quad \theta = \{x/b, y/a\}
  \]
  
  \[
  P(b) \lor C(b) \lor Q(a)
  \]
Equality Reasoning: Conclusions

• The inference rule of paramodulation together with the resolution and factoring inference rules constitute a sound a complete calculus for first-order logic with equality, i.e. can semi-decide the question whether

\[ E \vDash \phi \]

holds, where \( E \) is the theory of equality (Ref, Sym, Trans, Congruence) and \( \phi \) is an (arbitrary) formula.

• Caution: some search strategies no longer work (are incomplete), e.g. SOS
  – Unless "paramodulation into and below variables is permitted" (inpractical)
  – The practically most successful theorem provers are saturation-based, heavily use term orderings ("Replace bigger terms by smaller ones"), and the main inference rule is called "superposition"

• Natural question: can one "build-in" other/richer theories than \( E \)?
  – Answer: yes, the keyword is "Theory Reasoning"

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 \( T \vDash \phi \)?

More accurately:

Given \( \phi \) in \( L \), is it the case that \( T \vDash \forall \phi \)?

Examples

- "0/0, s/1, +/-2, <=/2" \( \vDash \exists \ y. \ y > x \)
- "The theory of equality \( E \)" \( \vDash \phi \) (\( \phi \) arbitrary formula, as above)
- "An equational theory" \( \vDash \exists \ s_1=t_1 \land \ldots \land s_n=t_n \) (E-Unification problem)
- "Some group theory" \( \vDash s = t \) (Word problem)

The T-validity problem is decidably only for restricted \( L \) and \( T \)
Theory Reasoning

The T-validity Problem

Is it the case that $T \vDash \phi$?

More accurately:

Is it the case that $T \vDash \forall \phi$?  I.e., Free vars are constants

The Dual Problem: T-satisfiability

Is it the case that $\phi$ is T-satisfiable?

More accurately:

Is it the case that $\exists \phi$ is T-satisfiable?  I.e., Free vars are constants

Prop: $T \vDash \Phi$ iff $\neg \Phi$ is T-unsatisfiable

Approaches to Theory Reasoning

- Theory-Reasoning in Automated First-Order Theorem Proving:
  - Semi-decide the T-validity problem, $T \vDash \phi$?
  - $\phi$ arbitrary first-order formula, $T$ universal theory
  - Generality is strength and weakness at the same time
  - Really successful only for specific instance:
    - $T =$ equality and equality inference rules like paramodulation
**Approaches to Hybrid Reasoning**

- **Satisfiability Modulo Theories (SMT)**
  - Decide the T-validity problem, $T \models \phi$?
  - Usual restrictions:
    - $\phi$ quantifier-free, i.e. all variables implicitly universally quantified
    - The T-satisfiability of conjunctions of literals must be decidable
  - SMT is the perhaps most advanced approach among those mentioned
  - Applications in particular to Formal verification

---

**Checking Satisfiability Modulo Theories**

**Usual Formulation**

**Given:**
- A decision procedure for T-satisfiability of sets of literals
- A quantifier-free formula $\phi$ (implicitly existentially quantified)

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

**Approaches:**
- **Eager translation into SAT**
  - Encode problem and theory into an equisatisfiable propositional formula
  - Feed formula to a SAT-solver
- **Lazy translation  into SAT**
  - Couple a SAT solver with a 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

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

Theory: Equality

\[ g(a) \equiv c \quad 1 \quad f(g(a)) \neq f(c) \quad 2 \quad g(a) = d \quad 3 \quad c \neq d \quad 4 \]

- 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.

- Send \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor \overline{3} \lor 4\} \) to SAT solver.
  SAT solver finds \( \{1, \overline{2} \lor 3, \overline{4}, \overline{1} \lor 2, \overline{1} \lor \overline{3} \lor 4\} \) unsatisfiable.
Lazy Translation Into SAT: Summary

• Mapping atoms to propositions is abstraction

• SAT solver computes a solution, i.e. boolean assignment for atoms in literal set

• Solution from SAT solver may not be true solution, i.e. the literal set is T-unsatisfiable

• Refine (strengthen) propositional formula by incorporating reason for false solution

• Reason provided by theory decision procedure, typically in form of subset of given literal set

More 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 restring from scratch)

▪ Backjumping
  – Instead of chronological backtracking

▪ 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
Some SMT Systems

- Argo-lib, University of Belgrade
- DPLL(T), Technical University of Catalonia, U Iowa
- CVC Lite, Stanford
- haRVey, Loria
- ICS, SRI
- Math-SAT, ITC
- Tsat++, University of Genova
- UCLID, CMU

Combining Theories

Theories:

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

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

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

Problem: Is

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

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 \( \mathcal{T}_2 \).

Nelson-Oppen Combination Method

*Variable abstraction + equality propagation:*

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

**Variable abstraction + equality propagation:**

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

\[ v_1 \]

\[ v_2 \]

\[ v_3 \]

\[ v_4 \]

\[ v_5 \]

\[ \mathcal{R} \quad \mathcal{L} \quad \mathcal{E} \]

\[
\begin{array}{ll}
  x & y \\
  x \leq y & P(v_2) \\
  y \leq x + v_1 & \neg P(v_5)
\end{array}
\]
Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

\[ v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \]

<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>-P(v_5)</td>
</tr>
<tr>
<td>v_2 = v_3 - v_4</td>
<td>[ ]</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></td>
<td>v_1 = \text{hd}(\text{cons}(v_5, \text{nil}))</td>
<td></td>
</tr>
</tbody>
</table>

Nelson-Oppen Combination Method

Variable abstraction + equality propagation:

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

\[ v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \]

<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>-P(v_5)</td>
</tr>
<tr>
<td>v_2 = v_3 - v_4</td>
<td>[ ]</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></td>
<td>v_1 = \text{hd}(\text{cons}(v_5, \text{nil}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v_1 = v_5</td>
<td></td>
</tr>
</tbody>
</table>
Variable abstraction + equality propagation:

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

<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>( v_1 = \text{hd}(\text{cons}(v_5, \text{nil})) )</td>
<td>( -P(v_5) )</td>
</tr>
<tr>
<td>( v_2 = v_3 - v_4 )</td>
<td>( v_3 = h(x) )</td>
<td>( v_4 = h(y) )</td>
</tr>
<tr>
<td>( v_5 = 0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x = y )</td>
<td>( v_1 = v_5 )</td>
<td>( v_3 = v_4 )</td>
</tr>
</tbody>
</table>
**Nelson-Oppen Combination Method**

**Variable abstraction + equality propagation:**

\[
x \leq y \land y \leq x + \underbrace{\text{hd(cons(0, nil))}}_{v_1} \land P(h(x) - h(y)) \land \neg P(0) \]

\[
\begin{array}{llll}
\mathcal{R} & \mathcal{L} & \mathcal{E} \\
\hline
x \leq y & \quad & P(v_2) \\
y \leq x + v_1 & \quad & \neg P(v_5) \\
v_2 = v_3 - v_4 & v_1 = \text{hd(cons(v_5, nil))} & v_3 = h(x) \\
v_5 = 0 & \quad & v_4 = h(y) \\
x = y & v_1 = v_5 & v_3 = v_4 \\
v_2 = v_5 & \quad & \bot
\end{array}
\]