

# Logic and Proof

Computer Science Tripos Part IB  
Michaelmas Term

Lawrence C Paulson

Computer Laboratory  
University of Cambridge

`lp15@cam.ac.uk`

Copyright © 2016 by Lawrence C. Paulson

## Introduction to Logic

Logic concerns **statements** in some **language**.

The language can be natural (English, Latin, . . .) or **formal**.

Some statements are **true**, others **false** or **meaningless**.

Logic concerns **relationships** between statements: consistency, entailment, . . .

Logical **proofs** model human reasoning (supposedly).



## Statements

Statements are declarative assertions:

Black is the colour of my true love's hair.

They are not greetings, questions or commands:

What is the colour of my true love's hair?

I wish my true love had hair.

Get a haircut!



## Schematic Statements

Now let the **variables**  $X, Y, Z, \dots$  range over 'real' objects

Black is the colour of  $X$ 's hair.

Black is the colour of  $Y$ .

$Z$  is the colour of  $Y$ .

Schematic statements can even express questions:

What things are black?



## Interpretations and Validity

An **interpretation** maps variables to real objects:

The interpretation  $Y \mapsto \text{coal}$  **satisfies** the statement

**Black is the colour of  $Y$ .**

but the interpretation  $Y \mapsto \text{strawberries}$  does not!

A statement  $A$  is **valid** if all interpretations satisfy  $A$ .



## Consistency, or Satisfiability

A set  $S$  of statements is **consistent** if some interpretation satisfies all elements of  $S$  at the same time. Otherwise  $S$  is **inconsistent**.

Examples of inconsistent sets:

$\{X \text{ part of } Y, Y \text{ part of } Z, X \text{ NOT part of } Z\}$

$\{n \text{ is a positive integer, } n \neq 1, n \neq 2, \dots\}$

**Satisfiable** means the same as consistent.

**Unsatisfiable** means the same as inconsistent.



## Entailment, or Logical Consequence

A set  $S$  of statements **entails**  $A$  if every interpretation that satisfies all elements of  $S$ , also satisfies  $A$ . We write  $S \models A$ .

$$\{X \text{ part of } Y, Y \text{ part of } Z\} \models X \text{ part of } Z$$

$$\{n \neq 1, n \neq 2, \dots\} \models n \text{ is NOT a positive integer}$$

$S \models A$  if and only if  $\{\neg A\} \cup S$  is inconsistent.

If  $S$  is inconsistent, then  $S \models A$  for any  $A$ .

$\models A$  if and only if  $A$  is valid, if and only if  $\{\neg A\}$  is inconsistent.



## Inference: Proving a Statement

We want to show that  $A$  is valid. We can't test infinitely many cases.

Let  $\{A_1, \dots, A_n\} \models B$ . If  $A_1, \dots, A_n$  are true then  $B$  must be true.

Write this as the **inference rule**

$$\frac{A_1 \quad \dots \quad A_n}{B}$$

We can use inference rules to construct finite proofs!



## Schematic Inference Rules

$$\frac{X \text{ part of } Y \quad Y \text{ part of } Z}{X \text{ part of } Z}$$

- A proof is correct if it has the **right syntactic form**, regardless of
- Whether the conclusion is desirable
- Whether the premises or conclusion are true
- Who (or what) created the proof



## Why Should we use a Formal Language?

Consider this ‘definition’: (Berry’s paradox)

The smallest positive integer not definable using nine words

Greater than The number of atoms in the Milky Way galaxy

This number is so large, it is greater than itself!

- A formal language prevents ambiguity.



## Survey of Formal Logics

**propositional logic** is traditional **boolean algebra**.

**first-order logic** can say **for all** and **there exists**.

**higher-order logic** reasons about sets and functions.

**modal/temporal logics** reason about what **must**, or **may**, happen.

**type theories** support **constructive** mathematics.

All have been used to prove correctness of computer systems.



## Syntax of Propositional Logic

$P, Q, R, \dots$  propositional letter

**t** true

**f** false

$\neg A$  not  $A$

$A \wedge B$   $A$  and  $B$

$A \vee B$   $A$  or  $B$

$A \rightarrow B$  if  $A$  then  $B$

$A \leftrightarrow B$   $A$  if and only if  $B$



## Semantics of Propositional Logic

$\neg$ ,  $\wedge$ ,  $\vee$ ,  $\rightarrow$  and  $\leftrightarrow$  are **truth-functional**: functions of their operands.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1



## Interpretations of Propositional Logic

An **interpretation** is a function from the propositional letters to  $\{1, 0\}$ .

Interpretation  $I$  **satisfies** a formula  $A$  if it evaluates to 1 (true).

Write  $\models_I A$

$A$  is **valid** (a **tautology**) if every interpretation satisfies  $A$ .

Write  $\models A$

$S$  is **satisfiable** if some interpretation satisfies every formula in  $S$ .



## Implication, Entailment, Equivalence

$A \rightarrow B$  means simply  $\neg A \vee B$ .

$A \models B$  means if  $\models_I A$  then  $\models_I B$  for every interpretation  $I$ .

$A \models B$  if and only if  $\models A \rightarrow B$ .

### Equivalence

$A \simeq B$  means  $A \models B$  and  $B \models A$ .

$A \simeq B$  if and only if  $\models A \leftrightarrow B$ .



## Equivalences

$$A \wedge A \simeq A$$

$$A \wedge B \simeq B \wedge A$$

$$(A \wedge B) \wedge C \simeq A \wedge (B \wedge C)$$

$$A \vee (B \wedge C) \simeq (A \vee B) \wedge (A \vee C)$$

$$A \wedge \mathbf{f} \simeq \mathbf{f}$$

$$A \wedge \mathbf{t} \simeq A$$

$$A \wedge \neg A \simeq \mathbf{f}$$

Dual versions: exchange  $\wedge$  with  $\vee$  and  $\mathbf{t}$  with  $\mathbf{f}$  in any equivalence

## Negation Normal Form

1. Get rid of  $\leftrightarrow$  and  $\rightarrow$ , leaving just  $\wedge$ ,  $\vee$ ,  $\neg$ :

$$A \leftrightarrow B \simeq (A \rightarrow B) \wedge (B \rightarrow A)$$

$$A \rightarrow B \simeq \neg A \vee B$$

2. Push negations in, using de Morgan's laws:

$$\neg\neg A \simeq A$$

$$\neg(A \wedge B) \simeq \neg A \vee \neg B$$

$$\neg(A \vee B) \simeq \neg A \wedge \neg B$$



## From NNF to Conjunctive Normal Form

3. Push disjunctions in, using distributive laws:

$$A \vee (B \wedge C) \simeq (A \vee B) \wedge (A \vee C)$$

$$(B \wedge C) \vee A \simeq (B \vee A) \wedge (C \vee A)$$

4. Simplify:

- Delete any disjunction containing  $P$  and  $\neg P$
- Delete any disjunction that includes another: for example, in  $(P \vee Q) \wedge P$ , delete  $P \vee Q$ .
- Replace  $(P \vee A) \wedge (\neg P \vee A)$  by  $A$



## Converting a Non-Tautology to CNF

$$P \vee Q \rightarrow Q \vee R$$

1. Elim  $\rightarrow$ :  $\neg(P \vee Q) \vee (Q \vee R)$
2. Push  $\neg$  in:  $(\neg P \wedge \neg Q) \vee (Q \vee R)$
3. Push  $\vee$  in:  $(\neg P \vee Q \vee R) \wedge (\neg Q \vee Q \vee R)$
4. Simplify:  $\neg P \vee Q \vee R$

Not a tautology: try  $P \mapsto \mathbf{t}$ ,  $Q \mapsto \mathbf{f}$ ,  $R \mapsto \mathbf{f}$



## Tautology checking using CNF

$$((P \rightarrow Q) \rightarrow P) \rightarrow P$$

1. Elim  $\rightarrow$ :  $\neg[\neg(\neg P \vee Q) \vee P] \vee P$
2. Push  $\neg$  in:  $[\neg\neg(\neg P \vee Q) \wedge \neg P] \vee P$   
 $[(\neg P \vee Q) \wedge \neg P] \vee P$
3. Push  $\vee$  in:  $(\neg P \vee Q \vee P) \wedge (\neg P \vee P)$
4. Simplify:  $\mathbf{t} \wedge \mathbf{t}$   
 $\mathbf{t}$  *It's a tautology!*

## A Simple Proof System

### *Axiom Schemes*

$$\text{K} \quad A \rightarrow (B \rightarrow A)$$

$$\text{S} \quad (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$$

$$\text{DN} \quad \neg\neg A \rightarrow A$$

### *Inference Rule: Modus Ponens*

$$\frac{A \rightarrow B \quad A}{B}$$

## A Simple (?) Proof of $A \rightarrow A$

$$\begin{aligned} & (A \rightarrow ((D \rightarrow A) \rightarrow A)) \rightarrow & (1) \\ & ((A \rightarrow (D \rightarrow A)) \rightarrow (A \rightarrow A)) \quad \text{by S} \\ & A \rightarrow ((D \rightarrow A) \rightarrow A) \quad \text{by K} & (2) \\ & (A \rightarrow (D \rightarrow A)) \rightarrow (A \rightarrow A) \quad \text{by MP, (1), (2)} & (3) \\ & A \rightarrow (D \rightarrow A) \quad \text{by K} & (4) \\ & A \rightarrow A \quad \text{by MP, (3), (4)} & (5) \end{aligned}$$

## Some Facts about Deducibility

$A$  is **deducible from** the set  $S$  if there is a finite proof of  $A$  starting from elements of  $S$ . Write  $S \vdash A$ .

