First-Order Theorem Proving

Peter Baumgartner

Logic and Computation Program

Peter.Baumgartner@nicta.com.au

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

Contents

Part I - Motivation

Theorem Proving in Relation to ...

Just **one** perspective to explain what theorem proving is about

Calculation: Compute function value at given point:

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

"Easy" (often polynomial)

Constraint solving: Find value(s) for variable(s) such that problem instance is satisfied:

Is there x, y such that $x^2 = 16?$ $\quad x^2 = 17?$ $\quad x^2 = y?$

"Difficult" (often exponential)

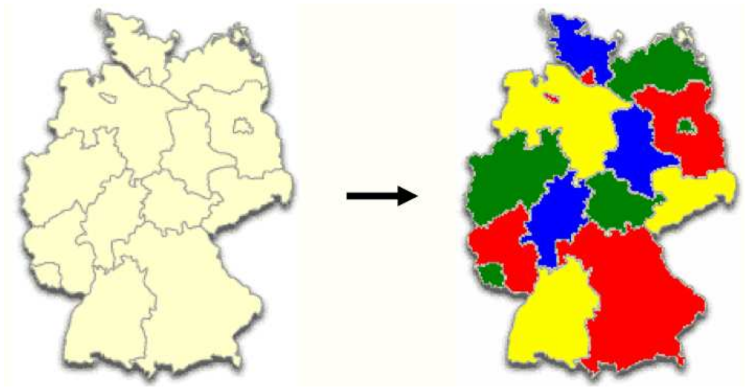
Theorem proving: Prove a formula holds true:

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

In general: (semi-)automatically analyse formula for logical properties

"Very difficult" (often undecidable)

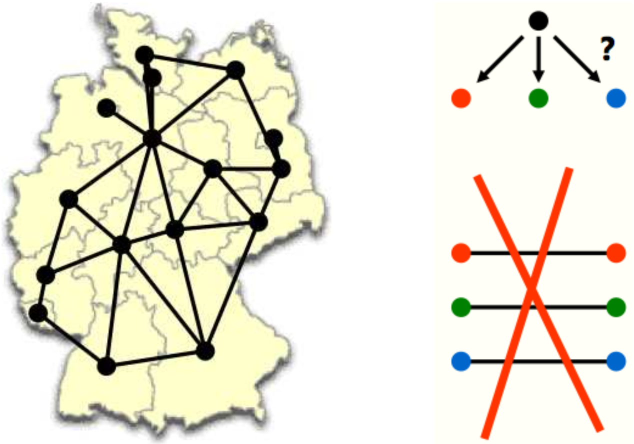
Example: Three Coloring Problem



Problem: Given a map. Can it be colored using only three colors, where neighbouring countries are colored differently?

Three Coloring Problem - Graph Theory Abstraction

Problem Instance **Problem Specification**



The Rôle of Theorem Proving?

Three Coloring Problem: The Rôle of Theorem Proving

To apply theorem provers, the domain has to be formalized in logic

Every node has at least one color

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

Every node has at most one color

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

Adjacent nodes have different color

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

Three Coloring Problem: The Rôle of Theorem Proving

Constraint Solving: Find value(s) for variable(s) such that problem instance is satisfied

Here: Variables: Nodes in graph

Values: Red, green or blue

Problem instance: Specific graph to be colored

Constraint solving $\rightsquigarrow$ **Theorem proving** Prove that

general three coloring formula (see previous slide) +
specific graph (as a formula)

is satisfiable

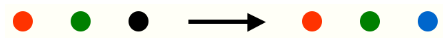
On such problems, a constraint solver is usually more efficient than a theorem prover

Other tasks where theorem proving is more appropriate?

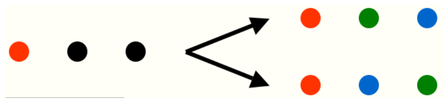
Three Coloring Problem: The Rôle of Theorem Proving

Functional dependancy

- Blue coloring depends functionally from the red and green coloring



- Blue coloring does not functionally depend from the red coloring



Theorem proving: Prove a formula holds true. Here:

Does "the blue coloring is functionally dependent from the red/red and green coloring" (as a formula) hold true?

For such general analysis tasks (wrt. **all** instances)
theorem proving is appropriate! See Demo.

