# Logic and Proof

Computer Science Tripos Part IB
Lent Term

Lawrence C Paulson

Computer Laboratory

University of Cambridge

lp15@cam.ac.uk

# 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: satisfiability, 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 $X$, $Y$, $Z$, ... 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 $Y \mapsto$ coal satisfies the statement

> Black is the colour of $Y$.

but the interpretation $Y \mapsto$ strawberries does not!

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

## Satisfiability

A set $S$ of statements is satisfiable if some interpretation satisfies all elements of $S$ at the same time. Otherwise $S$ is unsatisfiable.

Examples of unsatisfiable sets:

$$\{X \subseteq Y,\ Y \subseteq Z,\ \neg(X \subseteq Z)\}$$

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

## 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 $S \models A$.

$$\{X \subseteq Y,\ Y \subseteq Z\} \models X \subseteq Z$$

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

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

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

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

## Formal Proof

How can we prove that $A$ is valid? We can't test infinitely many cases.

A formal system is a model of mathematical reasoning

- theorems are inferred from axioms using inference rules.
- formal systems are themselves mathematical objects, hence we have meta-mathematics

# Inference Rules

An inference rule yields a conclusion from one or more premises.

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

This entailment suggests the inference rule

$$\frac{A_1 \qquad \ldots \qquad A_n}{B}$$

A system's axioms and inference rules must be selected carefully.

Theorems are constructed inductively from the axioms using rules.

## Schematic Inference Rules

$$\frac{X \subseteq Y \qquad Y \subseteq Z}{X \subseteq Z}$$

- 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

# Consistency vs Satisfiability

A formal system defines a set of theorems.

If every statement is a theorem, then the system is inconsistent.

An unsatisfiable set of axioms leads to an inconsistent formal system (in normal circumstances).

Satisfiability is the semantic counterpart of consistency.

## Richard's Paradox

Consider the list of all English phrases that define real numbers, e.g. "the base of the natural logarithm" or "the positive solution to $x^2 = 2$."

- Sort this list alphabetically, yielding a series $\{r_n\}$ of real numbers.
- Now define a new real number such that its $n$th decimal place is 1 if the $n$th decimal place of $r_n$ is not 1; otherwise 2.
- This is a real number not in our list of all definable real numbers.

## Why Should we use a Formal Language?

And again: 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

| | |
|---|---|
| $P, Q, R, \ldots$ | propositional letter |
| **t** | true |
| **f** | false |
| $\neg A$ | not $A$ |
| $A \land B$ | $A$ and $B$ |
| $A \lor B$ | $A$ or $B$ |
| $A \to B$ | 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 |

Later we shall see things like $\Box A$ that are not.

# 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).

$$\text{Write } \models_I A$$

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

$$\text{Write } \models A$$

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

## Implication, Entailment, Equivalence

$A \to B$ means simply $\neg A \lor B$.

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

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

### Equivalence

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

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

## An Issue: $A \to B$ Versus $\neg A \lor B$

It's called material implication, and it admits "paradoxes"* such as

$$P \to (Q \to P) \quad \text{and} \quad (P \to Q) \lor (Q \to R)$$

Some say that if $A \to B$ is true then $A$ should somehow cause $B$

Some "solutions":

- Relevance logic: still investigated by philosophers
- An interpretation in modal logic: see lecture 11

*these are not paradoxes

# Aside: Propositions as Types

Idea: instead of "$A$ is true", say "$a$ is evidence for $A$", written $a : A$

- If $a : A$ and $b : B$ then $(a, b) : A \times B$ Looks like conjunction!
- If $a : A$ then $\mathsf{Inl}(a) : A + B$
  If $b : B$ then $\mathsf{Inr}(b) : A + B$ Looks like disjunction!
- if $f(x) : B$ for all $x : A$
  then $\lambda x : A.\, b(x) : A \to B$ Looks like implication!

Also works for quantifiers, etc.: the basis of constructive type theory

# Constructive Logic is Weird

If $A \vee B$ then we know which one of $A$, $B$ is true

$A \vee \neg A$ is not a tautology

If $\exists x\, A$ then we know what $x$ is

$\exists$, $\forall$ are not duals

$A \to B$ isn't the same as $\neg A \vee B$

no material implication

$(P \to Q) \vee (Q \to R)$ is not a tautology, but $P \to (Q \to P)$ still is

Constructive (aka intuitionistic) logic is popular in theoretical CS

this material on constructive logic is NOT examinable

# Equivalences

$$A \land A \simeq A$$

$$A \land B \simeq B \land A$$

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

$$A \lor (B \land C) \simeq (A \lor B) \land (A \lor C)$$

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

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

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

Dual versions: exchange $\land$ with $\lor$ and **t** with **f** in any equivalence

## Equivalences Linking $\wedge$, $\vee$ and $\rightarrow$

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

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

The same ideas will be realised later in the sequent calculus

# Normal Forms in Computational Logic

Formal logics aim for readability, hence have a lot of redundancy

The connective NAND expresses all propositional formulas!

Negation normal form (NNF)

Conjunctive normal form (CNF)

Clause form and Prolog

Normal forms make proof procedures more efficient.

# Negation Normal Form

1. Get rid of $\leftrightarrow$ and $\rightarrow$, leaving just $\land$, $\lor$, $\neg$:

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

$$A \rightarrow B \simeq \neg A \lor B$$

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

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

$$\neg(A \land B) \simeq \neg A \lor \neg B$$

$$\neg(A \lor B) \simeq \neg A \land \neg B$$

## 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

$$P \lor Q \to Q \lor R$$

1. Elim $\to$: $\neg(P \lor Q) \lor (Q \lor R)$
2. Push $\neg$ in: $(\neg P \land \neg Q) \lor (Q \lor R)$
3. Push $\lor$ in: $(\neg P \lor Q \lor R) \land (\neg Q \lor Q \lor R)$
4. Simplify: $\neg P \lor Q \lor R$

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

## Tautology checking using CNF

$$((P \to Q) \to P) \to P$$

1. Elim $\to$: $\neg[\neg(\neg P \lor Q) \lor P] \lor P$

2. Push $\neg$ in: $[\neg\neg(\neg P \lor Q) \land \neg P] \lor P$

   $[(\neg P \lor Q) \land \neg P] \lor P$

3. Push $\lor$ in: $(\neg P \lor Q \lor P) \land (\neg P \lor P)$

4. Simplify: $\mathbf{t} \land \mathbf{t}$

   $\mathbf{t}$ *It's a tautology!*

In $A_1 \wedge \ldots \wedge A_n$ each $A_i$ can falsify the conjunction, if $n > 0$

Dually, DNF can detect unsatisfiability.

DNF was investigated in the 1960s for theorem proving by contradiction. We shall look at superior alternatives:

- Davis-Putnam methods, aka SAT solving
- binary decision diagrams (BDDs)

All can take exponential time—propositional satisfiability is NP-complete—but can solve big problems

# A Simple Proof System

*Axiom Schemes*

K $\quad A \rightarrow (B \rightarrow A)$

S $\quad (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

DN $\quad \neg\neg A \rightarrow A$

*Inference Rule: Modus Ponens*

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

This system regards $\neg$, $\vee$, $\wedge$ as abbreviations

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

$$(A \to ((D \to A) \to A)) \to \qquad (1)$$
$$((A \to (D \to A)) \to (A \to A)) \quad \text{by S}$$

$$A \to ((D \to A) \to A) \quad \text{by K} \qquad (2)$$

$$(A \to (D \to A)) \to (A \to A) \quad \text{by MP, (1), (2)} \qquad (3)$$

$$A \to (D \to A) \quad \text{by K} \qquad (4)$$

$$A \to A \quad \text{by MP, (3), (4)} \qquad (5)$$

Lengths of proofs here grow exponentially

# Aside: Propositions as Types Again*

Those axioms are not arbitrary (though many other such systems are)

Ever see a type-checking rule for function application?

$$\frac{f : A \to B \qquad a : A}{f(a) : B}$$

looks like Modus Ponens!

Axioms S and K give the types of combinators for expressing functions

A correspondence between terms and proofs, with links to $\lambda$-calculus

*not examinable

## 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 $S \vdash A$. We have some fundamental resuilts:

**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 $S \vdash A \to B$.

But meta-theory does not help us **use** the proof system.

# Gentzen's Natural Deduction Systems

The context of assumptions may vary.

To deduce $A \to B$, we get to assume $A$ temporarily:

$$\frac{\begin{matrix} A \\ \vdots \\ B \end{matrix}}{A \to B}$$

Each logical connective is defined independently.

Introduction and elimination rules: how to deduce and use $A \land B$:

$$\frac{A \qquad B}{A \land B} \qquad\qquad \frac{A \land B}{A} \qquad \frac{A \land B}{B}$$

# A Typical Natural Deduction Proof

$$\dfrac{\dfrac{\xcancel{A \lor B} \qquad \dfrac{\xcancel{A}}{B \lor A} \qquad \dfrac{\xcancel{B}}{B \lor A}}{B \lor A}}{A \lor B \to B \lor A}$$

Nice simple rules like

$$\frac{A}{A \lor B} \qquad \frac{B}{A \lor B} \qquad \frac{A \to B \qquad A}{B}$$

But the “crossing-out” process is confusing, and Natural Deduction works better for constructive logic

## The Sequent Calculus

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

$$\text{if } A_1 \land \ldots \land A_m \text{ then } B_1 \lor \ldots \lor B_n$$

$A_1$, ..., $A_m$ are assumptions; $B_1$, ..., $B_n$ 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

$$\frac{\Gamma \Rightarrow \Delta, A \qquad A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}\ (\mathit{cut})$$

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

$$\frac{A, B, \Gamma \Rightarrow \Delta}{A \land B, \Gamma \Rightarrow \Delta}\ (\land l) \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \land B}\ (\land r)$$

# More Sequent Calculus Rules

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

$$\frac{\Gamma \Rightarrow \Delta, A \qquad B, \Gamma \Rightarrow \Delta}{A \to B, \Gamma \Rightarrow \Delta} \ (\to l) \qquad \frac{A, \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B} \ (\to r)$$

## Proving the Formula $A \land B \to A$

$$
\dfrac{\dfrac{\overline{A, B \Rightarrow A}}{A \land B \Rightarrow A}\;(\land l)}{\Rightarrow (A \land B) \to A}\;(\to r)
$$

- Begin by writing down the sequent to be proved
- Be careful about skipping or combining steps
- You can't mix-and-match proof calculi. Just use sequent rules.