**Soundness Theorem.** If  $S \vdash A$  then  $S \models A$ .

**Completeness Theorem.** If  $S \models A$  then  $S \vdash A$ .

**Deduction Theorem.** If  $S \cup \{A\} \vdash B$  then  $S \vdash A \rightarrow B$ .



## Gentzen's Natural Deduction Systems

The context of **assumptions** may vary.

Each logical connective is defined **independently**.

The **introduction** rule for  $\wedge$  shows how to deduce  $A \wedge B$ :

$$\frac{A \quad B}{A \wedge B}$$

The **elimination** rules for  $\wedge$  shows what to deduce **from**  $A \wedge B$ :

$$\frac{A \wedge B}{A} \quad \frac{A \wedge B}{B}$$



## The Sequent Calculus

Sequent  $A_1, \dots, A_m \Rightarrow B_1, \dots, B_n$  means,

if  $A_1 \wedge \dots \wedge A_m$  then  $B_1 \vee \dots \vee B_n$

$A_1, \dots, A_m$  are **assumptions**;  $B_1, \dots, B_n$  are **goals**

$\Gamma$  and  $\Delta$  are **sets** in  $\Gamma \Rightarrow \Delta$

$A, \Gamma \Rightarrow A, \Delta$  is trivially true (and is called a **basic sequent**).



## Sequent Calculus Rules

$$\frac{\Gamma \Rightarrow \Delta, A \quad A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta} \text{ (cut)}$$

$$\frac{\Gamma \Rightarrow \Delta, A}{\neg A, \Gamma \Rightarrow \Delta} \text{ } (\neg l)$$

$$\frac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \text{ } (\neg r)$$

$$\frac{A, B, \Gamma \Rightarrow \Delta}{A \wedge B, \Gamma \Rightarrow \Delta} \text{ } (\wedge l)$$

$$\frac{\Gamma \Rightarrow \Delta, A \quad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \text{ } (\wedge r)$$

## More Sequent Calculus Rules

$$\frac{A, \Gamma \Rightarrow \Delta \quad B, \Gamma \Rightarrow \Delta}{A \vee B, \Gamma \Rightarrow \Delta} (\vee l)$$

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B} (\vee r)$$

$$\frac{\Gamma \Rightarrow \Delta, A \quad B, \Gamma \Rightarrow \Delta}{A \rightarrow B, \Gamma \Rightarrow \Delta} (\rightarrow l)$$

$$\frac{A, \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \rightarrow B} (\rightarrow r)$$



## Easy Sequent Calculus Proofs

$$\begin{array}{c}
 \overline{A, B \Rightarrow A} \\
 \hline
 A \wedge B \Rightarrow A \quad (\wedge l) \\
 \hline
 \Rightarrow (A \wedge B) \rightarrow A \quad (\rightarrow r)
 \end{array}$$

$$\begin{array}{c}
 \overline{A, B \Rightarrow B, A} \\
 \hline
 A \Rightarrow B, B \rightarrow A \quad (\rightarrow r) \\
 \hline
 \Rightarrow A \rightarrow B, B \rightarrow A \quad (\rightarrow r) \\
 \hline
 \Rightarrow (A \rightarrow B) \vee (B \rightarrow A) \quad (\vee r)
 \end{array}$$

### Part of a Distributive Law

$$\begin{array}{c}
 \frac{\overline{A \Rightarrow A, B} \quad \frac{\overline{B, C \Rightarrow A, B}}{B \wedge C \Rightarrow A, B} (\wedge l)}{A \vee (B \wedge C) \Rightarrow A, B} (\vee l) \\
 \frac{A \vee (B \wedge C) \Rightarrow A, B}{A \vee (B \wedge C) \Rightarrow A \vee B} (\vee r) \quad \text{similar} \\
 \hline
 A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C) \quad (\wedge r)
 \end{array}$$

Second subtree proves  $A \vee (B \wedge C) \Rightarrow A \vee C$  similarly

## A Failed Proof

$$\begin{array}{c}
 \frac{A \Rightarrow B, C \quad \overline{B \Rightarrow B, C}}{A \vee B \Rightarrow B, C} \quad (\vee l) \\
 \frac{A \vee B \Rightarrow B, C}{A \vee B \Rightarrow B \vee C} \quad (\vee r) \\
 \hline
 \Rightarrow (A \vee B) \rightarrow (B \vee C) \quad (\rightarrow r)
 \end{array}$$

$A \mapsto \mathbf{t}, B \mapsto \mathbf{f}, C \mapsto \mathbf{f}$  falsifies unproved sequent!

## Outline of First-Order Logic

Reasons about **functions** and **relations** over a set of **individuals**:

$$\frac{\text{father}(\text{father}(x)) = \text{father}(\text{father}(y))}{\text{cousin}(x, y)}$$

Reasons about **all** and **some** individuals:

$$\frac{\text{All men are mortal} \quad \text{Socrates is a man}}{\text{Socrates is mortal}}$$

Cannot reason about **all functions** or **all relations**, etc.



## Function Symbols; Terms

Each **function symbol** stands for an  $n$ -place function.

A **constant symbol** is a 0-place function symbol.

A **variable** ranges over all individuals.

A **term** is a variable, constant or a function application

$$f(t_1, \dots, t_n)$$

where  $f$  is an  $n$ -place function symbol and  $t_1, \dots, t_n$  are terms.

We choose the language, adopting any desired function symbols.



## Relation Symbols; Formulae

Each **relation symbol** stands for an  $n$ -place relation.

**Equality** is the 2-place relation symbol  $=$

An **atomic formula** has the form  $R(t_1, \dots, t_n)$  where  $R$  is an  $n$ -place relation symbol and  $t_1, \dots, t_n$  are terms.

A **formula** is built up from atomic formulæ using  $\neg$ ,  $\wedge$ ,  $\vee$ , and so forth.

Later, we can add **quantifiers**.



## The Power of Quantifier-Free FOL

It is surprisingly expressive, if we include strong induction rules.

We can easily prove the equivalence of mathematical functions:

$$p(z, 0) = 1$$

$$q(z, 1) = z$$

$$p(z, n + 1) = p(z, n) \times z \qquad q(z, 2 \times n) = q(z \times z, n)$$

$$q(z, 2 \times n + 1) = q(z \times z, n) \times z$$

The prover ACL2 uses this logic to do major hardware proofs.



## Universal and Existential Quantifiers

$\forall x A$  for all  $x$ , the formula  $A$  holds

$\exists x A$  there exists  $x$  such that  $A$  holds

Syntactic variations:

$\forall x y z A$  abbreviates  $\forall x \forall y \forall z A$

$\forall z . A \wedge B$  is an alternative to  $\forall z (A \wedge B)$

The variable  $x$  is **bound** in  $\forall x A$ ; compare with  $\int f(x) dx$



## The Expressiveness of Quantifiers

All men are mortal:

$$\forall x (\text{man}(x) \rightarrow \text{mortal}(x))$$

All mothers are female:

$$\forall x \text{female}(\text{mother}(x))$$

There exists a unique  $x$  such that  $A$ , sometimes written  $\exists!x A$

$$\exists x [A(x) \wedge \forall y (A(y) \rightarrow y = x)]$$

## The Point of Semantics

We have to attach meanings to symbols like  $1$ ,  $+$ ,  $<$ , etc.

Why is this necessary? Why can't  $1$  just mean  $1$ ??

The point is that mathematics derives its flexibility from allowing different interpretations of symbols.

- A **group** has a unit  $1$ , a product  $x \cdot y$  and inverse  $x^{-1}$ .
- In the most important uses of groups,  $1$  isn't a number but a 'unit permutation', 'unit rotation', etc.



## Constants: Interpreting mortal(Socrates)

An interpretation  $\mathcal{I} = (D, I)$  defines the **semantics** of a first-order language.

$D$  is a non-empty set, called the **domain** or **universe**.

$I$  maps symbols to 'real' elements, functions and relations:

$c$  a **constant** symbol  $I[c] \in D$

$f$  an  $n$ -place **function** symbol  $I[f] \in D^n \rightarrow D$

$P$  an  $n$ -place **relation** symbol  $I[P] \in D^n \rightarrow \{1, 0\}$



## Variables: Interpreting $\text{father}(y)$

A **valuation**  $V : \text{Var} \rightarrow D$  supplies the values of free variables.

$V$  and  $\mathcal{I}$  together determine the value of any term  $t$ , by recursion.

This value is written  $\mathcal{I}_V[t]$ , and here are the recursion rules:

$$\mathcal{I}_V[x] \stackrel{\text{def}}{=} V(x) \quad \text{if } x \text{ is a variable}$$

$$\mathcal{I}_V[c] \stackrel{\text{def}}{=} I[c]$$

$$\mathcal{I}_V[f(t_1, \dots, t_n)] \stackrel{\text{def}}{=} I[f](\mathcal{I}_V[t_1], \dots, \mathcal{I}_V[t_n])$$



## Tarski's Truth-Definition

An interpretation  $\mathcal{I}$  and valuation function  $V$  similarly specify the truth value (1 or 0) of any formula  $A$ .

**Quantifiers** are the only problem, as they bind variables.

$V\{a/x\}$  is the valuation that maps  $x$  to  $a$  and is otherwise like  $V$ .

With the help of  $V\{a/x\}$ , we now formally define  $\models_{\mathcal{I}, V} A$ , the truth value of  $A$ .



## The Meaning of Truth—In FOL!

For interpretation  $\mathcal{I}$  and valuation  $V$ , define  $\models_{\mathcal{I}, V}$  by recursion.