Another Application: Compiler Validation

Problem: prove equivalence of source and target program

Example:

```
1:  y := 1                1: y := 1
2:  if z = x*x*x          2: R1 := x*x
3:    then y := x*x + y   3: R2 := R1*x
4:  endif                 4: jmpNE(z,R2,6)
                          5: y := R1+1
```

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

$$y_1 \approx 1 \wedge z_0 \approx x_0 * x_0 * x_0 \wedge y_3 \approx x_0 * x_0 + y_1$$
$$y_1' \approx 1 \wedge R1_2 \approx x_0' * x_0' \wedge R2_3 \approx R1_2 * x_0' \wedge z_0' \approx R2_3 \wedge y_5' \approx R1_2 + 1$$
$$\wedge\, x_0 \approx x_0' \wedge y_0 \approx y_0' \wedge z_0 \approx z_0' \quad \models \quad y_3 \approx y_5'$$

Motivation

Theorem proving is about . . .

Logics: Propositional, First-Order, Higher-Order, Modal, Description, . . .

Calculi and proof procedures: Resolution, DPLL, Tableaux, . . .

Systems: Interactive, Automated

Applications: Knowledge Representation, Verification, . . .

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Logic Based Knowledge Representation

90s: Modern Theory and Implementations, "A Basis for Applications"

2000s: Specializations for Applications

Motivation

In this lecture, theorem proving is about . . .

Logics: Propositional, **First-Order**, Higher-Order, Modal, Description, . . .

Calculi and proof procedures: **Resolution**, DPLL, Tableaux, . . .

Systems: Interactive, **Automated**

Applications: Knowledge Representation, Verification, . . .

Milestones

60s: Calculi: DPLL, Resolution, Model Elimination

70s: Logic Programming

80s: Logic Based Knowledge Representation

90s: Modern Theory and Implementations, "A Basis for Applications"

2000s: Specializations for Applications

Part II – Predicate Logic, from the viewpoint of FOTP

- Syntax and semantics of first-order predicate logic: a mathematical example
- Normal forms

A Mathematical Example

The sum of two continuous function is continuous.

Definition $f : \mathbb{R} \to \mathbb{R}$ is **continuous** at a, if for every $\varepsilon > 0$ there is a $\delta > 0$, such that for all x with $|x - a| < \delta$ it holds $|f(x) - f(a)| < \varepsilon$.

Proposition If f and g are continuous, so is their sum.

Proof Let $h = f + g$ assume $\varepsilon > 0$ given. With f and g continuous, there are δ_f and δ_g greater than 0 such that, if $|x - a| < \delta_f$, then $|f(x) - f(a)| < \varepsilon/2$ and, if $|x - a| < \delta_g$, then $|g(x) - g(a)| < \varepsilon/2$. Chose $\delta = \min(\delta_f, \delta_g)$. If $|x - a| < \delta$ then we approximate:

$$\begin{aligned}|h(x) - h(a)| &= |f(x) + g(x) - f(a) - g(a)| \\ &= |(f(x) - f(a)) + (g(x) - g(a))| \\ &\leq |f(x) - f(a)| + |g(x) - g(a)| < \varepsilon/2 + \varepsilon/2 = \varepsilon\end{aligned}$$

The Language of Predicate Logic

"f is continuous", expressed in first-order predicate logic:

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

Can pass this formula to a theorem prover?
What does it "mean" to the prover?

Predicate Logic Syntax

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

Variables ε, a, δ, 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, each of given arity, comprise a signature Σ.

Predicate Logic Semantics

Universe (aka Domain): Set U

Variables $\mapsto$ values in U (mapping is called "assignment")

Function symbols $\mapsto$ (total) functions over U

Predicate symbols $\mapsto$ relations over U

Boolean conectives $\mapsto$ the usual boolean functions

Quantifiers $\mapsto$ "for all ... holds", "there is a ..., such that"

Terms $\mapsto$ values in U

Formulas $\mapsto$ Boolean (Truth-) values

The underlying mathematical concept is that of a Σ-algebra.