# Another Easy Sequent Calculus Proof

$$
\cfrac{\cfrac{\cfrac{\cfrac{}{A, B \Rightarrow B, C}}{A \Rightarrow B, B \to C}\;(\to r)}{\Rightarrow A \to B,\; B \to C}\;(\to r)}{\Rightarrow (A \to B) \lor (B \to C)}\;(\lor r)
$$

this was a "paradox of material implication"

## Part of a Distributive Law

$$
\dfrac{
  \dfrac{
    \dfrac{
      \overline{A \Rightarrow A, B}
      \qquad
      \dfrac{\overline{B, C \Rightarrow A, B}}{B \land C \Rightarrow A, B}\,(\land l)
    }{A \lor (B \land C) \Rightarrow A, B}\,(\lor l)
  }{A \lor (B \land C) \Rightarrow A \lor B}\,(\lor r)
  \qquad
  \text{similar}
}{A \lor (B \land C) \Rightarrow (A \lor B) \land (A \lor C)}\,(\land r)
$$

Second subtree proves $A \lor (B \land C) \Rightarrow A \lor C$ similarly

## A Failed Proof

$$\dfrac{\dfrac{A \Rightarrow B, C \qquad \overline{B \Rightarrow B, C}}{A \vee B \Rightarrow B, C}\,(\vee l)}{\dfrac{A \vee B \Rightarrow B \vee C}{\Rightarrow (A \vee B) \to (B \vee C)}\,(\to r)}\,(\vee r)$$

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

## Relevance to Automatic Theorem Proving

- Hao Wang's "Toward mechanical mathematics" (1960): spectacular results for both propositional and first-order logic
- Based on backward proof using the sequent calculus rules
- Modern tableaux calculi generalise these ideas

The sequent calculus is not practical for proving theorems on paper, as you will soon discover!

# The Tradeoffs in Formal Logic

We start with propositional logic

We enrich the language to first-order logic

We can enrich the language further with types, etc.

The price of expressiveness is difficulty of automation

Automation sometimes involves reversing the process of enrichment

this is basically the course plan

# Outline of First-Order Logic

Reasons about functions and relations over a set of individuals:

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

Reasons about all and some individuals:

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

Cannot reason about all functions or all relations, etc.

## Aside: Example of Syllogisms by Lewis Carroll

“All soldiers are strong; All soldiers are brave.

∴ Some strong men are brave.”

“None but the brave deserve the fair; Some braggarts are cowards.

∴ Some braggarts do not deserve the fair.”

“All soldiers can march; Some babies are not soldiers.

∴ Some babies cannot march”.*

*not valid

# 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

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

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

We choose the language, adopting any desired function symbols.

## Relation Symbols; Formulae

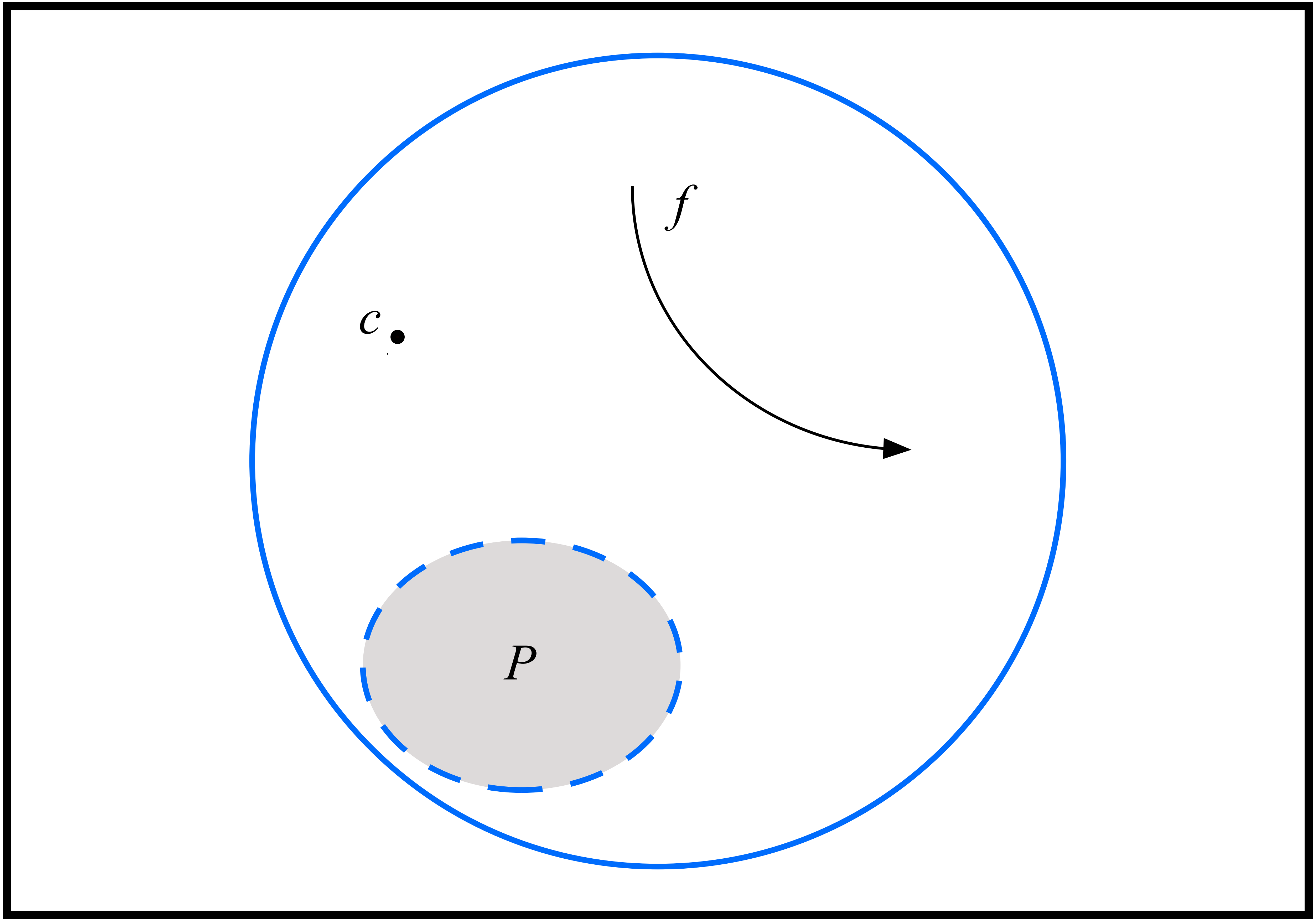
Each relation symbol stands for an $n$-place relation.

Equality is the 2-place relation symbol $=$

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

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

Later, we can add quantifiers.



$f$
$c$
$P$

# Aside: 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:

$$p(z,0) = 1 \qquad\qquad q(z,1) = z$$

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

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

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

based on early work by Robert Boyer and J Moore

# 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:

| | |
|---|---|
| $\forall xyz\, A$ | abbreviates $\forall x\, \forall y\, \forall z\, A$ |
| $\forall z\, .\, A \land B$ | is an alternative to $\forall z\, (A \land B)$ |

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

# The Expressiveness of Quantifiers

All men are mortal:

$$\forall x\,(\mathrm{man}(x) \to \mathrm{mortal}(x))$$

All mothers are female:

$$\forall x\,\mathrm{female}(\mathrm{mother}(x))$$

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

$$\exists x\,[A(x) \land \forall y\,(A(y) \to y = x)]$$

## 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 $\mathrm{mortal}(\mathrm{Socrates})$

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

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

A valuation $V : \mathsf{Var} \to D$ 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:

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

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

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

# 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 $V$.

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

automated theorem provers need to be based on rigorous theory

# 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 $\mathcal{I}_V[u]$ |
| $\models_{\mathcal{I},V} A \land B$ | 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 $\mathcal{I}$.

# A Final Remark On Syllogisms

Started with Aristotle and continued into the 19th Century

A highly technical subject with four “categorical sentences”:

**Type A** Every B is A

**Type I** Some B is A

**Type E** No B is A

**Type O** Some B is not A

And their 24 valid combinations, etc., etc. Be grateful for quantifiers!

## Reminder: 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 $A$:

$$
\begin{aligned}
(B \land C)[t/x] \quad &\text{is} \quad B[t/x] \land C[t/x] \\
(\forall x\, B)[t/x] \quad &\text{is} \quad \forall x\, B \\
(\forall y\, B)[t/x] \quad &\text{is} \quad \forall y\, B[t/x] \qquad (x \neq y) \\
(P(u))[t/x] \quad &\text{is} \quad P(u[t/x])
\end{aligned}
$$

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

As with propositional logic, we shall need normal forms!

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

$$\forall x\, A \simeq \forall x\, A \land A[t/x]$$

$$(\forall x\, A) \land (\forall x\, B) \simeq \forall x\, (A \land B)$$

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

Dual versions: exchange $\forall$ with $\exists$ and $\land$ with $\lor$

## Further Quantifier Equivalences

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

$$(\forall x\, A) \land B \simeq \forall x\, (A \land B)$$

$$(\forall x\, A) \lor B \simeq \forall x\, (A \lor B)$$

$$(\forall x\, A) \to B \simeq \exists x\, (A \to B)$$

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

## Aside: Reasoning by Equivalences

We saw an example of theorem proving by transformations in Lecture 2

[More sophisticated transformational approaches exist than CNF!]