$\models_{\mathcal{I}, V} P(t)$	if $I[P](\mathcal{I}_V[t])$ equals 1 (is true)
$\models_{\mathcal{I}, V} t = u$	if $\mathcal{I}_V[t]$ equals $\mathcal{I}_V[u]$
$\models_{\mathcal{I}, V} A \wedge B$	if $\models_{\mathcal{I}, V} A$ and $\models_{\mathcal{I}, V} B$
$\models_{\mathcal{I}, V} \exists x A$	if $\models_{\mathcal{I}, V\{m/x\}} A$ holds for some $m \in D$

Finally, we define

$\models_{\mathcal{I}} A$	if $\models_{\mathcal{I}, V} A$ holds for all $V$ .
---------------------------	---

A **closed** formula  $A$  is **satisfiable** if  $\models_{\mathcal{I}} A$  for some  $\mathcal{I}$ .



## Free vs Bound Variables

All occurrences of  $x$  in  $\forall x A$  and  $\exists x A$  are **bound**

An occurrence of  $x$  is **free** if it is not bound:

$$\forall y \exists z R(y, z, f(y, x))$$

In this formula,  $y$  and  $z$  are bound while  $x$  is free.

We may **rename** bound variables without affecting the meaning:

$$\forall w \exists z' R(w, z', f(w, x))$$



## Substitution for Free Variables

$A[t/x]$  means **substitute  $t$  for  $x$  in  $A$** :

$$(B \wedge C)[t/x] \text{ is } B[t/x] \wedge C[t/x]$$

$$(\forall x B)[t/x] \text{ is } \forall x B$$

$$(\forall y B)[t/x] \text{ is } \forall y B[t/x] \quad (x \neq y)$$

$$(P(u))[t/x] \text{ is } P(u[t/x])$$

When substituting  $A[t/x]$ , no variable of  $t$  may be bound in  $A$ !

Example:  $(\forall y (x = y)) [y/x]$  **is not equivalent to**  $\forall y (y = y)$



## Some Equivalences for Quantifiers

$$\neg(\forall x A) \simeq \exists x \neg A$$

$$\forall x A \simeq \forall x A \wedge A[t/x]$$

$$(\forall x A) \wedge (\forall x B) \simeq \forall x (A \wedge B)$$

But we do not have  $(\forall x A) \vee (\forall x B) \simeq \forall x (A \vee B)$ .

Dual versions: exchange  $\forall$  with  $\exists$  and  $\wedge$  with  $\vee$



## Further Quantifier Equivalences

These hold only if  $x$  is not free in  $B$ .

$$(\forall x A) \wedge B \simeq \forall x (A \wedge B)$$

$$(\forall x A) \vee B \simeq \forall x (A \vee B)$$

$$(\forall x A) \rightarrow B \simeq \exists x (A \rightarrow B)$$

These let us expand or contract a quantifier's scope.



## Reasoning by Equivalences

$$\begin{aligned}\exists x (x = a \wedge P(x)) &\simeq \exists x (x = a \wedge P(a)) \\ &\simeq \exists x (x = a) \wedge P(a) \\ &\simeq P(a)\end{aligned}$$

$$\begin{aligned}\exists z (P(z) \rightarrow P(a) \wedge P(b)) \\ &\simeq \forall z P(z) \rightarrow P(a) \wedge P(b) \\ &\simeq \forall z P(z) \wedge P(a) \wedge P(b) \rightarrow P(a) \wedge P(b) \\ &\simeq \mathbf{t}\end{aligned}$$



## Sequent Calculus Rules for $\forall$

$$\frac{A[t/x], \Gamma \Rightarrow \Delta}{\forall x A, \Gamma \Rightarrow \Delta} (\forall l) \qquad \frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \forall x A} (\forall r)$$

Rule  $(\forall l)$  can create many instances of  $\forall x A$

Rule  $(\forall r)$  holds **provided**  $x$  is not free in the conclusion!

**Not** allowed to prove

$$\frac{\overline{P(y) \Rightarrow P(y)}}{P(y) \Rightarrow \forall y P(y)} (\forall r)$$

**This is nonsense!**



## A Simple Example of the $\forall$ Rules

$$\frac{\frac{\overline{P(f(y)) \Rightarrow P(f(y))}}{\forall x P(x) \Rightarrow P(f(y))} (\forall l)}{\forall x P(x) \Rightarrow \forall y P(f(y))} (\forall r)$$

## A Not-So-Simple Example of the $\forall$ Rules

$$\begin{array}{c}
 \frac{\overline{P \Rightarrow Q(y), P} \quad \overline{P, Q(y) \Rightarrow Q(y)}}{P, P \rightarrow Q(y) \Rightarrow Q(y)} (\rightarrow l) \\
 \hline
 P, P \rightarrow Q(y) \Rightarrow Q(y) \quad (\forall l) \\
 \hline
 P, \forall x (P \rightarrow Q(x)) \Rightarrow Q(y) \quad (\forall r) \\
 \hline
 P, \forall x (P \rightarrow Q(x)) \Rightarrow \forall y Q(y) \quad (\rightarrow r) \\
 \hline
 \forall x (P \rightarrow Q(x)) \Rightarrow P \rightarrow \forall y Q(y)
 \end{array}$$

In  $(\forall l)$ , we must replace  $x$  by  $y$ .



## Sequent Calculus Rules for $\exists$

$$\frac{A, \Gamma \Rightarrow \Delta}{\exists x A, \Gamma \Rightarrow \Delta} (\exists l) \qquad \frac{\Gamma \Rightarrow \Delta, A[t/x]}{\Gamma \Rightarrow \Delta, \exists x A} (\exists r)$$

Rule  $(\exists l)$  holds **provided**  $x$  is not free in the conclusion!

Rule  $(\exists r)$  can create many instances of  $\exists x A$

For example, to prove this counter-intuitive formula:

$$\exists z (P(z) \rightarrow P(a) \wedge P(b))$$



## Part of the $\exists$ Distributive Law

$$\begin{array}{c}
 \frac{\frac{\overline{P(x) \Rightarrow P(x), Q(x)}}{P(x) \Rightarrow P(x) \vee Q(x)} (\vee r)}{P(x) \Rightarrow \exists y (P(y) \vee Q(y))} (\exists r) \\
 \frac{\frac{\overline{P(x) \Rightarrow \exists y (P(y) \vee Q(y))}}{\exists x P(x) \Rightarrow \exists y (P(y) \vee Q(y))} (\exists l)}{\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists y (P(y) \vee Q(y))} (\exists l)
 \end{array}
 \quad
 \begin{array}{c}
 \text{similar} \\
 \frac{\overline{\exists x Q(x) \Rightarrow \exists y \dots}}{\exists x Q(x) \Rightarrow \exists y \dots} (\exists l) \\
 \frac{\exists x Q(x) \Rightarrow \exists y \dots}{\exists x P(x) \vee \exists x Q(x) \Rightarrow \exists y (P(y) \vee Q(y))} (\vee l)
 \end{array}$$

Second subtree proves  $\exists x Q(x) \Rightarrow \exists y (P(y) \vee Q(y))$  similarly

In  $(\exists r)$ , we must replace  $y$  by  $x$ .



## A Failed Proof

$$\begin{array}{rcl}
 & \dfrac{P(x), Q(y) \Rightarrow P(x) \wedge Q(x)}{P(x), Q(y) \Rightarrow \exists z (P(z) \wedge Q(z))} & (\exists r) \\
 & \dfrac{P(x), Q(y) \Rightarrow \exists z (P(z) \wedge Q(z))}{P(x), \exists x Q(x) \Rightarrow \exists z (P(z) \wedge Q(z))} & (\exists l) \\
 & \dfrac{P(x), \exists x Q(x) \Rightarrow \exists z (P(z) \wedge Q(z))}{\exists x P(x), \exists x Q(x) \Rightarrow \exists z (P(z) \wedge Q(z))} & (\exists l) \\
 & \dfrac{\exists x P(x), \exists x Q(x) \Rightarrow \exists z (P(z) \wedge Q(z))}{\exists x P(x) \wedge \exists x Q(x) \Rightarrow \exists z (P(z) \wedge Q(z))} & (\wedge l)
 \end{array}$$

We cannot use  $(\exists l)$  twice with the same variable

This attempt renames the  $x$  in  $\exists x Q(x)$ , to get  $\exists y Q(y)$



## Clause Form

**Clause**: a disjunction of **literals**

$$\neg K_1 \vee \dots \vee \neg K_m \vee L_1 \vee \dots \vee L_n$$

Set notation:  $\{\neg K_1, \dots, \neg K_m, L_1, \dots, L_n\}$

Kowalski notation:  $K_1, \dots, K_m \rightarrow L_1, \dots, L_n$

$$L_1, \dots, L_n \leftarrow K_1, \dots, K_m$$

Empty clause:  $\{\}$  or  $\square$

Empty clause is equivalent to **f**, meaning **contradiction!**



## Outline of Clause Form Methods

To prove  $\bar{A}$ , obtain a contradiction from  $\neg\bar{A}$ :

1. Translate  $\neg\bar{A}$  into CNF as  $A_1 \wedge \dots \wedge A_m$
2. This is the set of clauses  $A_1, \dots, A_m$
3. Transform the clause set, **preserving consistency**

Deducing the **empty clause** refutes  $\neg\bar{A}$ .

An empty **clause set** (all clauses deleted) means  $\neg\bar{A}$  is satisfiable.

The basis for **SAT solvers** and **resolution provers**.



## The Davis-Putnam-Logeman-Loveland Method

1. Delete tautological clauses:  $\{P, \neg P, \dots\}$
2. For each unit clause  $\{L\}$ ,
  - delete all clauses containing  $L$
  - delete  $\neg L$  from all clauses
3. Delete all clauses containing **pure literals**
4. Perform a **case split** on some literal; **stop** if a model is found

DPLL is a **decision procedure**: it finds a contradiction or a model.