Example

Let Σ_{PA} be the standard signature of Peano Arithmetic.
The standard interpretation for Peano Arithmetic then is:

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

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

Example

Values over $\mathbb{N}$ for sample terms and formulas:
Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$\begin{array}{lcl}
\mathbb{N}(\beta)(s(x)+s(0)) & = & 3 \\
\mathbb{N}(\beta)(x+y \approx s(y)) & = & \mathit{True} \\
\mathbb{N}(\beta)(\forall x, y(x+y \approx y+x)) & = & \mathit{True} \\
\mathbb{N}(\beta)(\forall z\, z \leq y) & = & \mathit{False} \\
\mathbb{N}(\beta)(\forall x \exists y\, x < y) & = & \mathit{True}
\end{array}$$

If ϕ is a closed formula, then, instead of $I(\phi) = \mathit{True}$ one writes $I \models \phi$
("I is a model of ϕ").
E.g. $\mathbb{N} \models \forall x \exists y\, x < y$

Axiomatizing the Real Numbers

In our proof problem, we have to "axiomatize" all those properties of the standard functions and predicate symbols that are needed to get a proof. There are only some of them here.
Addition and Subtraction:

$$\begin{array}{rcl}
x+y & = & y+x \\
(x+y)+z & = & x+(y+z) \\
x-y & = & x+(-y) \\
-(x+y) & = & (-x)+(-y)
\end{array}$$

Axiomatizing the Real Numbers

Ordering:

$$\begin{aligned} & \neg\, x < x \\ x < y \,\wedge\, y < z \;&\rightarrow\; x < z \\ & x \le x \\ x \le y \;&\leftrightarrow\; x < y \,\vee\, x = y \\ & x \le y \,\vee\, y < x \end{aligned}$$

divide by 2 and absolute values:

$$\begin{aligned} x/2 \le 0 \;&\rightarrow\; x \le 0 \\ x < z/2 \,\wedge\, y < z/2 \;&\rightarrow\; x + y < z \\ & |x+y| \le |x| + |y| \end{aligned}$$

Now one can prove:

$$\text{Axioms over } \mathbb{R} \wedge \text{continuous}(f) \wedge \text{continuous}(g) \quad \models \quad \text{continuous}(f+g)$$

It can even be proven fully automatically!

Algorithmic Problems

The following is a list of practically relevant problems:

Validity(F): $\models F$? (is F true in every interpretation?)

Satisfiability(F): F satisfiable?

Entailment(F,G): $F \models G$? (does F entail G?),

Model(A,F): $A \models F$?

Solve(A,F): find an assignment β such that $A, \beta \models F$

Solve(F): find a substitution σ such that $\models F\sigma$

Abduce(F): find G with "certain properties" such that G entails F

Different problems may require rather different methods! But ...

Refutational Theorem Proving

- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \rightarrow G$ is valid.
- Equivalently, we can prove that $\neg F$, i.e.$H \wedge \neg G$ is unsatisfiable.

This principle of "refutational theorem proving" is the basis of almost all automated theorem proving methods.

Normal Forms

Study of normal forms motivated by

- reduction of logical concepts,
- efficient data structures for theorem proving.

The main problem in first-order logic is the treatment of quantifiers. The subsequent normal form transformations are intended to eliminate many of them.

Prenex Normal Form

Prenex formulas have the form

$$Q_1x_1 \ldots Q_nx_n\ F,$$

where F is quantifier-free and $Q_i \in \{\forall, \exists\}$;
we call $Q_1x_1 \ldots Q_nx_n$ the **quantifier prefix** and F the **matrix** of the formula.

Prenex Normal Form

Computing prenex normal form by the rewrite relation $\Rightarrow_P$:

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

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

In the Example

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

$$\Rightarrow_P$$

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

$$\Rightarrow_P$$

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

$$\Rightarrow_P$$

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

$$\Rightarrow_P$$

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

Skolemization

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

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

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

In the Example

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

$$\Rightarrow_S$$

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

Skolemization

Together: $F \overset{*}{\Rightarrow}_P \underbrace{G}_{\text{prenex}} \overset{*}{\Rightarrow}_S \underbrace{H}_{\text{prenex, no } \exists}$

Theorem: The given and the final formula are equi-satisfiable.

Clausal Normal Form (Conjunctive Normal Form)

$$\begin{array}{rcl}
(F \leftrightarrow G) & \Rightarrow_K & (F \to G) \land (G \to F) \\
(F \to G) & \Rightarrow_K & (\neg F \lor G) \\
\neg(F \lor G) & \Rightarrow_K & (\neg F \land \neg G) \\
\neg(F \land G) & \Rightarrow_K & (\neg F \lor \neg G) \\
\neg\neg F & \Rightarrow_K & F \\
(F \land G) \lor H & \Rightarrow_K & (F \lor H) \land (G \lor H) \\
(F \land \top) & \Rightarrow_K & F \\
(F \land \bot) & \Rightarrow_K & \bot \\
(F \lor \top) & \Rightarrow_K & \top \\
(F \lor \bot) & \Rightarrow_K & F
\end{array}$$

These rules are to be applied modulo associativity and commutativity of $\land$ and $\lor$. The first five rules, plus the rule $(\neg Q)$, compute the **negation normal form** (NNF) of a formula.