Some say these are better than Gentzen methods (for hand proofs)

Trivial example: In $P \lor Q$ can simplify $Q$ assuming $P = \mathbf{f}$

In $P \land Q$ and $P \to Q$ can simplify $Q$ assuming $P = \mathbf{t}$

For both of those, simply by case analysis on $P$

## Reasoning by Equivalences with Quantifiers

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

$$
\begin{aligned}
&\exists z\,(P(z) \to P(a) \land P(b)) \\
&\qquad \simeq \forall z\,P(z) \to P(a) \land P(b) \\
&\qquad \simeq \forall z\,P(z) \land P(a) \land P(b) \to P(a) \land P(b) \\
&\qquad \simeq \mathbf{t}
\end{aligned}
$$

## Whitehead and Russell's *Principia Mathematica*

Includes a proof system for a sort of higher-order logic

Includes a valuable discussion of logical paradoxes

Attempts to show that maths can be reduced to logic

It's still in print including an abridged paperback edition.

CUP predicted that it would lose £600 — requested that the authors cover £100 of this. The Royal Society covered a further £200.

That was in 1910. For today's money, multiply by 100!

362 PROLEGOMENA TO CARDINAL ARITHMETIC [PART II

***54·42.** $\vdash :: \alpha \in 2 . \supset :. \beta \subset \alpha . \exists ! \beta . \beta \neq \alpha . \equiv . \beta \in \iota '' \alpha$

*Dem.*

$\vdash . *54\cdot4 . \quad \supset \vdash :: \alpha = \iota ' x \cup \iota ' y . \supset :.$

$\beta \subset \alpha . \exists ! \beta . \equiv : \beta = \Lambda . \vee . \beta = \iota ' x . \vee . \beta = \iota ' y . \vee . \beta = \alpha : \exists ! \beta :$

[*24·53·56.*51·161] $\equiv : \beta = \iota ' x . \vee . \beta = \iota ' y . \vee . \beta = \alpha$ (1)

$\vdash . *54\cdot25 .$ Transp $. *52\cdot22 . \supset \vdash : x \neq y . \supset . \iota ' x \cup \iota ' y \neq \iota ' x . \iota ' x \cup \iota ' y \neq \iota ' y :$

[*13·12] $\supset \vdash : \alpha = \iota ' x \cup \iota ' y . x \neq y . \supset . \alpha \neq \iota ' x . \alpha \neq \iota ' y$ (2)

$\vdash . (1) . (2) . \supset \vdash :: \alpha = \iota ' x \cup \iota ' y . x \neq y . \supset :.$

$\beta \subset \alpha . \exists ! \beta . \beta \neq \alpha . \equiv : \beta = \iota ' x . \vee . \beta = \iota ' y :$

[*51·235] $\equiv : (\exists z) . z \in \alpha . \beta = \iota ' z :$

[*37·6] $\equiv : \beta \in \iota '' \alpha$ (3)

$\vdash . (3) . *11\cdot11\cdot35 . *54\cdot101 . \supset \vdash .$ Prop

***54·43.** $\vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

*Dem.*

$\vdash . *54\cdot26 . \supset \vdash :. \alpha = \iota ' x . \beta = \iota ' y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

[*51·231] $\equiv . \iota ' x \cap \iota ' y = \Lambda .$

[*13·12] $\equiv . \alpha \cap \beta = \Lambda$ (1)

$\vdash . (1) . *11\cdot11\cdot35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota ' x . \beta = \iota ' y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda$ (2)

$\vdash . (2) . *11\cdot54 . *52\cdot1 . \supset \vdash .$ Prop

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

## 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

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

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

$$
\dfrac{\dfrac{\dfrac{\dfrac{\overline{P \Rightarrow Q(y), P} \qquad \overline{P, Q(y) \Rightarrow Q(y)}}{P,\ P \to Q(y) \Rightarrow Q(y)}\ (\to l)}{P,\ \forall x\,(P \to Q(x)) \Rightarrow Q(y)}\ (\forall l)}{P,\ \forall x\,(P \to Q(x)) \Rightarrow \forall y\, Q(y)}\ (\forall r)}{\forall x\,(P \to Q(x)) \Rightarrow P \to \forall y\, Q(y)}\ (\to r)
$$

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:

$$\exists z\, (P(z) \to P(a) \land P(b))$$

## Part of the $\exists$ Distributive Law

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{P(x) \Rightarrow P(x), Q(x)}}{P(x) \Rightarrow P(x) \lor Q(x)}\,(\lor r)}{P(x) \Rightarrow \exists y\,(P(y) \lor Q(y))}\,(\exists r)}{\exists x\,P(x) \Rightarrow \exists y\,(P(y) \lor Q(y))}\,(\exists l) \qquad \dfrac{\text{similar}}{\exists x\,Q(x) \Rightarrow \exists y\ \ldots}\,(\exists l)}{\exists x\,P(x) \lor \exists x\,Q(x) \Rightarrow \exists y\,(P(y) \lor Q(y))}\,(\lor l)$$

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

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

## A Failed Proof

$$\dfrac{\dfrac{\dfrac{\dfrac{P(x), Q(y) \Rightarrow P(x) \land Q(x)}{P(x), Q(y) \Rightarrow \exists z\,(P(z) \land Q(z))}\,(\exists r)}{P(x), \exists x\,Q(x) \Rightarrow \exists z\,(P(z) \land Q(z))}\,(\exists l)}{\exists x\,P(x), \exists x\,Q(x) \Rightarrow \exists z\,(P(z) \land Q(z))}\,(\exists l)}{\exists x\,P(x) \land \exists x\,Q(x) \Rightarrow \exists z\,(P(z) \land Q(z))}\,(\land l)$$

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

$$\neg K_1 \lor \cdots \lor \neg K_m \lor L_1 \lor \cdots \lor L_n$$

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

Kowalski notation: $K_1, \cdots, K_m \to L_1, \cdots, L_n$

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

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

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

## Outline of Clause Form Methods

To prove $A$, get a contradiction from $\neg A$:

1. Translate $\neg A$ into CNF as $A_1 \wedge \cdots \wedge A_m$
2. This is the set of clauses $A_1, \ldots, A_m$
3. Transform this clause set, preserving satisfiability

Deducing the empty clause shows unsatisfiability, refuting $\neg A$.

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

The basis for SAT solvers and resolution provers.

## Clause Methods: Historical Background

Herbrand's theorem (1930) reduces first-order logic to propositional.

The prospect of fully automatic mathematics attracted logicians:

W V O Quine, Paul Gilmore, Martin Davis, Hilary Putnam, ...

- Sequent calculus: handles quantifiers but useless for big problems
- Conversion to DNF (1960): shows unsatisfiability; exponential time
- Davis–Putnam and DPLL (1962): good only for propositional logic
- J. A. Robinson's resolution and unification (1965)

## Aside: Why Does a Contradiction imply Everything?

A challenge to Russell: “Given $1 = 0$, prove that you are the Pope.”

Russell: “Then $2 = 1 \ldots$

and the set $\{\text{Russell}, \text{Pope}\}$ has only one element.”

A special case if $a$ and $b$ are integers, reals, etc:

$$\text{if } 1 = 0 \text{ then } a = a \times 1 = a \times 0 = b \times 0 = b \times 1 = b$$

hence $a = b$, and also $0 < 0$ and therefore $a < b$, etc.

## The Davis-Putnam-Logeman-Loveland Method

1. Delete tautological clauses: $\{P, \neg P, \ldots\}$
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 $P \mapsto$ **t**, $Q \mapsto$ **f**, $R \mapsto$ **f**

## Example of a Case Split on $P$

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

| | | | | |
|---|---|---|---|---|
| $\{\neg Q, R\}$ | $\{\neg R, Q\}$ | $\{Q, R\}$ | $\{\neg Q\}$ | 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 unsatisfiable!

## SAT solvers in the Real World

- Progressed from joke to killer technology in 10 years.
- Princeton's zChaff (2001) 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 $B \lor A$ and $\neg B \lor C$ infer $A \lor C$

In set notation,

$$\frac{\{B, A_1, \ldots, A_m\} \qquad \{\neg B, C_1, \ldots, C_n\}}{\{A_1, \ldots, A_m, C_1, \ldots, C_n\}}$$

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

$$\frac{\{B\} \qquad \{\neg B, C_1, \ldots, C_n\}}{\{C_1, \ldots, C_n\}} \qquad\qquad \frac{\{B\} \qquad \{\neg B\}}{\square}$$

*but resolution is only useful for first-order logic

## Simple Example: Proving $P \land Q \to Q \land P$

Hint: use $\neg(A \to B) \simeq A \land \neg B$

1. Negate! $\neg[P \land Q \to Q \land P]$
2. Push $\neg$ in: $(P \land Q) \land \neg(Q \land P)$

   $(P \land Q) \land (\neg Q \lor \neg P)$

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 $\{\neg R\}$.