## DPLL on a Non-Tautology

Consider  $P \vee Q \rightarrow Q \vee R$

Clauses are  $\{P, Q\}$   $\{\neg Q\}$   $\{\neg R\}$

$\{P, Q\}$   $\{\neg Q\}$   $\{\neg R\}$  initial clauses

$\{P\}$   $\{\neg R\}$  unit  $\neg Q$

$\{\neg R\}$  unit  $P$  (also pure)

unit  $\neg R$  (also pure)

**All clauses deleted!** Clauses satisfiable by  $P \mapsto \mathbf{t}$ ,  $Q \mapsto \mathbf{f}$ ,  $R \mapsto \mathbf{f}$



## Example of a Case Split on $P$

$\{\neg Q, R\}$   $\{\neg R, P\}$   $\{\neg R, Q\}$   $\{\neg P, Q, R\}$   $\{P, Q\}$   $\{\neg P, \neg Q\}$

$\{\neg Q, R\}$   $\{\neg R, Q\}$   $\{Q, R\}$   $\{\neg Q\}$  if  $P$  is true

$\{\neg R\}$   $\{R\}$  unit  $\neg Q$

$\{\}$  unit  $R$

$\{\neg Q, R\}$   $\{\neg R\}$   $\{\neg R, Q\}$   $\{Q\}$  if  $P$  is false

$\{\neg Q\}$   $\{Q\}$  unit  $\neg R$

$\{\}$  unit  $\neg Q$

Both cases yield contradictions: the clauses are **inconsistent!**



## SAT solvers in the Real World

- Progressed from joke to killer technology in 10 years.
- Princeton's zChaff has solved problems with more than one million variables and 10 million clauses.
- Applications include finding bugs in device drivers (Microsoft's SLAM project).
- SMT solvers (satisfiability modulo theories) extend SAT solving to handle arithmetic, arrays and bit vectors.



## The Resolution Rule

From  $B \vee A$  and  $\neg B \vee C$  infer  $A \vee C$

In set notation,

$$\frac{\{B, A_1, \dots, A_m\} \quad \{\neg B, C_1, \dots, C_n\}}{\{A_1, \dots, A_m, C_1, \dots, C_n\}}$$

Some special cases: (remember that  $\square$  is just  $\{\}$ )

$$\frac{\{B\} \quad \{\neg B, C_1, \dots, C_n\}}{\{C_1, \dots, C_n\}}$$

$$\frac{\{B\} \quad \{\neg B\}}{\square}$$



## Simple Example: Proving $P \wedge Q \rightarrow Q \wedge P$

Hint: use  $\neg(A \rightarrow B) \simeq A \wedge \neg B$

1. Negate!  $\neg[P \wedge Q \rightarrow Q \wedge P]$

2. Push  $\neg$  in:  $(P \wedge Q) \wedge \neg(Q \wedge P)$

$(P \wedge Q) \wedge (\neg Q \vee \neg P)$

Clauses:  $\{P\}$   $\{Q\}$   $\{\neg Q, \neg P\}$

Resolve  $\{P\}$  and  $\{\neg Q, \neg P\}$  getting  $\{\neg Q\}$ .

Resolve  $\{Q\}$  and  $\{\neg Q\}$  getting  $\square$ : we have refuted the negation.



## Another Example

Refute  $\neg[(P \vee Q) \wedge (P \vee R) \rightarrow P \vee (Q \wedge R)]$

From  $(P \vee Q) \wedge (P \vee R)$ , get clauses  $\{P, Q\}$  and  $\{P, R\}$ .

From  $\neg[P \vee (Q \wedge R)]$  get clauses  $\{\neg P\}$  and  $\{\neg Q, \neg R\}$ .

Resolve  $\{\neg P\}$  and  $\{P, Q\}$  getting  $\{Q\}$ .

Resolve  $\{\neg P\}$  and  $\{P, R\}$  getting  $\{R\}$ .

Resolve  $\{Q\}$  and  $\{\neg Q, \neg R\}$  getting  $\{\neg R\}$ .

Resolve  $\{R\}$  and  $\{\neg R\}$  getting  $\square$ , contradiction.



## The Saturation Algorithm

At start, all clauses are **passive**. None are **active**.

1. Transfer a clause (**current**) from **passive** to **active**.
2. Form all resolvents between **current** and an **active** clause.
3. Use new clauses to simplify both **passive** and **active**.
4. Put the new clauses into **passive**.

Repeat until **contradiction** found or **passive** becomes empty.



## Heuristics and Hacks for Resolution

**Orderings** to focus the search on specific literals

**Subsumption**, or deleting redundant clauses

**Indexing**: elaborate data structures for speed

**Preprocessing**: removing tautologies, symmetries . . .

**Weighting**: giving priority to “good” clauses over those containing unwanted constants



## Reducing FOL to Propositional Logic

**NNF :** Eliminate all connectives except  $\vee$ ,  $\wedge$  and  $\neg$

**Skolemize:** Remove quantifiers, preserving **consistency**

**Herbrand models:** Reduce the class of interpretations

**Herbrand's Thm:** Contradictions have **finite**, **ground** proofs

**Unification:** Automatically find the right instantiations

Finally, combine unification with **resolution**



## Skolemization, or Getting Rid of $\exists$

Start with a formula in NNF, with quantifiers nested like this:

$$\forall x_1 (\dots \forall x_2 (\dots \forall x_k (\dots \exists y A \dots) \dots) \dots)$$

Choose a fresh  $k$ -place function symbol, say  $f$

Delete  $\exists y$  and replace  $y$  by  $f(x_1, x_2, \dots, x_k)$ . We get

$$\forall x_1 (\dots \forall x_2 (\dots \forall x_k (\dots A[f(x_1, x_2, \dots, x_k)/y] \dots) \dots) \dots)$$

Repeat until no  $\exists$  quantifiers remain



## Example of Conversion to Clauses

For proving  $\exists x [P(x) \rightarrow \forall y P(y)]$

$\neg [\exists x [P(x) \rightarrow \forall y P(y)]]$       negated goal

$\forall x [P(x) \wedge \exists y \neg P(y)]$       conversion to NNF

$\forall x [P(x) \wedge \neg P(f(x))]$       Skolem term  $f(x)$

$\{P(x)\} \quad \{\neg P(f(x))\}$       Final clauses



## Correctness of Skolemization

The formula  $\forall x \exists y A$  is consistent

$\iff$  it holds in some interpretation  $\mathcal{I} = (D, I)$

$\iff$  for all  $x \in D$  there is some  $y \in D$  such that  $A$  holds

$\iff$  some function  $\hat{f}$  in  $D \rightarrow D$  yields suitable values of  $y$

$\iff A[f(x)/y]$  holds in some  $\mathcal{I}'$  extending  $\mathcal{I}$  so that  $f$  denotes  $\hat{f}$

$\iff$  the formula  $\forall x A[f(x)/y]$  is consistent.



## The Herbrand Universe for a Set of Clauses $S$

$H_0 \stackrel{\text{def}}{=} \text{the set of constants in } S$  (must be non-empty)

$H_{i+1} \stackrel{\text{def}}{=} H_i \cup \{f(t_1, \dots, t_n) \mid t_1, \dots, t_n \in H_i$

and  $f$  is an  $n$ -place function symbol in  $S\}$

$H \stackrel{\text{def}}{=} \bigcup_{i \geq 0} H_i$       Herbrand Universe

$H_i$  contains just the terms with at most  $i$  nested function applications.

$H$  consists of the terms in  $S$  that contain no variables (ground terms).



## The Herbrand Semantics of Predicates

An Herbrand interpretation defines an  $n$ -place predicate  $P$  to denote a truth-valued function in  $H^n \rightarrow \{1, 0\}$ , making  $P(t_1, \dots, t_n)$  true ...

- if and only if the **formula**  $P(t_1, \dots, t_n)$  holds in our desired “real” interpretation  $\mathcal{I}$  of the clauses.
- Thus, an Herbrand interpretation can imitate **any** other interpretation.



## The Inspiration for Clause Methods

Herbrand's Theorem: *Let  $S$  be a set of clauses.*

*$S$  is unsatisfiable  $\iff$  there is a **finite** unsatisfiable set  $S'$  of **ground instances** of clauses of  $S$ .*

- **Finite**: we can compute it
- **Instance**: result of substituting for variables
- **Ground**: no variables remain—it's propositional!

**Example:**  $S$  could be  $\{P(x)\} \quad \{\neg P(f(y))\}$ ,  
and  $S'$  could be  $\{P(f(a))\} \quad \{\neg P(f(a))\}$ .



## Unification

Finding a **common instance** of two terms. Lots of applications:

- **Prolog** and other logic programming languages
- **Theorem proving**: resolution and other procedures
- Tools for reasoning with **equations** or satisfying **constraints**
- Polymorphic type-checking (**ML** and other functional languages)

It is an intuitive generalization of pattern-matching.



## Four Unification Examples

$f(x, b)$	$f(x, x)$	$f(x, x)$	$j(x, x, z)$
$f(a, y)$	$f(a, b)$	$f(y, g(y))$	$j(w, a, h(w))$
$f(a, b)$	None	None	$j(a, a, h(a))$
$[a/x, b/y]$	Fail	Fail	$[a/w, a/x, h(a)/z]$

The output is a **substitution**, mapping variables to terms.

Other occurrences of those variables also must be updated.

Unification yields a **most general** substitution (in a technical sense).



## Theorem-Proving Example 1

$$(\exists y \forall x R(x, y)) \rightarrow (\forall x \exists y R(x, y))$$

After negation, the clauses are  $\{R(x, a)\}$  and  $\{\neg R(b, y)\}$ .

The literals  $R(x, a)$  and  $R(b, y)$  have unifier  $[b/x, a/y]$ .

We have the contradiction  $R(b, a)$  and  $\neg R(b, a)$ .

The theorem is proved by contradiction!