In the Example

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

$$\Rightarrow_K$$

$$0 < d(\varepsilon, a) \ \lor\ \neg(0 < \varepsilon)$$

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

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

The Complete Picture

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

$N = \{C_1, \dots, C_k\}$ is called the **clausal (normal) form** (CNF) of F.
Note: the variables in the clauses are implicitly universally quantified.

Now we arrived at clause logic and the proof problem to show that the CNF of F is unsatisfiable. That is, to show there is no interpretation that satisfies the CNF of F.

Herbrand Theory

Some thoughts

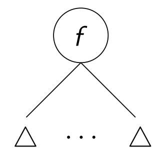
- Suppose we want to prove $H \models G$.
- Equivalently, we can prove that $F := H \land \neg G$ is unsatisfiable.
- We have seen how F can be syntactically simplified to clause form F' in a satisfiability preserving way.
- It remains to prove that F' is unsatisfiable.
- Does this mean that "**all** interpretations have to be searched"?
  No! It suffices to "search only through Herbrand interpretations"

Herbrand Theory

Significance: semantical basis for most theorem proving systems

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

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

$f_{\mathcal{A}}(\triangle, \dots, \triangle) =$ 

Herbrand Interpretations

In other words, **values are fixed** to be ground terms and **functions are fixed** to be the **term constructors**. Only predicate symbols $p/m \in \Pi$ may be freely interpreted as relations $p_{\mathcal{A}} \subseteq \mathrm{T}_{\Sigma}^{m}$.

Proposition
Every set of ground atoms I uniquely determines a Herbrand interpretation $\mathcal{A}$ via

$$(s_1, \dots, s_n) \in p_{\mathcal{A}} \quad :\Leftrightarrow \quad p(s_1, \dots, s_n) \in I$$

Thus we shall identify Herbrand interpretations (over Σ) with sets of Σ-ground atoms.

Herbrand Interpretations

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

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

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

Existence of Herbrand Models

A Herbrand interpretation I is called a **Herbrand model** of F iff $I \models F$.

Theorem

Let N be a set of Σ-clauses.

$$
\begin{aligned}
N \text{ is satisfiable} \quad &\Leftrightarrow \quad N \text{ has a Herbrand model (over } \Sigma) \\
&\Leftrightarrow \quad G_\Sigma(N) \text{ has a Herbrand model (over } \Sigma)
\end{aligned}
$$

where

$$G_\Sigma(N) = \{C\sigma \text{ ground clause} \mid C \in N,\ \sigma : X \rightarrow \mathrm{T}_\Sigma\}$$

is the set of **ground instances** of N.

Example of a G_Σ

For Σ_{Pres} one obtains for

$$C = (x < y) \vee (y \leq s(x))$$

the following ground instances:

$(0 < 0) \vee (0 \leq s(0))$

$(s(0) < 0) \vee (0 \leq s(s(0)))$

$\ldots$

$(s(0)+s(0) < s(0)+0) \vee (s(0)+0 \leq s(s(0)+s(0)))$

$\ldots$

Herbrand's Theorem

Theorem (Skolem-Herbrand-Theorem)

$\forall\phi$ is unsatisfiable iff some finite set of ground instances $\{\phi\gamma_1, \ldots, \phi\gamma_n\}$ is unsatisfiable

Applied to clause logic:

Theorem

Let N be a set of Σ-clauses.

$$
\begin{aligned}
N \text{ is unsatisfiable} \quad &\Leftrightarrow \quad G_\Sigma(N) \text{ has no Herbrand model (over } \Sigma) \\
&\Leftrightarrow \quad \text{there is a \textbf{finite} subset of } G_\Sigma(N) \\
& \quad \text{that has no Herbrand model (over } \Sigma)
\end{aligned}
$$

Significance: It's the core argument to show that there are complete (and sound) proof systems for first-order logic.

For instance, "Gilmore's Method".