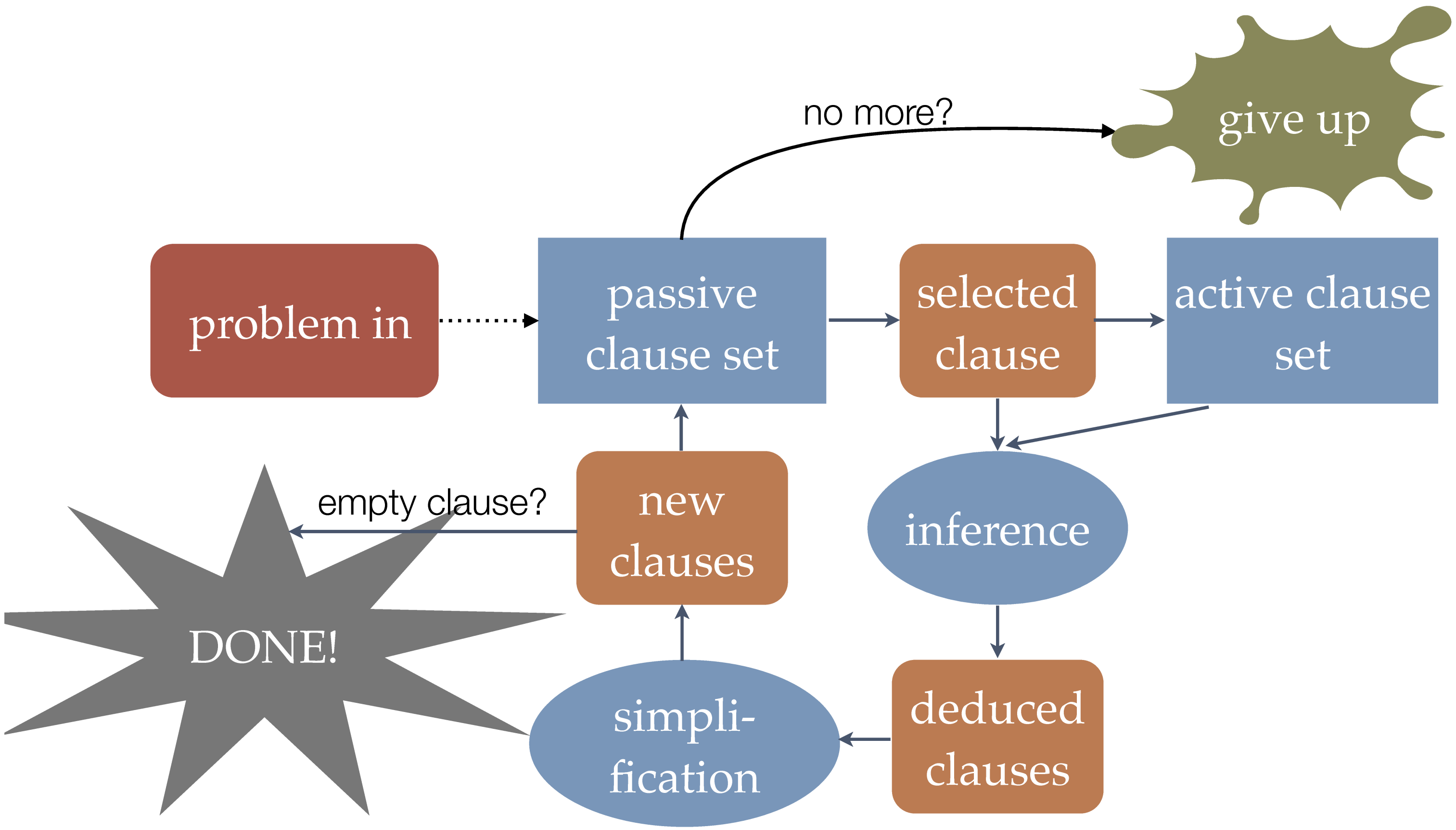
Resolve $\{R\}$ and $\{\neg R\}$ getting $\square$, contradiction.

# 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.


no more?
give up
problem in
passive clause set
selected clause
active clause set
empty clause?
new clauses
inference
DONE!
simpli-fication
deduced clauses

## A Resolution Heuristic: Clause Selection by Weight

assign weights to constants (penalising “bad” constants)

the weight of a clause is the sum of the weights of its constants

the lightest clause is likely to be shortest or the “simplest”

But we want to keep completeness: all theorems can be proved

completeness requires fairness: every clause is selected eventually

# Other Heuristics and Hacks for Resolution

Orderings to focus the search on specific literals and exploit symmetry

Subsumption to delete redundant clauses $\{P, Q\}$ subsumes $\{P, Q, R\}$

Indexing: elaborate data structures for speed

Preprocessing: removing tautologies, symmetries . . . at the very start

DPLL is extremely effective—

but in its pure form only works for propositional logic

How can we extend it to quantifiers?

How do we come up with witnessing terms?

- In 1962, the idea was ad-hoc guessing (still being used today)
- Robinson's answer in 1965: unification

# Reducing FOL to Propositional Logic

| | |
|---|---|
| NNF: | Leaving only $\forall$, $\exists$, $\wedge$, $\vee$, and $\neg$ on atoms |
| Skolemize: | Remove quantifiers, preserving satisfiability |
| 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:

$$\forall x_1 \, (\cdots \forall x_2 \, (\cdots \forall x_k \, (\cdots \exists y \, A \cdots ) \cdots ) \cdots )$$

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

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

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

Repeat until no $\exists$ quantifiers remain

# Example of Conversion to Clauses

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

| | |
|---|---|
| $\neg\, [\exists x\, [P(x) \to \forall y\, P(y)]]$ | negated goal |
| $\forall x\, [P(x) \land \exists y\, \neg P(y)]$ | conversion to NNF |
| $\forall x\, [P(x) \land \neg P(f(x))]$ | Skolem term $f(x)$ |
| $\{P(x)\} \qquad \{\neg P(f(x))\}$ | Final clauses |

## Correctness of Skolemization

The formula $\forall x\, \exists y\, A$ is satisfiable

$\iff$ 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 $D \to D$ 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 satisfiable.

# Simplifying the Search for Models

$S$ is satisfiable if even one model makes all of its clauses true.

There are infinitely many models to consider!

Also many duplicates: “states of the USA” and “the integers 1 to 50”

Fortunately, canonical models exist.

- They have a uniform structure based on the language’s syntax.
- They satisfy the clauses if any model does.

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

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

$$H_{i+1} \stackrel{\text{def}}{=} H_i \cup \{f(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in H_i$$
$$\text{and } f \text{ is an } n\text{-place function symbol in } S\}$$

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

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

$H$ consists of all ground terms built using symbols from $S$.

Our semantics will interpret function symbols by **operations on terms**.

## The Herbrand Semantics of Terms

Herbrand models are syntactic: every constant stands for itself.

Every function symbol stands for a term-forming operation:

$f$ denotes the function that puts '$f$' in front of the given arguments.

The Herbrand universe with 0, 1, minus and binary $+$ is

$$0 \quad 1 \quad -0 \quad -1 \quad 0+0 \quad 0+1 \quad 1+0 \quad 1+1 \quad --0 \cdots$$

$X + 0$ is not equal to $X$!!

Every ground term denotes itself.

This is the promised uniform structure!

## The Herbrand Semantics of Predicates

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

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

# Example of an Herbrand Model

$$\left.\begin{aligned} \neg even(1) \\ even(2) \\ even(X \cdot Y) \leftarrow even(X), even(Y) \end{aligned}\right\} \textit{clauses}$$

$$H = \{1, 2, 1 \cdot 1, 1 \cdot 2, 2 \cdot 1, 2 \cdot 2, 1 \cdot (1 \cdot 1), \ldots\}$$

$$HB = \{even(1), even(2), even(1 \cdot 1), even(1 \cdot 2), \ldots\}$$

$$I[even] = \{even(2), even(1 \cdot 2), even(2 \cdot 1), even(2 \cdot 2), \ldots\}$$

(for the model where · denotes product; could instead denote sum!)

## Herbrand's Theorem for a Set of Clauses, $S$

*$S$ is unsatisfiable $\iff$ no Herbrand interpretation satisfies $S$*

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

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

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

# 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

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

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

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

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

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

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

[Most general is a notion of minimality. E.g. to unify

$$f(x, y) \qquad f(a, z)$$

we could get $f(a, y)$ or $f(a, z)$ but not $f(a, a)$.]

# Reminder: the Scope of Variables in a Clause

Variables are local to a clause

Variables must be renamed prior to each resolution to prevent clashes

[renaming variables apart]

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

Resolution tends to make clauses longer!

Though $\{P, P, Q\} = \{P, Q\}$ simply because they are sets.

A factoring inference collapses unifiable literals in one clause:

$$\frac{\{B_1, \ldots, B_k, A_1, \ldots, A_m\}}{\{B_1, A_1, \ldots, A_m\}\sigma} \qquad \text{provided } B_1\sigma = \cdots = B_k\sigma$$

Resolution + factoring is **complete for first-order logic**:

Every valid formula will be proved (given enough space and time)

# Example of Resolution with Factoring

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

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

the lack of unit clauses shows we need factoring

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

And now, resolution yields the empty clause!

# A Non-Trivial Proof

$$\exists x\,[P \to Q(x)] \land \exists x\,[Q(x) \to P] \to \exists x\,[P \leftrightarrow Q(x)]$$

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 $\square$

## The Problem of Relevance

Real-world problems may have 1000s of irrelevant clauses

For example, axioms of background theories

Our examples here are minimal: every clause is necessary

Part of the theorem prover's task is to keep focused

Heuristics to constrain the proof effort to the negated conjecture

# 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

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

## The Origins of Prolog

People hoped theorem proving could “think”: robot planning, , . . .

Those early experiments with resolution were disappointing!

Restricted forms of resolution were studied to improve performance

- A procedural interpretation of Horn clauses
- Cool behaviours not possible in standard languages or even LISP
- Plus lots of non-logical hacks for arithmetic, I/O, etc.