## Theorem-Proving Example 2

$$(\forall x \exists y R(x, y)) \rightarrow (\exists y \forall x R(x, y))$$

After negation, the clauses are  $\{R(x, f(x))\}$  and  $\{\neg R(g(y), y)\}$ .

The literals  $R(x, f(x))$  and  $R(g(y), y)$  are not unifiable.

(They fail the **occurs check**.)

We can't get a contradiction. **Formula is not a theorem!**



## The Binary Resolution Rule

$$\frac{\{B, A_1, \dots, A_m\} \quad \{\neg D, C_1, \dots, C_n\}}{\{A_1, \dots, A_m, C_1, \dots, C_n\}\sigma} \quad \text{provided } B\sigma = D\sigma$$

( $\sigma$  is a **most general** unifier of  $B$  and  $D$ .)

First, **rename variables apart** in the clauses! For example, given

$$\{P(x)\} \quad \text{and} \quad \{\neg P(g(x))\},$$

we must rename  $x$  in one of the clauses. (Otherwise, unification fails.)



## The Factoring Rule

This inference collapses unifiable literals **in one clause**:

$$\frac{\{B_1, \dots, B_k, A_1, \dots, A_m\}}{\{B_1, A_1, \dots, A_m\}\sigma} \quad \text{provided } B_1 \sigma = \dots = B_k \sigma$$

**Example:** Prove  $\forall x \exists y \neg(P(y, x) \leftrightarrow \neg P(y, y))$

The clauses are  $\{\neg P(y, a), \neg P(y, y)\}$        $\{P(y, y), P(y, a)\}$

Factoring yields  $\{\neg P(a, a)\}$        $\{P(a, a)\}$

Resolution yields the empty clause!



## A Non-Trivial Proof

$$\exists x [P \rightarrow Q(x)] \wedge \exists x [Q(x) \rightarrow P] \rightarrow \exists x [P \leftrightarrow Q(x)]$$

Clauses are  $\{P, \neg Q(b)\}$   $\{P, Q(x)\}$   $\{\neg P, \neg Q(x)\}$   $\{\neg P, Q(a)\}$

Resolve  $\{P, \underline{\neg Q(b)}\}$  with  $\{P, \underline{Q(x)}\}$  getting  $\{P, P\}$

Factor  $\{P, P\}$  getting  $\{P\}$

Resolve  $\{\neg P, \underline{\neg Q(x)}\}$  with  $\{\neg P, \underline{Q(a)}\}$  getting  $\{\neg P, \neg P\}$

Factor  $\{\neg P, \neg P\}$  getting  $\{\neg P\}$

Resolve  $\{P\}$  with  $\{\neg P\}$  getting  $\square$



## What About Equality?

In theory, it's enough to add the **equality axioms**:

- The **reflexive**, **symmetric** and **transitive** laws.
- **Substitution** laws like  $\{x \neq y, f(x) = f(y)\}$  for each  $f$ .
- **Substitution** laws like  $\{x \neq y, \neg P(x), P(y)\}$  for each  $P$ .

In practice, we need something special: the **paramodulation rule**

$$\frac{\{B[t'], A_1, \dots, A_m\} \quad \{t = u, C_1, \dots, C_n\}}{\{B[u], A_1, \dots, A_m, C_1, \dots, C_n\}\sigma} \quad (\text{if } t\sigma = t'\sigma)$$



## Prolog Clauses

Prolog clauses have a restricted form, with **at most one** positive literal.

The **definite clauses** form the program. Procedure  $B$  with body “commands”  $A_1, \dots, A_m$  is

$$B \leftarrow A_1, \dots, A_m$$

The single **goal clause** is like the “execution stack”, with say  $m$  tasks left to be done.

$$\leftarrow A_1, \dots, A_m$$

## Prolog Execution

Linear resolution:

- Always resolve some program clause with the goal clause.
- The result becomes the new goal clause.

Try the program clauses in **left-to-right** order.

Solve the goal clause's literals in **left-to-right** order.

Use **depth-first search**. (Performs **backtracking**, using little space.)

Do unification without **occurs check**. (**Unsound**, but needed for speed)



## A (Pure) Prolog Program

```
parent(elizabeth,charles) .  
parent(elizabeth,andrew) .
```

```
parent(charles,william) .  
parent(charles,henry) .
```

```
parent(andrew,beatrice) .  
parent(andrew,eugenia) .
```

```
grand(X,Z) :- parent(X,Y) , parent(Y,Z) .  
cousin(X,Y) :- grand(Z,X) , grand(Z,Y) .
```



## Prolog Execution

```

                                :- cousin(X,Y) .
                                :- grand(Z1,X), grand(Z1,Y) .
        :- parent(Z1,Y2), parent(Y2,X), grand(Z1,Y) .
*           :- parent(charles,X), grand(elizabeth,Y) .
X=william   :- grand(elizabeth,Y) .
            :- parent(elizabeth,Y5), parent(Y5,Y) .
*
Y=beatrice  :- parent(andrew,Y) .
            :- □.

```

\* = backtracking choice point

16 solutions including `cousin(william,william)`  
and `cousin(william,henry)`



## Another FOL Proof Procedure: Model Elimination

A Prolog-like method to run on fast Prolog architectures.

**Contrapositives:** treat clause  $\{A_1, \dots, A_m\}$  like the  $m$  clauses

$$A_1 \leftarrow \neg A_2, \dots, \neg A_m$$

$$A_2 \leftarrow \neg A_3, \dots, \neg A_m, \neg A_1$$

$$\vdots$$

$$A_m \leftarrow \neg A_1, \dots, \neg A_{m-1}$$

**Extension** rule: when proving goal  $P$ , assume  $\neg P$ .



## A Survey of Automatic Theorem Provers

First-order Resolution: E, SPASS, Vampire, ...

Higher-Order Logic: TPS, LEO and LEO-II, Satallax

Model Elimination: Prolog Technology Theorem Prover, SETHEO  
(historical)

Parallel ME: PARTHENON, PARTHEO

Tableau (sequent) based: LeanTAP, 3TAP, ...



## Decision Problems

Any formally-stated question: is  $n$  prime or not? Is the string  $s$  accepted by a given context-free grammar?

Unfortunately, most decision problems for logic are difficult:

- **Propositional satisfiability** NP-complete.
- The **halting problem** is undecidable. Therefore there is no decision procedure to identify first-order theorems.
- The theory of **integer arithmetic** is undecidable (Gödel).



## Solvable Decision Problems

Propositional formulas are decidable: use the DPLL algorithm.

Linear arithmetic formulas are decidable:

- comparisons using  $+$  and  $-$  but  $\times$  only with constants, e.g.
- $2x < y \wedge y < x$  (satisfiable by  $y = -3, x = -2$ ) or  $2x < y \wedge y < x \wedge 3x > 2$  (unsatisfiable)
- the integer and real (or rational) cases require different algorithms

Polynomial arithmetic is decidable, and so is Euclidean geometry.



## Fourier-Motzkin Variable Elimination

Decides conjunctions of linear constraints over reals/rationals

$$\bigwedge_{i=1}^m \sum_{j=1}^n a_{ij} x_j \leq b_i$$

Eliminate variables one-by-one until one remains, or contradiction

Devised by Fourier (1826) — resembles Gaussian elimination

One of the first decision procedures to be implemented

Worst-case complexity:  $O(m^{2^n})$



## Basic Idea: Upper and Lower Bounds

To eliminate variable  $x_n$ , consider constraint  $i$ , for  $i = 1, \dots, m$ :

Define  $\beta_i = b_i - \sum_{j=1}^{n-1} a_{ij}x_j$ . Rewrite constraint  $i$ :

$$\text{If } a_{in} > 0 \text{ then } x_n \leq \frac{\beta_i}{a_{in}}$$

$$\text{if } a_{in} < 0 \text{ then } -x_n \leq -\frac{\beta_i}{a_{in}}$$

Adding two such constraints yields  $0 \leq \frac{\beta_i}{a_{in}} - \frac{\beta_{i'}}{a_{i'n}}$

Do this for **all combinations** with opposite signs

Then delete original constraints (except where  $a_{in} = 0$ )



## Fourier-Motzkin Elimination Example

initial problem	eliminate $x$	eliminate $z$	result
$x \leq y$	$z \leq 0$	$0 \leq -1$	UNSAT
$x \leq z$	$y + z \leq 0$	$y \leq -1$	
$-x + y + 2z \leq 0$			
$-z \leq -1$	$-z \leq -1$		



## Quantifier Elimination (QE)

Skolemization eliminates quantifiers but only preserves **consistency**.

QE transforms a formula to a quantifier-free but **equivalent** formula.

The idea of Fourier-Motzkin is that (e.g.)

$$\exists xy (2x < y \wedge y < x) \iff \exists x 2x < x \iff \mathbf{f}$$

In general, the quantifier-free formula is **enormous**.

- With no free variables, the end result must be **t** or **f**.
- But even then, the time complexity tends to be hyper-exponential!



## Other Decidable Theories

Linear **integer** arithmetic: use Omega test or Cooper's algorithm, but **any** decision algorithm has a worst-case runtime of at least  $2^{2^{cn}}$

QE for **real polynomial arithmetic**:

$$\exists x [ax^2 + bx + c = 0] \iff b^2 \geq 4ac \wedge (c = 0 \vee a \neq 0 \vee b^2 > 4ac)$$

There exist decision procedures for arrays, lists, bit vectors, ...

Sometimes, they can cooperate to decide **combinations of theories**.



## Problem: To Combine Theories with Boolean Logic

These procedures expect **existentially quantified conjunctions**.

Formulas must be converted to **disjunctive** normal form.

Universal quantifiers must be eliminated using  $\forall x A \simeq \neg(\exists x (\neg A))$ .