Gilmore's Method - Based on Herbrand's Theorem

Preprocessing:

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

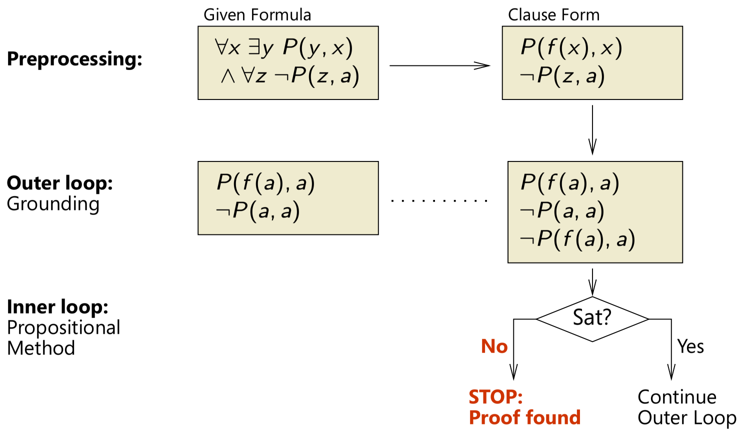
Given Formula: $\forall x\, \exists y\, P(y,x) \land \forall z\, \neg P(z,a)$ $\longrightarrow$ Clause Form: $P(f(x),x)$, $\neg P(z,a)$

Outer loop:
Grounding

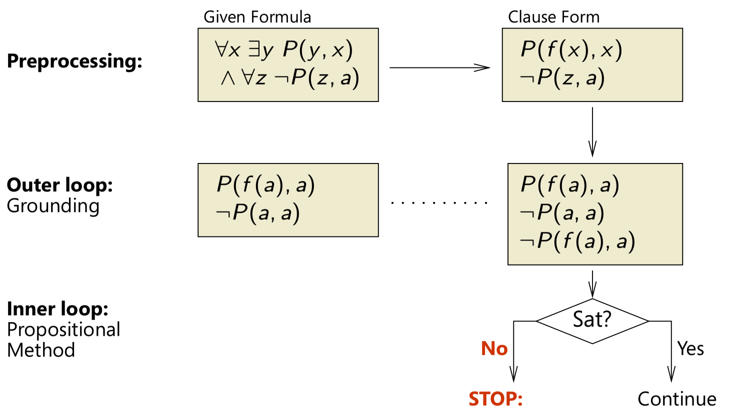
$P(f(a),a)$
$\neg P(a,a)$
. . . .
$P(f(a),a)$
$\neg P(a,a)$
$\neg P(f(a),a)$

Inner loop:
Propositional
Method

Gilmore's Method - Based on Herbrand's Theorem



Gilmore's Method - Based on Herbrand's Theorem



Problems:

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

Part III: Proof Systems - In Particular the Resolution Calculus

Two fundamental results limit what can be achieved:

Theorem (Gödel, 1929)
There are proof systems that enumerate all valid formulas of first-order predicate logic. (This is also a consequence of Herbrand's Theorem)

Theorem (Church/Turing, about 1935)
The validity problem of first-order logic formulas is undecidable.

(Thus, the model existence problem is undecidable, too.)

Automated theorem proving is oriented at the first, positive result.

But "model computation" is gaining increasingly importance.

Inference Systems and Proofs

Inference systems Γ (proof calculi) are sets of tuples

$$(F_1, \ldots, F_n, F_{n+1}), \;\; n \geq 0,$$

called **inferences** or **inference rules**, and written

$$\frac{\overbrace{F_1 \;\ldots\; F_n}^{\text{premises}}}{\underbrace{F_{n+1}}_{\text{conclusion}}} \; .$$

Clausal inference system: premises and conclusions are clauses. One also considers inference systems over other data structures.

Proofs

A **proof** in Γ of a formula F from a a set of formulas N (called **assumptions**) is a sequence $F_1, \ldots, F_k$ of formulas where

1. $F_k = F$,
2. for all $1 \leq i \leq k$: $F_i \in N$, or else there exists an inference $(F_{i_1}, \ldots, F_{i_{n_i}}, F_i)$ in Γ, such that $0 \leq i_j < i$, for $1 \leq j \leq n_i$.

Soundness and Completeness

Provability $\vdash_\Gamma$ of F from N in Γ:
$N \vdash_\Gamma F$ $:\Leftrightarrow$ there exists a proof Γ of F from N.
Γ is called **sound** $:\Leftrightarrow$

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

Γ is called **complete** $:\Leftrightarrow$

$$N \models F \quad \Rightarrow \quad N \vdash_\Gamma F$$

Γ is called **refutationally complete** $:\Leftrightarrow$

$$N \models \bot \quad \Rightarrow \quad N \vdash_\Gamma \bot$$

Soundness and Completeness

Proposition

1. Let Γ be sound. Then $N \vdash_\Gamma F \Rightarrow N \models F$
2. $N \vdash_\Gamma F \Rightarrow$ there exist $F_1, \ldots, F_n \in N$ s.t. $F_1, \ldots, F_n \vdash_\Gamma F$ (resembles compactness).