[Alain Colmerauer, Phillipe Roussel, Robert Kowalski]

# Horn (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" $A_1, \ldots, A_m$ is

$$B \leftarrow A_1, \ldots, A_m$$

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

$$\leftarrow A_1, \ldots, A_m$$

# 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).
*                                    :- parent(andrew,Y).
Y=beatrice                                          :- □.
```

$\star$ = backtracking choice point

16 solutions including `cousin(william,william)`

and `cousin(william,henry)`

## Some Prolog Applications

- Deductive databases, as we've just seen
- Definite clause grammars: a direct way to code natural language syntax and semantics into Prolog systems
- AI applications based on backtracking (replacing specialised languages like Carl Hewitt's PLANNER)

In the 1980s, people went mad about Prolog

# Another FOL Proof Procedure: Model Elimination

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

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

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

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

$$\vdots$$

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

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

# A Survey of Automatic Theorem Provers

Model Elimination: Prolog Technology Theorem Prover, SETHEO, etc.

Connection calculus (evolved from model elimination): leanCoP

Higher-Order Logic: TPS, LEO-III, Satallax

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

First-order Resolution: E (eprover), SPASS, Vampire, . . .

# The Limitations of Pure Logic

Imagine using resolution or DPLL to prove

$$\frac{354}{113} < \pi < \frac{355}{113}$$

Program verification involves

integers • reals • lists • booleans • arrays

**How can we combine logical reasoning with specialised theories?**

Decision procedures are one answer.

## Decision Problems

Precise yes/no questions:

is $n$ prime or not? Is this string accepted by that grammar?

Unfortunately, most decision problems for logic are hard:

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

Polynomial arithmetic is decidable; hence, 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 arithmetic 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 $i = 1, \ldots, m$:

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 |
| $x \leq z$ | $y + z \leq 0$ | $y \leq -1$ | |
| $-x + y + 2z \leq 0$ | | | |
| $-z \leq -1$ | $-z \leq -1$ | | |

## Two Worked Out Examples

$$\begin{array}{rr} & x \le y \\ (+) & -x + y + 2z \le 0 \\ \hline & y + 2z \le y \end{array} \qquad \begin{array}{rr} & x \le z \\ (+) & -x + y + 2z \le 0 \\ \hline & y + 2z \le z \end{array}$$

and so $z \le 0$ and so $y + z \le 0$

## Quantifier Elimination (QE)

Skolemization removes quantifiers but only preserves satisfiability.

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

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

$$\exists xy\,(2x < y \land y < x) \iff \exists x\, 2x < x \iff \mathbf{t}$$

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

QE for real polynomial arithmetic:

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

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

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

Doing logic with DNF is poor

Is there a better way? Maybe using 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

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