**Could there be a better way? Couldn't we somehow use DPLL?**



## Satisfiability Modulo Theories

**Idea:** use DPLL for logical reasoning, decision procedures for theories

Clauses can have literals like  $2x < y$ , which are used as **names**.

If DPLL finds a contradiction, then the clauses are unsatisfiable.

Asserted literals are checked by the decision procedure:

- **Unsatisfiable** conjunctions of literals are noted as new clauses.
- Case splitting is interleaved with decision procedure calls.



## SMT Example

$$\{c = 0, 2a < b\} \quad \{b < a\} \quad \{3a > 2, a < 0\} \quad \{c \neq 0, \neg(b < a)\}$$


---

$$\{c = 0, 2a < b\} \quad \{3a > 2, a < 0\} \quad \{c \neq 0\} \quad \text{unit } b < a$$

$$\{2a < b\} \quad \{3a > 2, a < 0\} \quad \text{unit } c \neq 0$$

$$\{3a > 2, a < 0\} \quad \text{unit } 2a < b$$

Now a case split returns a “model”:  $b < a, c \neq 0, 2a < b, 3a > 2$

But the dec. proc. finds these contradictory and returns a new clause:

$$\{\neg(b < a), \neg(2a < b), \neg(3a > 2)\}$$

Finally, we get a true model:  $b < a \wedge c \neq 0 \wedge 2a < b \wedge a < 0$

## SMT Solvers and Their Applications

Popular ones include Z3, Yices, CVC4, but there are many others.

Representative applications:

- Hardware and software verification
- Program analysis and symbolic software execution
- Planning and constraint solving
- Hybrid systems and control engineering



## BDDs: Binary Decision Diagrams

A **canonical form** for boolean expressions: decision trees with sharing.

- **ordered** propositional symbols (the **variables**)
- **sharing** of identical subtrees
- **hashing** and other optimisations

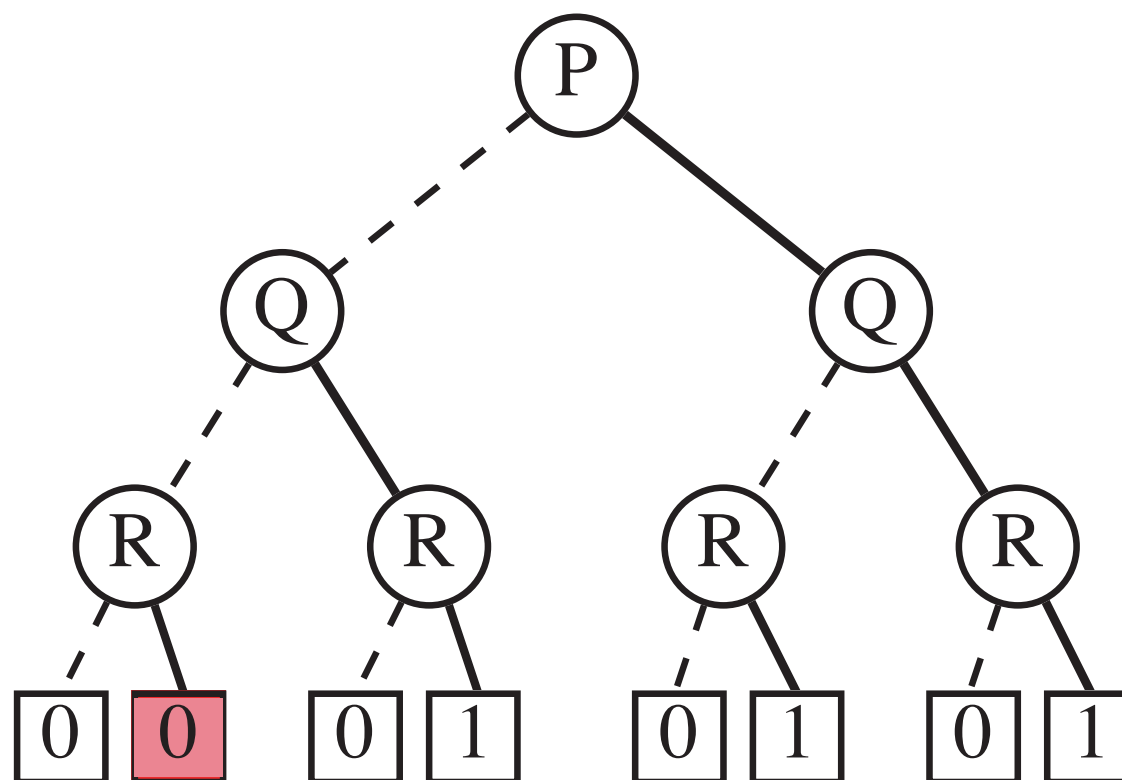
Detects if a formula is tautologous ( $=1$ ) or inconsistent ( $=0$ ).

Exhibits **models** (paths to 1) if the formula is satisfiable.

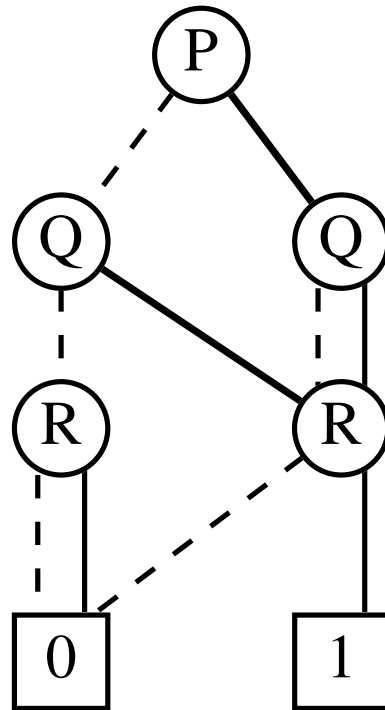
Excellent for verifying digital circuits, with many other applications.



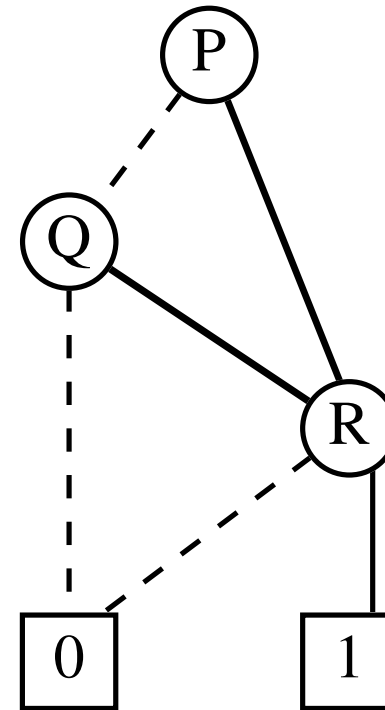
## Decision Diagram for $(P \vee Q) \wedge R$



## Converting a Decision Diagram to a BDD



No duplicates



No redundant tests



## Building BDDs Efficiently

Do not construct the full binary tree!

Do not expand  $\rightarrow$ ,  $\leftrightarrow$ ,  $\oplus$  (exclusive OR) to other connectives!!

- Recursively convert operands to BDDs.
- Combine operand BDDs, respecting the ordering and sharing.
- Delete redundant variable tests.



## Canonical Form Algorithm

To convert  $Z \wedge Z'$ , where  $Z$  and  $Z'$  are already BDDs:

*Trivial if either operand is 1 or 0.*

Let  $Z = \text{if}(P, X, Y)$  and  $Z' = \text{if}(P', X', Y')$

- If  $P = P'$  then recursively convert  $\text{if}(P, X \wedge X', Y \wedge Y')$ .
- If  $P < P'$  then recursively convert  $\text{if}(P, X \wedge Z', Y \wedge Z')$ .
- If  $P > P'$  then recursively convert  $\text{if}(P', Z \wedge X', Z \wedge Y')$ .



## Canonical Forms of Other Connectives

$Z \vee Z'$ ,  $Z \rightarrow Z'$  and  $Z \leftrightarrow Z'$  are converted to BDDs similarly.

Some cases, like  $Z \rightarrow 0$  and  $Z \leftrightarrow 0$ , reduce to negation.

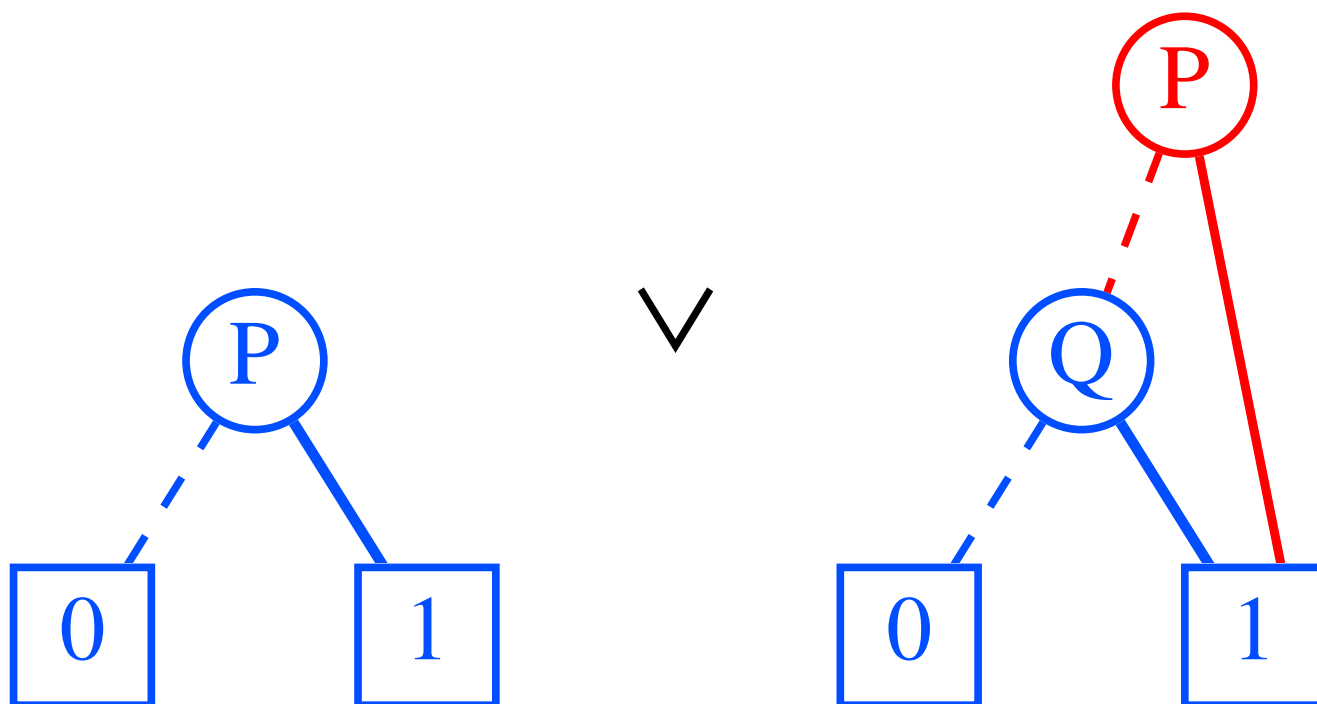
Here is how to convert  $\neg Z$ , where  $Z$  is a BDD:

- If  $Z = \text{if}(P, X, Y)$  then recursively convert  $\text{if}(P, \neg X, \neg Y)$ .
- if  $Z = 1$  then return 0, and if  $Z = 0$  then return 1.

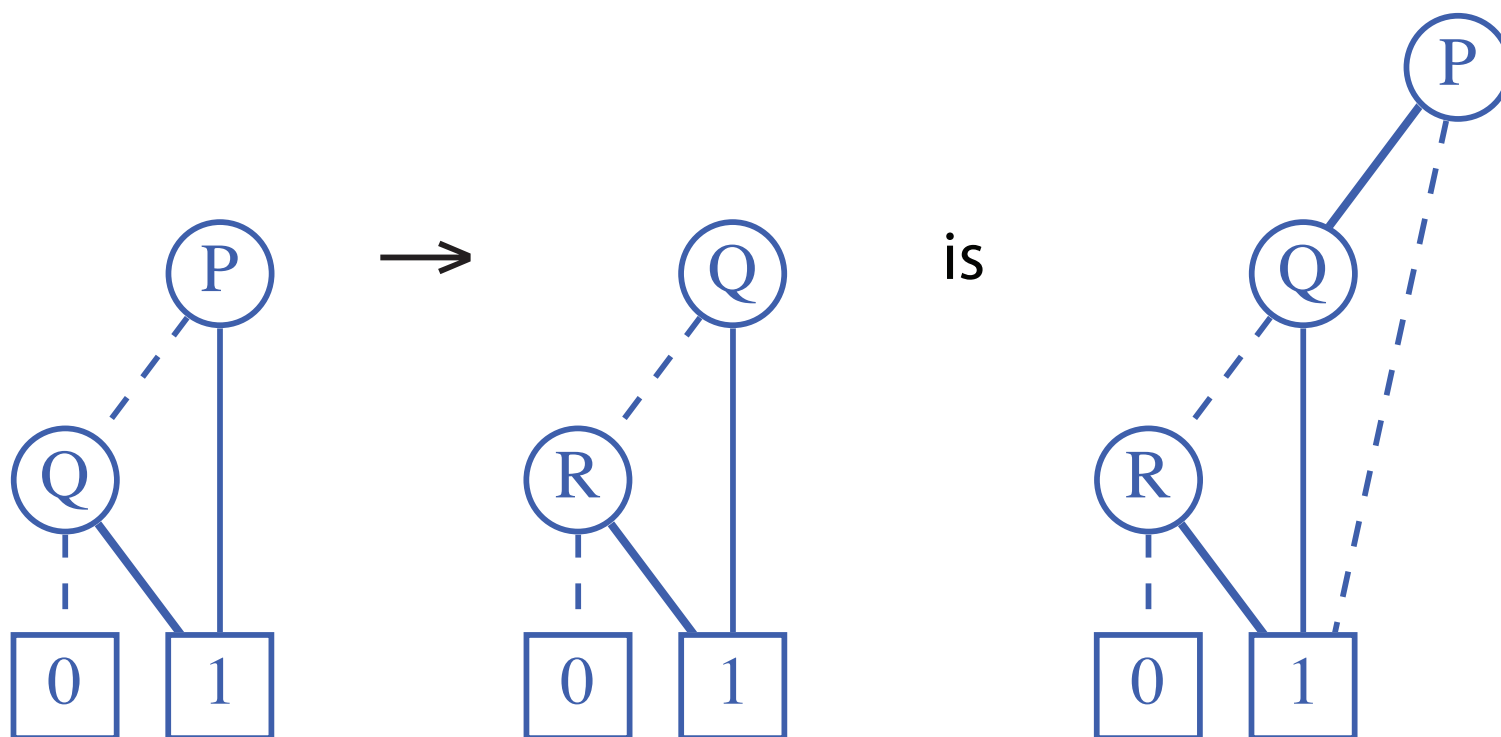
(In effect we copy the BDD but exchange the 1 and 0 at the bottom.)



## Canonical Form (that is, BDD) of $P \vee Q$



# Canonical Form of $P \vee Q \rightarrow Q \vee R$



## Optimisations

Never build the same BDD twice, but share pointers. Advantages:

- If  $X \simeq Y$ , then the addresses of  $X$  and  $Y$  are equal.
- Can see if  $\text{if}(P, X, Y)$  is redundant by checking if  $X = Y$ .
- Can quickly simplify special cases like  $X \wedge X$ .

Never convert  $X \wedge Y$  twice, but keep a hash table of known canonical forms. This prevents redundant computations.



## Final Observations

The variable ordering is crucial. Consider this formula:

$$(P_1 \wedge Q_1) \vee \cdots \vee (P_n \wedge Q_n)$$

A **good ordering** is  $P_1 < Q_1 < \cdots < P_n < Q_n$ : the BDD is linear.

With  $P_1 < \cdots < P_n < Q_1 < \cdots < Q_n$ , the BDD is **exponential**.

Many digital circuits have small BDDs: adders, but not multipliers.

BDDs can solve problems in hundreds of variables.

The general case remains hard (it is NP-complete).



## Modal Operators

$W$ : set of **possible worlds** (machine states, future times, . . .)

$R$ : **accessibility relation** between worlds

$(W, R)$  is called a **modal frame**

$\Box A$  means  $A$  is **necessarily true**  
 $\Diamond A$  means  $A$  is **possibly true**

} in all worlds **accessible** from **here**

$$\neg \Diamond A \simeq \Box \neg A$$

$A$  cannot be true  $\iff A$  must be false



## Semantics of Propositional Modal Logic

For a particular frame  $(W, R)$

An **interpretation**  $I$  maps the propositional letters to **subsets** of  $W$

$w \Vdash A$  means  **$A$  is true in world  $w$**

$$w \Vdash P \iff w \in I(P)$$

$$w \Vdash A \wedge B \iff w \Vdash A \text{ and } w \Vdash B$$

$$w \Vdash \Box A \iff v \Vdash A \text{ for all } v \text{ such that } R(w, v)$$

$$w \Vdash \Diamond A \iff v \Vdash A \text{ for some } v \text{ such that } R(w, v)$$



## Truth and Validity in Modal Logic

For a particular frame  $(W, R)$ , and interpretation  $I$

$w \Vdash A$  means  $A$  is true in world  $w$

$\models_{W,R,I} A$  means  $w \Vdash A$  for all  $w$  in  $W$

$\models_{W,R} A$  means  $w \Vdash A$  for all  $w$  and all  $I$

$\models A$  means  $\models_{W,R} A$  for all frames;  $A$  is **universally valid**

... but typically we constrain  $R$  to be, say, **transitive**.

**All propositional tautologies are universally valid!**



## A Hilbert-Style Proof System for K

Extend your favourite propositional proof system with

$$\text{Dist} \quad \Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$$

Inference Rule: **Necessitation**

$$\frac{A}{\Box A}$$

Treat  $\Diamond$  as a **definition**

$$\Diamond A \stackrel{\text{def}}{=} \neg \Box \neg A$$



## Variant Modal Logics

Start with pure modal logic, which is called K

Add **axioms** to constrain the accessibility relation:

T	$\Box A \rightarrow A$	(reflexive)	logic T
4	$\Box A \rightarrow \Box \Box A$	(transitive)	logic S4
B	$A \rightarrow \Box \Diamond A$	(symmetric)	logic S5

And countless others!

We mainly look at S4, which resembles a logic of time.



## Extra Sequent Calculus Rules for S4

$$\frac{A, \Gamma \Rightarrow \Delta}{\Box A, \Gamma \Rightarrow \Delta} \quad (\Box l)$$

$$\frac{\Gamma^* \Rightarrow \Delta^*, A}{\Gamma \Rightarrow \Delta, \Box A} \quad (\Box r)$$

$$\frac{A, \Gamma^* \Rightarrow \Delta^*}{\Diamond A, \Gamma \Rightarrow \Delta} \quad (\Diamond l)$$

$$\frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \Diamond A} \quad (\Diamond r)$$

$$\Gamma^* \stackrel{\text{def}}{=} \{\Box B \mid \Box B \in \Gamma\}$$

Erase **non- $\Box$**  assumptions.

$$\Delta^* \stackrel{\text{def}}{=} \{\Diamond B \mid \Diamond B \in \Delta\}$$

Erase **non- $\Diamond$**  goals!