Proofs as Trees

markings	$\widehat{=}$	formulas		
leaves	$\widehat{=}$	assumptions and axioms		
other nodes	$\widehat{=}$	inferences: conclusion	$\widehat{=}$	ancestor
		premises	$\widehat{=}$	direct descendants

$$\frac{P(g(a,b)) \qquad \dfrac{\dfrac{\dfrac{P(f(a)) \lor Q(b) \qquad \dfrac{\dfrac{P(f(a)) \lor Q(b) \quad \neg P(f(a)) \lor \neg P(f(a)) \lor Q(b)}{\neg P(f(a)) \lor Q(b) \lor Q(b)}}{\neg P(f(a)) \lor Q(b)}}{Q(b) \lor Q(b)}}{Q(b)} \qquad \neg P(f(a)) \lor \neg Q(b)}{\neg P(g(a,b))}}{\bot}$$

The 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 Refutation

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 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.
Resolution is also refutationally complete.

Methods for First-Order Clause Logic

- Gilmore's method, see above (considered "naive" nowadays)
- The Resolution Calculus, see below
  The Resolution Calculus [Robinson 1965] (for first-order clause logic) is much better suited for automatization on a computer than earlier calculi:
  - Simpler (one single inference rule)
  - Less search space

There are other methods that are not based on Resolution (not treated here)

- Tableaux and connection methods, Model Elimination
- Instance Based Methods (not here - see my home page for tutorial)

Gilmore's Method vs. Versus Resolution

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

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

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

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

Redundancy concepts, e.g. **subsumption deletion**:

$$P(y,z) \quad \text{subsumes} \quad P(y,y) \vee Q(y,y)$$

Not available in Gilmore's method

First-Order Resolution through Instantiation

Idea: instantiate clauses to ground clauses:

$$P(z',z') \vee \neg Q(z) \xrightarrow{[a/z',\, f(a,b)/z]} P(a,a) \vee \neg Q(f(a,b))$$

$$\neg P(a,y) \xrightarrow{[a/y]} \neg P(a,a) \qquad \neg P(a,y) \xrightarrow{[b/y]} \neg P(a,b)$$

$$P(x',b) \vee Q(f(x',x)) \xrightarrow{[a/x',\, b/x]} P(a,b) \vee Q(f(a,b))$$

$$\frac{P(a,a) \vee \neg Q(f(a,b)) \qquad \neg P(a,a)}{\neg Q(f(a,b))} \qquad \frac{\neg P(a,b) \qquad P(a,b) \vee Q(f(a,b))}{Q(f(a,b))}$$

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

Bears ressemblance with Gilmore's method - not optimal.

First-Order Resolution through Instantiation

Problems

- More than one instance of a clause can participate in a proof.
- Even worse: There are infinitely many possible instances.

Observation

- Instantiation must produce complementary literals
  (so that inferences become possible).

Idea

- Do not instantiate more than necessary to get complementary literals.

First-Order Resolution

Idea: do not instantiate more than necessary:

$$P(z',z') \vee \neg Q(z) \xrightarrow{[a/z']} P(a,a) \vee \neg Q(z)$$

$$\neg P(a,y) \xrightarrow{[a/y]} \neg P(a,a) \qquad \neg P(a,y) \xrightarrow{[b/y]} \neg P(a,b)$$

$$P(x',b) \vee Q(f(x',x)) \xrightarrow{[a/x']} P(a,b) \vee Q(f(a,x))$$

$$\frac{P(a,a) \vee \neg Q(z) \qquad \neg P(a,a)}{\neg Q(z)} \qquad \frac{\neg P(a,b) \qquad P(a,b) \vee Q(f(a,x))}{Q(f(a,x))}$$

$$\neg Q(z) \xrightarrow{[f(a,x)/z]} \neg Q(f(a,x)) \qquad Q(f(a,x)) \longrightarrow Q(f(a,x))$$

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

Lifting Principle

Problem: Make closure under Resolution and Factorization of infinite sets of clauses as they arise from taking the (ground) instances of finitely many **first-order** clauses (with variables) effective and efficient.

Idea (Robinson 65):

- Resolution for first-order clauses:
- **Equality** of ground atoms is generalized to **unifiability** of general atoms;
- Only compute **most general** (minimal) unifiers.