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

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

# SMT Example (Continued)

Now a case split on $3a > 2$ returns a "model":

$b < a, c \neq 0, 2a < b, 3a > 2$

But the decision proc. finds these contradictory, killing the $3a > 2$ case

It returns a new clause:

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

Finally get a satisfiable result: $b < a \land c \neq 0 \land 2a < b \land a < 0$

# Remarks on the Previous Example

DPLL works only for propositional formulas!

We should properly write

$$\{\boxed{c = 0}, \boxed{2a < b}\} \qquad \{\neg\boxed{c = 0}, \neg\boxed{b < a}\} \qquad \cdots$$

The DPLL part knows nothing about arithmetic.

SMT makes two independent reasoners cooperate!

# 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.

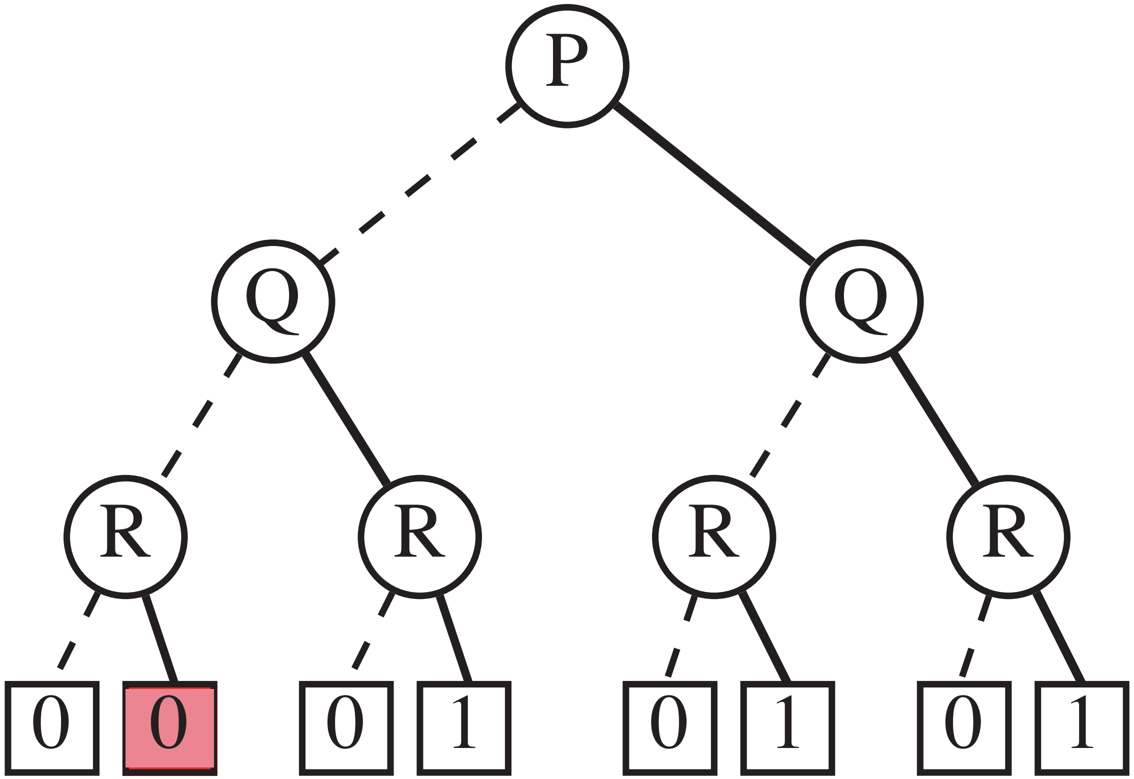
- ordered propositional symbols (the variables)
- sharing of identical subtrees
- hashing and other optimisations

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

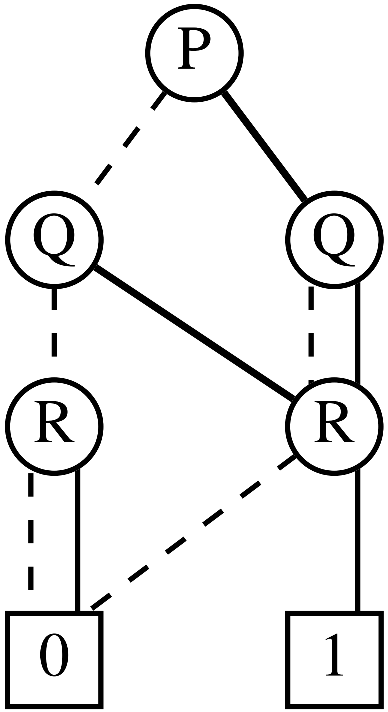
Exhibits models (paths to 1) if the formula is satisfiable.

Excellent for verifying digital circuits, with many other applications.

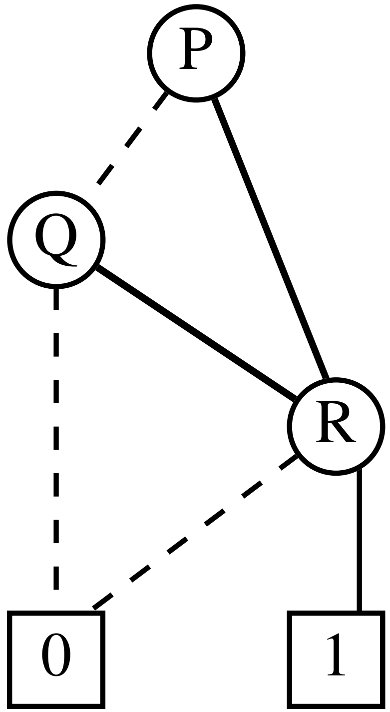
## Decision Diagram for $(P \lor Q) \land R$



# Converting a Decision Diagram to a BDD




No duplicates




No redundant tests

# Efficiently Converting a Formula to a BDD

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.

BDD packages can handle 100 million nodes

## Canonical Form Algorithm for Negation

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

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

(We copy the BDD but exchange the 1 and 0 at the bottom.)

The treatment of $Z \to 0$ and $Z \leftrightarrow 0$ turns out the same way.

## Canonical Form Algorithm for Binary Connectives

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

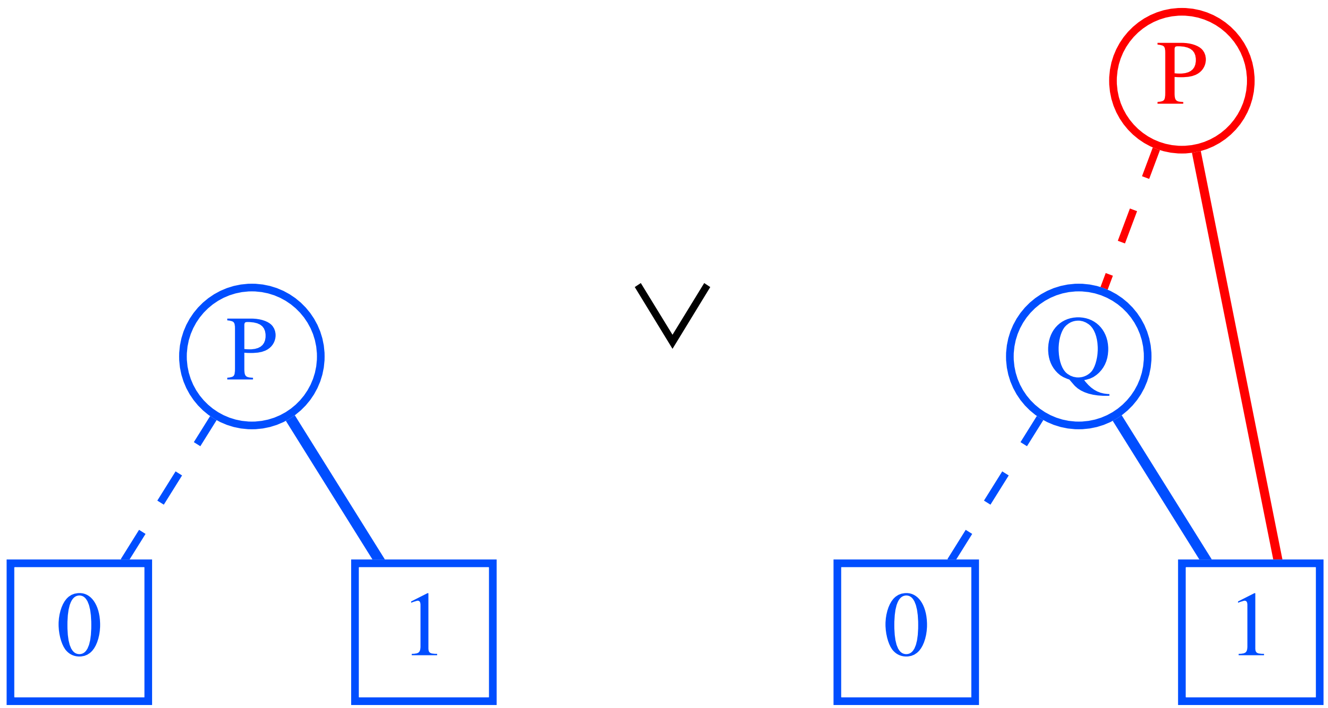
*Trivial if either operand is 1 or 0.*

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

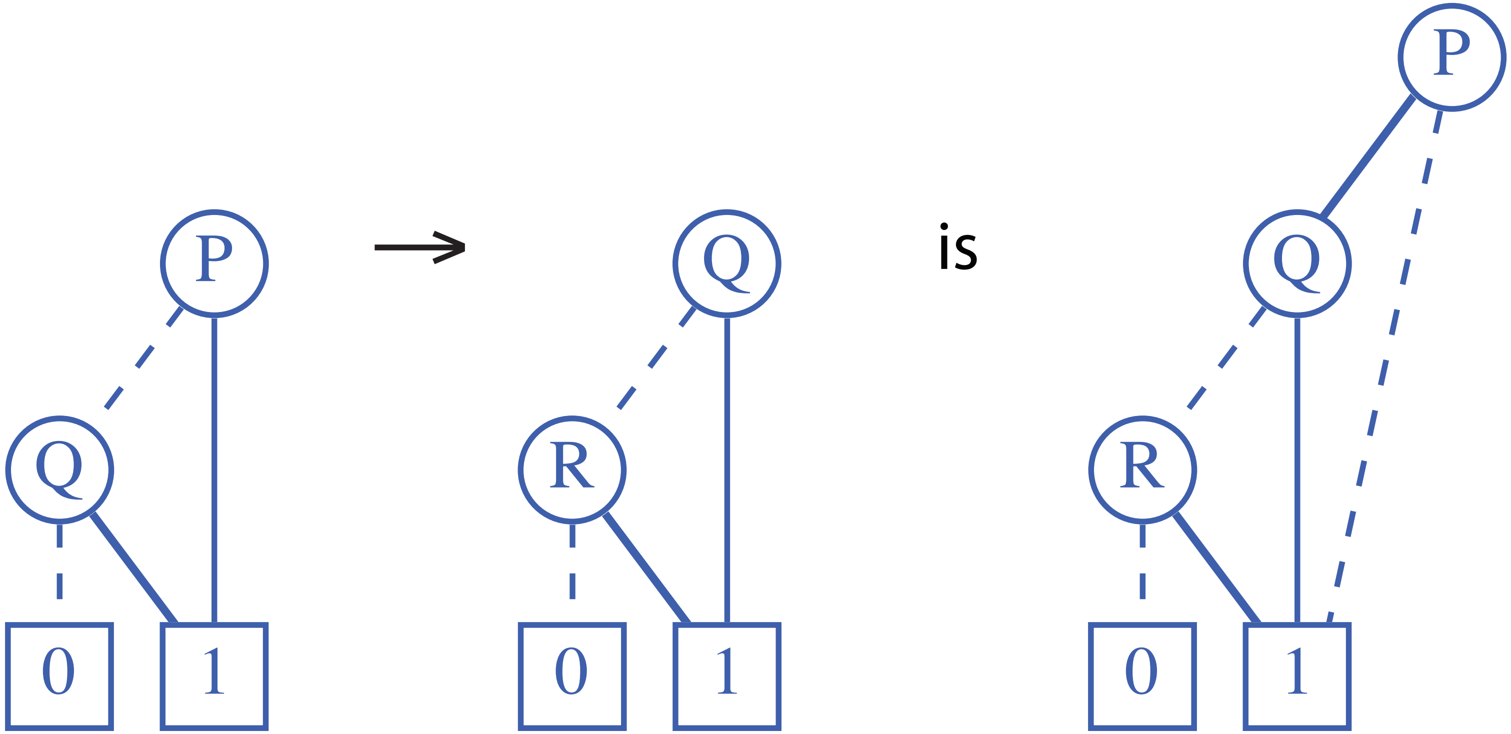
- If $P = P'$ then recursively convert $\textbf{if}(P,\ X \wedge X',\ Y \wedge Y')$.
- If $P < P'$ then recursively convert $\textbf{if}(P,\ X \wedge Z',\ Y \wedge Z')$.
- If $P > P'$ then recursively convert $\textbf{if}(P',\ Z \wedge X',\ Z \wedge Y')$.

similarly for $Z \vee Z'$, $Z \to Z'$ and even $Z \leftrightarrow Z'$

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

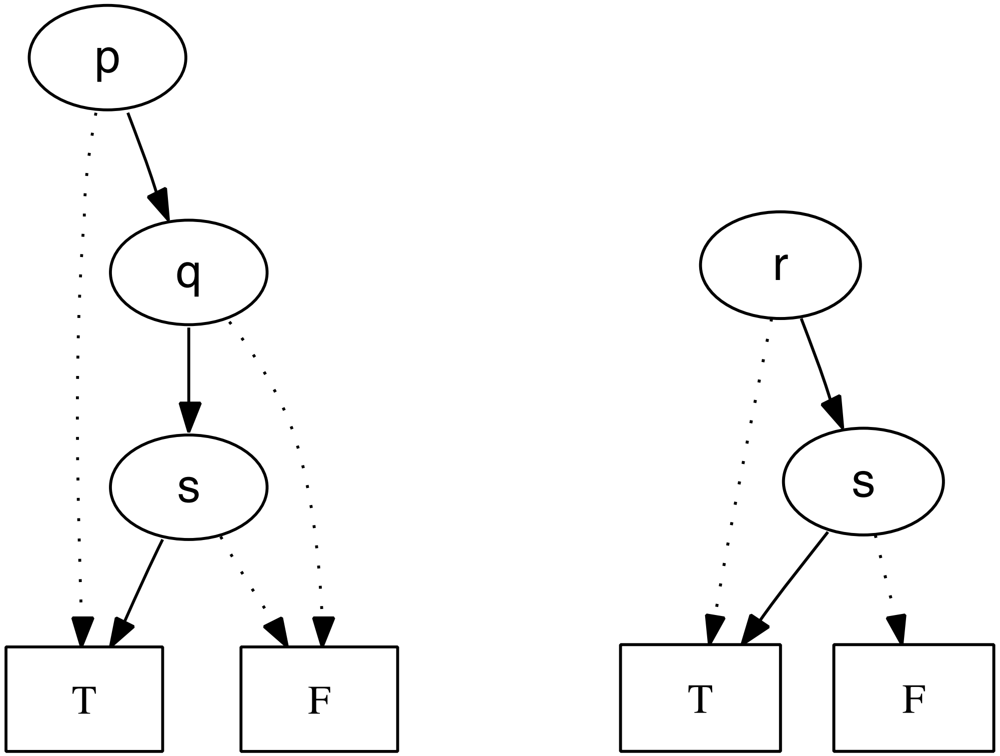


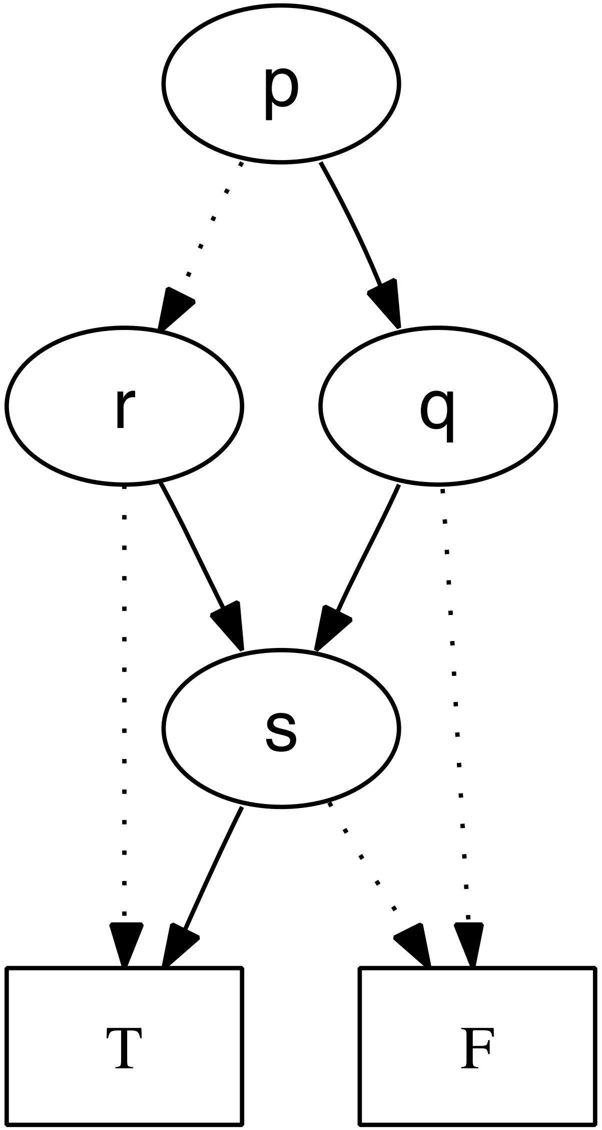
# Canonical Form of $P \lor Q \to Q \lor R$



# A Exam Question: 2010 P5 Q5

BDD for $[p \to (q \land s)] \ \land \ [s \lor (r \to s)]$, alphabetic ordering.




p
r
q
s
T
F

# Tricks for Doing BDDs by Hand

“Two Finger Method”

Treat the cases of the variables strictly in order

Insert “redundant tests” to make the top variables match

Be careful to preserve sharing rather than copy

**If a variable repeats on any path, you’ve gone wrong!**

## 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 **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.

# BDDs versus SAT Solvers

Timeline: original DPLL (1962), BDDs (1986), faster SAT (2001)

| BDDs | SAT solvers |
|---|---|
| all counterexamples | one counterexample* |
| full logic including XOR | clause form only |
| for hardware: adders, latches | general constraint problems |
| used in model checkers | combined with decision procs |