## A Proof of the Distribution Axiom

$$\begin{array}{c}
 \frac{\overline{A \Rightarrow B, A} \quad \overline{B, A \Rightarrow B}}{A \rightarrow B, A \Rightarrow B} \quad (\rightarrow\text{l}) \\
 \hline
 \frac{A \rightarrow B, A \Rightarrow B}{A \rightarrow B, \Box A \Rightarrow B} \quad (\Box\text{l}) \\
 \hline
 \frac{A \rightarrow B, \Box A \Rightarrow B}{\Box(A \rightarrow B), \Box A \Rightarrow B} \quad (\Box\text{l}) \\
 \hline
 \frac{\Box(A \rightarrow B), \Box A \Rightarrow B}{\Box(A \rightarrow B), \Box A \Rightarrow \Box B} \quad (\Box\text{r})
 \end{array}$$

And thus  $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

Must apply  $(\Box\text{r})$  first!



## Part of an “Operator String Equivalence”

$$\begin{array}{c}
 \overline{\Diamond A \Rightarrow \Diamond A} \\
 \hline
 \Box \Diamond A \Rightarrow \Diamond A \quad (\Box l) \\
 \hline
 \Diamond \Box \Diamond A \Rightarrow \Diamond A \quad (\Diamond l) \\
 \hline
 \Box \Diamond \Box \Diamond A \Rightarrow \Diamond A \quad (\Box l) \\
 \hline
 \Box \Diamond \Box \Diamond A \Rightarrow \Box \Diamond A \quad (\Box r)
 \end{array}$$

In fact,  $\Box \Diamond \Box \Diamond A \simeq \Box \Diamond A$       also  $\Box \Box A \simeq \Box A$

The S4 operator strings are  $\Box$     $\Diamond$     $\Box \Diamond$     $\Diamond \Box$     $\Box \Diamond \Box$     $\Diamond \Box \Diamond$



## Two Failed Proofs

$$\frac{\Rightarrow A}{\Rightarrow \Diamond A} (\Diamond r)$$

$$\frac{\Rightarrow \Diamond A}{A \Rightarrow \Box \Diamond A} (\Box r)$$

$$\frac{B \Rightarrow A \wedge B}{B \Rightarrow \Diamond(A \wedge B)} (\Diamond r)$$

$$\frac{B \Rightarrow \Diamond(A \wedge B)}{\Diamond A, \Diamond B \Rightarrow \Diamond(A \wedge B)} (\Diamond l)$$

Can extract a countermodel from the proof attempt



## Simplifying the Sequent Calculus

7 connectives (or 9 for modal logic):

$\neg \quad \wedge \quad \vee \quad \rightarrow \quad \leftrightarrow \quad \forall \quad \exists \quad (\Box \quad \Diamond)$

Left and right: so 14 rules (or 18) plus basic sequent, cut

Idea! Work in **Negation Normal Form**

Fewer connectives:  $\wedge \quad \vee \quad \forall \quad \exists \quad (\Box \quad \Diamond)$

Sequents need **one side only!**



## Tableau Calculus: Left-Only

$$\begin{array}{c}
 \frac{}{\neg A, A, \Gamma \Rightarrow} \text{ (basic)} \qquad \frac{\neg A, \Gamma \Rightarrow \quad A, \Gamma \Rightarrow}{\Gamma \Rightarrow} \text{ (cut)} \\
 \\
 \frac{A, B, \Gamma \Rightarrow}{A \wedge B, \Gamma \Rightarrow} (\wedge l) \qquad \frac{A, \Gamma \Rightarrow \quad B, \Gamma \Rightarrow}{A \vee B, \Gamma \Rightarrow} (\vee l) \\
 \\
 \frac{A[t/x], \Gamma \Rightarrow}{\forall x A, \Gamma \Rightarrow} (\forall l) \qquad \frac{A, \Gamma \Rightarrow}{\exists x A, \Gamma \Rightarrow} (\exists l)
 \end{array}$$

Rule  $(\exists l)$  holds **provided**  $x$  is not free in the conclusion!



## Tableau Rules for S4

$$\frac{A, \Gamma \Rightarrow}{\Box A, \Gamma \Rightarrow} (\Box I) \qquad \frac{A, \Gamma^* \Rightarrow}{\Diamond A, \Gamma \Rightarrow} (\Diamond I)$$

$$\Gamma^* \stackrel{\text{def}}{=} \{\Box B \mid \Box B \in \Gamma\} \quad \text{Erase non-}\Box \text{ assumptions}$$

From 14 (or 18) rules to 4 (or 6)

Left-side only system uses **proof by contradiction**

Right-side only system is an exact **dual**



## Tableau Proof of $\forall x (P \rightarrow Q(x)) \rightarrow [P \rightarrow \forall y Q(y)]$

Negate and convert to NNF:

$$P, \exists y \neg Q(y), \forall x (\neg P \vee Q(x)) \Rightarrow$$

$$\frac{\frac{\frac{P, \neg Q(y), \neg P \Rightarrow \quad P, \neg Q(y), Q(y) \Rightarrow}{P, \neg Q(y), \neg P \vee Q(y) \Rightarrow} (\vee I)}{P, \neg Q(y), \forall x (\neg P \vee Q(x)) \Rightarrow} (\forall I)}{P, \exists y \neg Q(y), \forall x (\neg P \vee Q(x)) \Rightarrow} (\exists I)$$



## The Free-Variable Tableau Calculus

Rule  $(\forall I)$  now inserts a **new** free variable:

$$\frac{A[z/x], \Gamma \Rightarrow}{\forall x A, \Gamma \Rightarrow} (\forall I)$$

Let unification instantiate **any free variable**

In  $\neg A, B, \Gamma \Rightarrow$  try unifying  $A$  with  $B$  to make a basic sequent

**Updating a variable affects entire proof tree**

What about rule  $(\exists I)$ ? **Do not use it!** Instead, **Skolemize!**



## Skolemization from NNF

Recall e.g. that we Skolemize

$$[\forall y \exists z Q(y, z)] \wedge \exists x P(x) \quad \text{to} \quad [\forall y Q(y, f(y))] \wedge P(a)$$

**Remark:** pushing quantifiers in (**miniscoping**) gives better results.

**Example:** proving  $\exists x \forall y [P(x) \rightarrow P(y)]$ :

**Negate; convert to NNF:**  $\forall x \exists y [P(x) \wedge \neg P(y)]$

**Push in the  $\exists y$ :**  $\forall x [P(x) \wedge \exists y \neg P(y)]$

**Push in the  $\forall x$ :**  $(\forall x P(x)) \wedge (\exists y \neg P(y))$

**Skolemize:**  $\forall x P(x) \wedge \neg P(a)$



## Free-Variable Tableau Proof of $\exists x \forall y [P(x) \rightarrow P(y)]$

$y \mapsto f(z)$	
$P(y), \neg P(f(y)), P(z), \neg P(f(z)) \Rightarrow$	(basic)
$P(y), \neg P(f(y)), P(z) \wedge \neg P(f(z)) \Rightarrow$	( $\wedge I$ )
$P(y), \neg P(f(y)), \forall x [P(x) \wedge \neg P(f(x))] \Rightarrow$	( $\forall I$ )
$P(y) \wedge \neg P(f(y)), \forall x [P(x) \wedge \neg P(f(x))] \Rightarrow$	( $\wedge I$ )
$\forall x [P(x) \wedge \neg P(f(x))] \Rightarrow$	( $\forall I$ )

Unification chooses the term for ( $\forall I$ )



## A Failed Proof

Try to prove  $\forall x [P(x) \vee Q(x)] \rightarrow [\forall x P(x) \vee \forall x Q(x)]$

**NNF:**  $\exists x \neg P(x) \wedge \exists x \neg Q(x) \wedge \forall x [P(x) \vee Q(x)] \Rightarrow$

**Skolemize:**  $\neg P(a), \neg Q(b), \forall x [P(x) \vee Q(x)] \Rightarrow$

$$\begin{array}{c}
 \frac{y \mapsto a}{\neg P(a), \neg Q(b), P(y) \Rightarrow} \quad \frac{y \mapsto b???}{\neg P(a), \neg Q(b), Q(y) \Rightarrow} \\
 \hline
 \frac{\neg P(a), \neg Q(b), P(y) \vee Q(y) \Rightarrow}{\neg P(a), \neg Q(b), \forall x [P(x) \vee Q(x)] \Rightarrow} \quad (\forall I)
 \end{array}$$



## The World's Smallest Theorem Prover?

```

prove ( (A,B) , UnExp, Lits, FreeV, VarLim) :- !,                                and
    prove (A, [B|UnExp], Lits, FreeV, VarLim) .
prove ( (A;B) , UnExp, Lits, FreeV, VarLim) :- !,                                or
    prove (A, UnExp, Lits, FreeV, VarLim) ,
    prove (B, UnExp, Lits, FreeV, VarLim) .
prove (all (X,Fml) , UnExp, Lits, FreeV, VarLim) :- !,                            forall
    \+ length (FreeV, VarLim) ,
    copy_term ( (X,Fml, FreeV) , (X1,Fml1, FreeV) ) ,
    append (UnExp, [all (X,Fml) ] , UnExp1) ,
    prove (Fml1, UnExp1, Lits, [X1|FreeV], VarLim) .
prove (Lit, _, [L|Lits], _, _) :-                                              literals; negation
    (Lit = -Neg; -Lit = Neg) ->
    (unify (Neg, L); prove (Lit, [], Lits, _, _)) .
prove (Lit, [Next|UnExp], Lits, FreeV, VarLim) :-                               next formula
    prove (Next, UnExp, [Lit|Lits], FreeV, VarLim) .

```