Lifting Principle

Significance: The advantage of the method in (Robinson 65) compared with (Gilmore 60) is that unification enumerates only those instances of clauses that participate in an inference.

Moreover, clauses are not right away instantiated into ground clauses. Rather they are instantiated only as far as required for an inference. Inferences with non-ground clauses in general represent infinite sets of ground inferences which are computed simultaneously in a single step.

Resolution for First-Order Clauses

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

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

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

Example

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

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

Unification

A **substitution** σ is a mapping from variables to terms which is the identity almost everywhere.
Example: $\sigma = [f(a, x)/z,\ b/y]$

A substitutions can be **applied** to a term t, written as $t\sigma$.
Example, where σ is from above: $g(x, y, z)\sigma = g(x, b, f(a, x))$.

Let $E = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ (s_i, t_i terms or atoms) a multi-set of **equality problems.**

A substitution σ is called a **unifier** of E if $s_i\sigma = t_i\sigma$ for all $1 \le i \le n$.

If a unifier of E exists, then E is called **unifiable.**

Unification

A substitution σ is called **more general** than a substitution τ, denoted by $\sigma \leq \tau$, if there exists a substitution ρ such that $\rho \circ \sigma = \tau$, where $(\rho \circ \sigma)(x) := (x\sigma)\rho$ is the composition of σ and ρ as mappings.

If a unifier of E is more general than any other unifier of E, then we speak of a **most general unifier** of E, denoted by $\mathrm{mgu}(E)$.

Unification after Martelli/Montanari

$$
\begin{array}{rll}
t \doteq t, E & \Rightarrow_{MM} & E \\
f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n), E & \Rightarrow_{MM} & s_1 \doteq t_1, \ldots, s_n \doteq t_n, E \\
f(\ldots) \doteq g(\ldots), E & \Rightarrow_{MM} & \perp \\
x \doteq t, E & \Rightarrow_{MM} & x \doteq t, E[t/x] \\
 & & \text{if } x \in var(E), x \notin var(t) \\
x \doteq t, E & \Rightarrow_{MM} & \perp \\
 & & \text{if } x \neq t, x \in var(t) \\
t \doteq x, E & \Rightarrow_{MM} & x \doteq t, E \\
 & & \text{if } t \notin X
\end{array}
$$

Main Properties

If $E = x_1 \doteq u_1, \ldots, x_k \doteq u_k$, with x_i pairwise distinct, $x_i \notin var(u_j)$, then E is called (an equational problem) in **solved form** representing the solution $\sigma_E = [u_1/x_1, \ldots, u_k/x_k]$.

Proposition
If E is a solved form then σ_E is am mgu of E.

Main Properties

Theorem

1. If $E \Rightarrow_{MM} E'$ then σ is a (most general) unifier of E iff σ is a (most general) unifier of E'
2. If $E \Rightarrow^{*}_{MM} \perp$ then E is not unifiable.
3. If $E \Rightarrow^{*}_{MM} E'$ with E' in solved form, then $\sigma_{E'}$ is an mgu of E.

Theorem
E is unifiable if and only if there is a most general unifier σ of E, such that σ is idempotent and $dom(\sigma) \cup codom(\sigma) \subseteq var(E)$.

Problem: **exponential growth** of terms possible

Properties of Resolution

Theorem: Resolution is **sound**. That is, all derived formulas are logical consequences of the given ones

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 + Semantic Tree proof technique + Lifting Theorem

(This result can be considerably strengthened using other techniques)

Closure can be achieved by the "Given Clause Loop" on next slide.

The "Given Clause Loop"

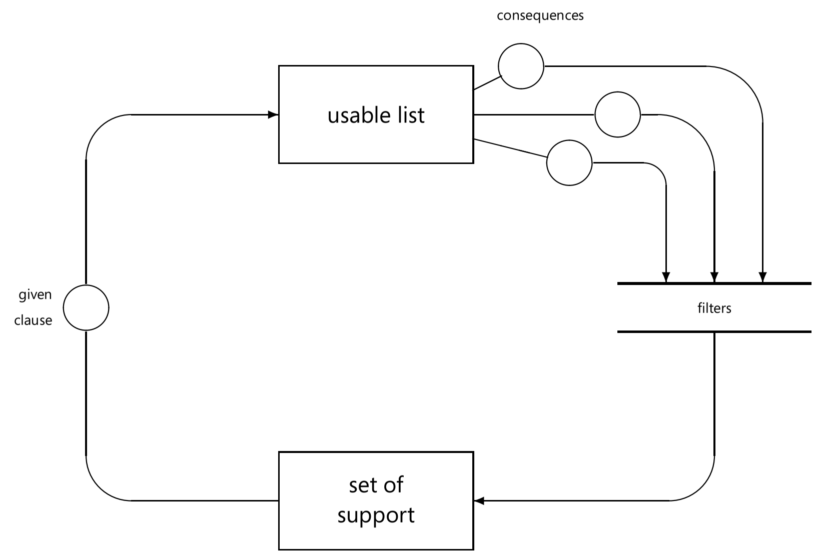
As used in the Otter theorem prover:
Lists of clauses maintained by the algorithm: `usable` and `sos`.
Initialize `sos` with the input clauses, `usable` empty.