*Good for counterexample-driven abstraction refinement

## Final Observations

The variable ordering is crucial. Consider this formula:

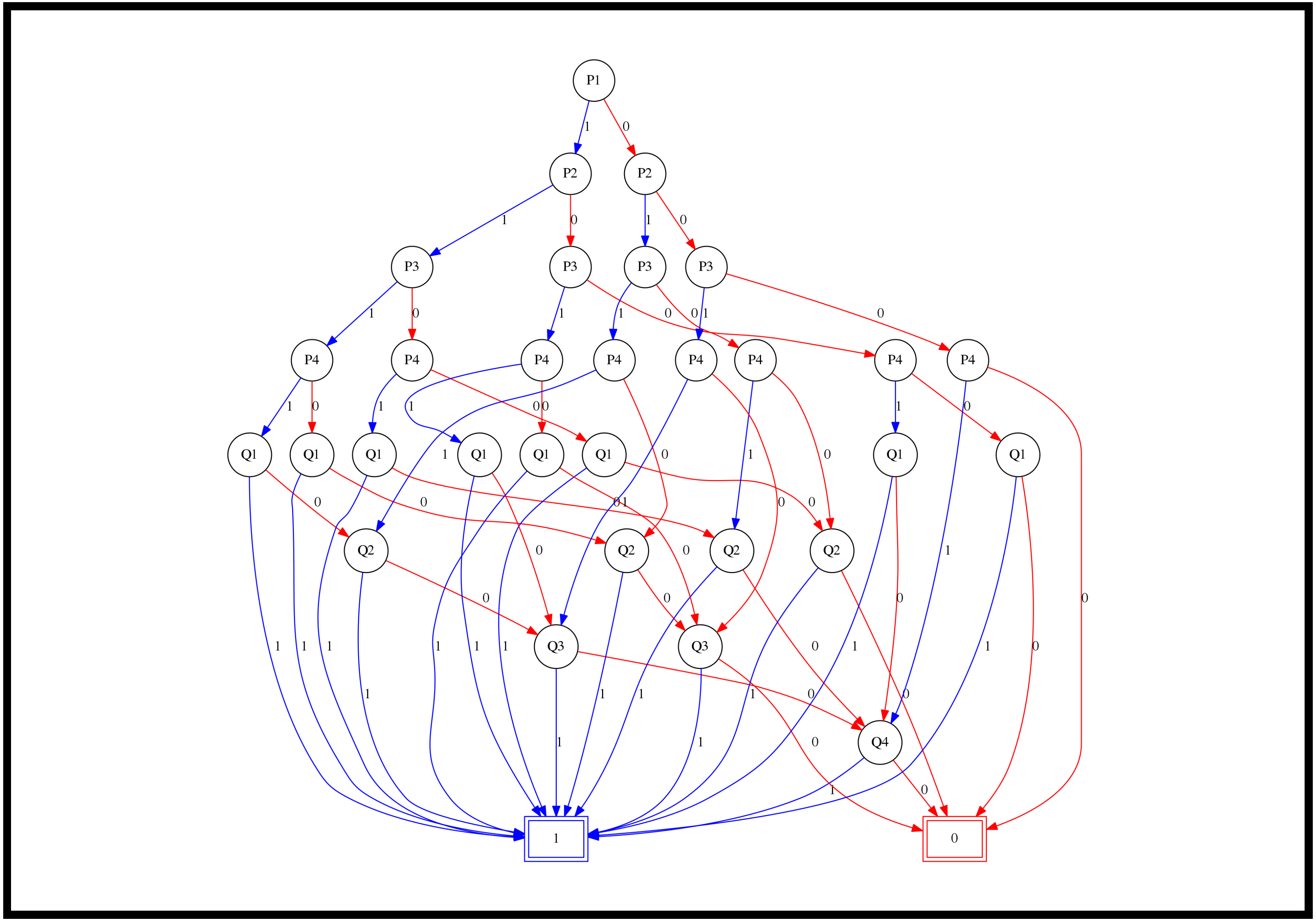
$$(P_1 \land Q_1) \lor \cdots \lor (P_n \land Q_n)$$

A good ordering is $P_1 < Q_1 < \cdots < P_n < Q_n$

- the BDD is linear: exactly $2n$ nodes

A bad ordering is $P_1 < \cdots < P_n < Q_1 < \cdots < Q_n$

- the BDD is exponential: exactly $2^{n+1}$ nodes


P1
P2 P2
P3 P3 P3 P3
P4 P4 P4 P4 P4 P4 P4 P4
Q1 Q1 Q1 Q1 Q1 Q1 Q1 Q1
Q2 Q2 Q2 Q2
Q3 Q3
Q4
1
0

# Modal Operators

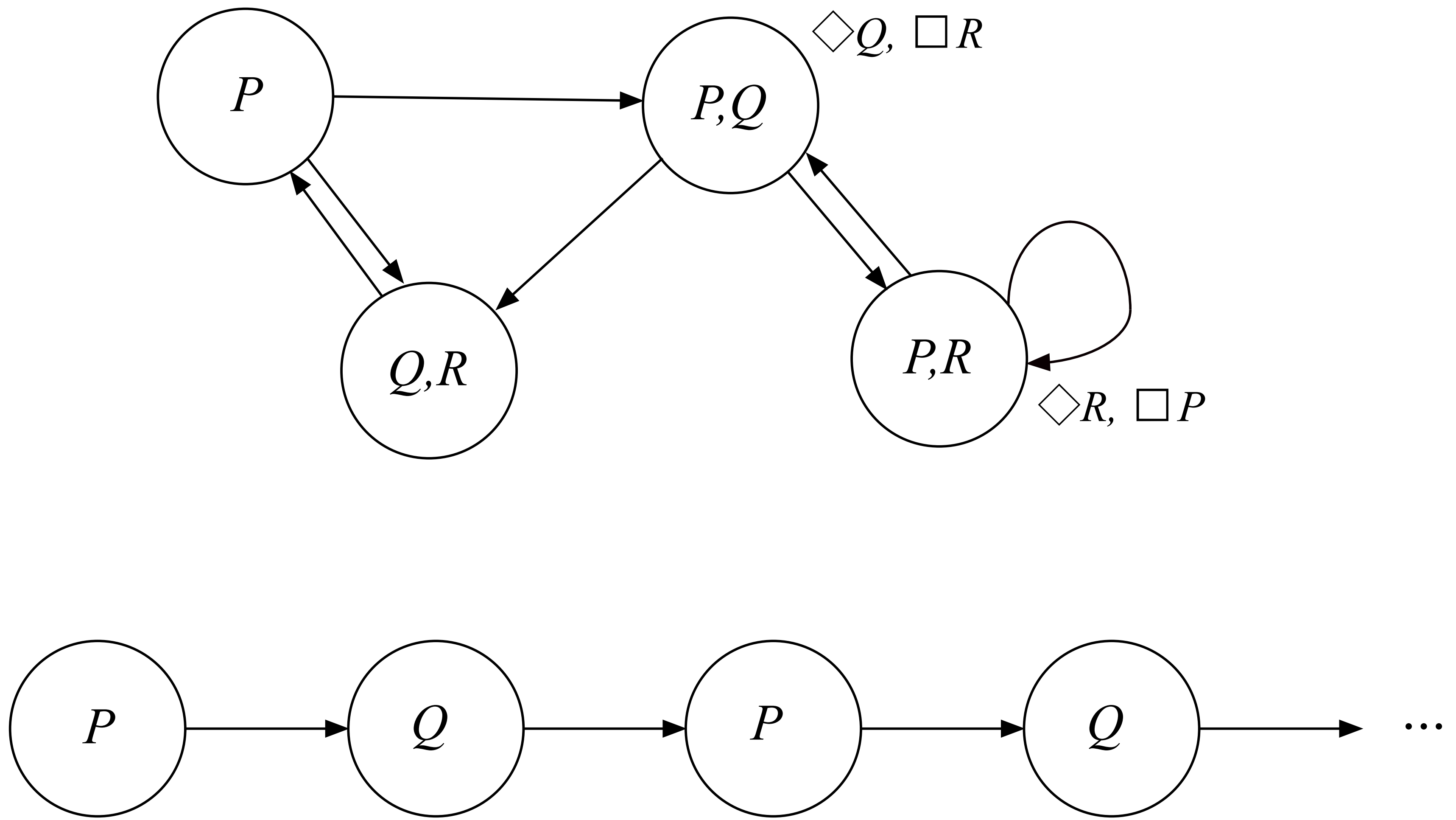
$W$: set of possible worlds (machine states, future times, . . .)

$R$: accessibility relation between worlds

$(W, R)$ is called a modal frame or Kripke frame

$\left.\begin{array}{l}\Box A \text{ means } A \text{ is necessarily true}\\ \Diamond A \text{ means } A \text{ is possibly true}\end{array}\right\}$ in all worlds accessible from here

$\neg\Diamond A \simeq \Box\neg A$ $\qquad$ $A$ cannot be true $\iff$ $A$ must be false



P
P,Q
◇Q, □R
Q,R
P,R
◇R, □P
P
Q
P
Q
...

## 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 $w$

$$
\begin{aligned}
w \Vdash P &\iff w \in I(P) \\
w \Vdash A \land B &\iff w \Vdash A \text{ and } w \Vdash B \\
w \Vdash \Box A &\iff v \Vdash A \text{ for all } v \text{ such that } R(w, v) \\
w \Vdash \Diamond A &\iff v \Vdash A \text{ for some } v \text{ such that } R(w, v)
\end{aligned}
$$

sometimes called Kripke semantics

# Truth and Validity in Modal Logic

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

$$
\begin{array}{rl}
w \Vdash A & \text{means } A \text{ is true in world } w \\
\models_{W,R,I} A & \text{means } w \Vdash A \text{ for all } w \text{ in } W \\
\models_{W,R} A & \text{means } w \Vdash A \text{ for all } w \text{ and all } I
\end{array}
$$

$\models A$ 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 $\mathbb{K}$

Extend your favourite propositional proof system with an axiom:

$$\text{Dist} \qquad \Box(A \to B) \to (\Box A \to \Box B)$$

And with an 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 $\mathrm{K}$

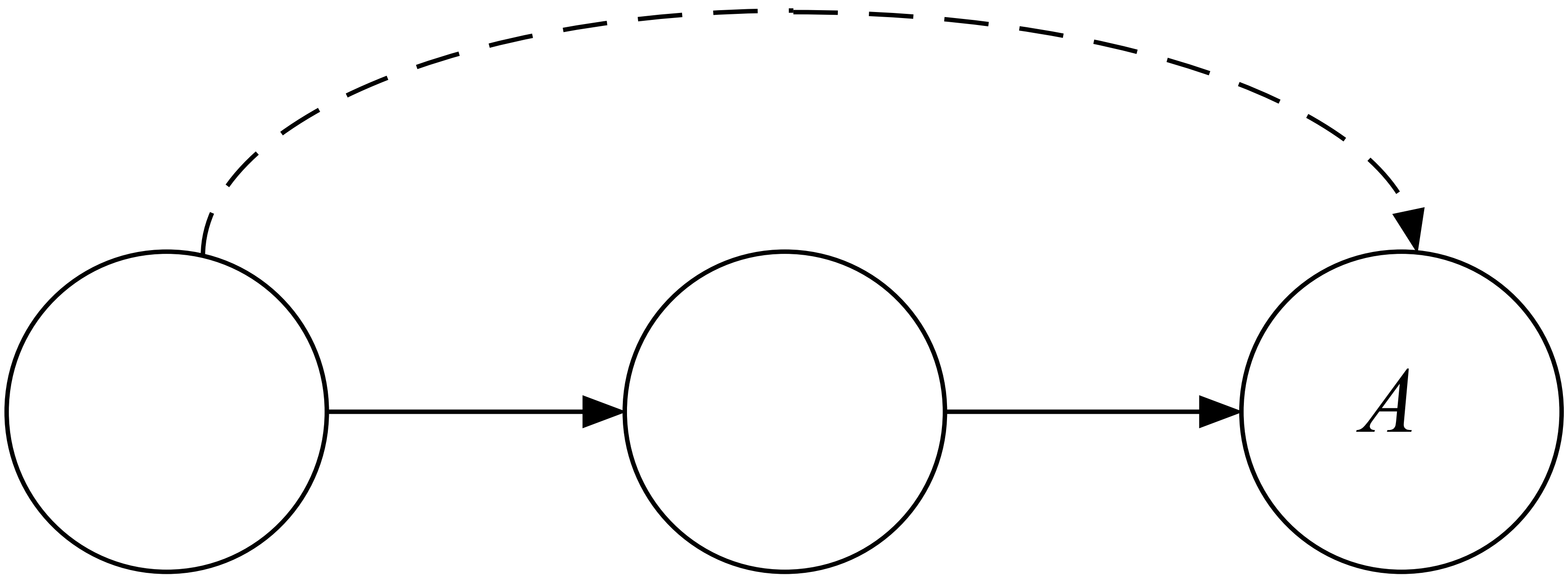
Add axioms to constrain the accessibility relation:

| | | | |
|---|---|---|---|
| T | $\Box A \rightarrow A$ | (reflexive) | logic $\mathrm{T}$ |
| 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.

# Justifying Axiom 4 (Transitivity)




So if $\Box A$ then $\Box\Box A$

## $S4$ as a Temporal Logic

$\Box A$ means $A$ holds at every future time

$\Diamond A$ means $A$ holds some time in the future

$\Box\Diamond A$ means $A$ holds infinitely often

$\Diamond\Box A$ means $A$ will become permanently true after some time

$\Box\neg(P \land Q)$ implies mutual exclusion for $P$, $Q$

$\Box(P \to \Diamond Q)$ means $P$ will eventually trigger $Q$

What about $\Box\Box A$ and $\Diamond\Diamond A$?

# Extra Sequent Calculus Rules for $S4$

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

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

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \Rightarrow B, A} \qquad \overline{B, A \Rightarrow B}}{A \to B, A \Rightarrow B}\,(\to l)}{A \to B, \Box A \Rightarrow B}\,(\Box l)}{\Box(A \to B), \Box A \Rightarrow B}\,(\Box l)}{\Box(A \to B), \Box A \Rightarrow \Box B}\,(\Box r)$$

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

Must apply $(\Box r)$ first!

# Part of an "Operator String Equivalence"

$$