Algorithm (straight from the Otter manual):

```
While (sos is not empty and no refutation has been found)
  1. Let given_clause be the 'lightest' clause in sos;
  2. Move given_clause from sos to usable;
  3. Infer and process new clauses using the inference rules in
     effect; each new clause must have the given_clause as
     one of its parents and members of usable as its other
     parents;  new clauses that pass the retention tests
     are appended to sos;
End of while loop.
```

Fairness: define clause weight e.g. as "depth + length" of clause.

The "Given Clause Loop" - Graphically



Part IV: Model Generation

Scenario: no "theorem" to prove, or a non-theorem
A model provides further information then
Why compute models?

Planning: Can be formalised as propositional satisfiability problem.
[Kautz& Selman, AAAI96; Dimopolous et al, ECP97]

Diagnosis: Minimal models of *abnormal* literals (circumscription).
[Reiter, AI87]

Databases: View materialisation, View Updates, Integrity Constraints.

Nonmonotonic reasoning: Various semantics (GCWA, Well-founded, Perfect, Stable,...), all based on minimal models. [Inoue et al, CADE 92]

Software Verification: Counterexamples to conjectured theorems.

Theorem proving: Counterexamples to conjectured theorems.
Finite models of quasigroups, (MGTP/G). [Fujita et al, IJCAI 93]

Part IV: Model Generation

Why compute models (cont'd)?

Natural Language Processing:

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

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

- Consistency checks ("Mia's husband loves Sally. She is not married.")

$$\begin{array}{ll} & BG\text{-}Knowledge \cup Discourse\_so\_far \not\models \neg New\_utterance \\ iff & BG\text{-}Knowledge \cup Discourse\_so\_far \cup New\_utterance \text{ is } \textbf{satisfiable} \end{array}$$

- Informativity checks ("Mia's husband loves Sally. She is married.")

$$\begin{array}{ll} & BG\text{-}Knowledge \cup Discourse\_so\_far \not\models New\_utterance \\ iff & BG\text{-}Knowledge \cup Discourse\_so\_far \cup \neg New\_utterance \text{ is } \textbf{satisfiable} \end{array}$$

Example - Group Theory

The following axioms specify a group

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

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

$e : 1$

$i :$

1	2	3	4	5	6
1	2	3	5	4	6

$* :$

	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

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:	$\{p(a) \vee f(x) = a\}$
Flattened:	$p(y) \vee f(x) = y \vee a \neq y$
Instances:	$p(1) \vee f(1) = 1 \vee a \neq 1$
	$p(2) \vee f(1) = 1 \vee a \neq 2$
	$p(1) \vee f(2) = 1 \vee a \neq 1$
	$p(2) \vee f(2) = 1 \vee a \neq 2$
Totality:	$a = 1 \vee a = 2$
	$f(1) = 1 \vee f(1) = 2$
	$f(2) = 1 \vee f(2) = 2$
Functionality:	$a \neq 1 \vee a \neq 2$
	$f(1) \neq 1 \vee f(1) \neq 2$
	$f(2) \neq 1 \vee f(2) \neq 2$

A model is obtained by setting the blue literals true

Further Considerations

Choice. There have been many inference systems developed. Which one is best suited for my application?

Local search space. Design small inference systems that allow for as little as inferences as possible.

Global redundancy elimination. In general there are many proofs of a given formula. Proof attempts that are "subsumed" by previous attempts should be pruned.

Efficient data structures. Determine as fast as possible the possible inferences.

Building-in theories. Specialized reasoning procedures for "data structures", like $\mathbb{R}$, $\mathbb{Z}$, lists, arrays, sets, etc.
(These can be axiomatized, but in general this leads to nowhere.)