\dfrac{\dfrac{\dfrac{\dfrac{\overline{\Diamond A \Rightarrow \Diamond A}}{\Box\Diamond A \Rightarrow \Diamond A}\,(\Box l)}{\Diamond\Box\Diamond A \Rightarrow \Diamond A}\,(\Diamond l)}{\Box\Diamond\Box\Diamond A \Rightarrow \Diamond A}\,(\Box l)}{\Box\Diamond\Box\Diamond A \Rightarrow \Box\Diamond A}\,(\Box r)
$$

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

$$\dfrac{\dfrac{\Rightarrow A}{\Rightarrow \Diamond A}\,(\Diamond r)}{A \Rightarrow \Box\Diamond A}\,(\Box r) \qquad \left(\text{versus} \quad \dfrac{\dfrac{\Box A \Rightarrow A}{\Box A \Rightarrow \Diamond A}\,(\Diamond r)}{\Box A \Rightarrow \Box\Diamond A}\,(\Box r)\right)$$

$$\dfrac{\dfrac{B \Rightarrow A \land B}{B \Rightarrow \Diamond(A \land B)}\,(\Diamond r)}{\Diamond A, \Diamond B \Rightarrow \Diamond(A \land B)}\,(\Diamond l)$$

Can extract a countermodel from the proof attempt

## Some Remarks on Model Checking

- Temporal formulas can be proved by state enumeration
- . . . using specially designed temporal logics
- Typically extend the language: “until” modalities, etc.
- branching-time vs linear-time; discrete vs continuous time
- Applications to verifying hardware or concurrent systems

examples of model-checkers: SPIN, NuSMV (which is BDD-based)

# Simplifying the Sequent Calculus

7 connectives (or 9 for modal logic):

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

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

Idea! Work in Negation Normal Form

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

Sequents need one side only!

## Tableau Calculus: Left-Only

$$\frac{}{\neg A, A, \Gamma \Rightarrow} \text{(basic)} \qquad \frac{\neg A, \Gamma \Rightarrow \qquad A, \Gamma \Rightarrow}{\Gamma \Rightarrow} \text{(cut)}$$

$$\frac{A, B, \Gamma \Rightarrow}{A \land B, \Gamma \Rightarrow} (\land l) \qquad \frac{A, \Gamma \Rightarrow \qquad B, \Gamma \Rightarrow}{A \lor B, \Gamma \Rightarrow} (\lor l)$$

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

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

# Tableau Rules for $S4$

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

$\Gamma^* \stackrel{\text{def}}{=} \{\Box B \mid \Box B \in \Gamma\}$ Erase non-$\Box$ 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 $\forall x\,(P \to Q(x)) \to [P \to \forall y\, Q(y)]$

Negate and convert to NNF:

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

$$\cfrac{\cfrac{\cfrac{\overline{P,\ \neg Q(y),\ \neg P \Rightarrow} \qquad \overline{P,\ \neg Q(y),\ Q(y) \Rightarrow}}{P,\ \neg Q(y),\ \neg P \lor Q(y) \Rightarrow}\,(\lor l)}{P,\ \neg Q(y),\ \forall x\,(\neg P \lor Q(x)) \Rightarrow}\,(\forall l)}{P,\ \exists y\, \neg Q(y),\ \forall x\,(\neg P \lor Q(x)) \Rightarrow}\,(\exists l)$$

# The Free-Variable Tableau Calculus

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

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

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 $(\exists l)$? Do not use it! Instead, Skolemize!

## Skolemization from NNF

Recall e.g. that we Skolemize

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

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

Example: proving $\exists x\, \forall y\, [P(x) \to P(y)]$:

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

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

Push in the $\forall x$ : $(\forall x\, P(x)) \land (\exists y\, \neg P(y))$

Skolemize: $\forall x\, P(x) \land \neg P(a)$

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

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{y \mapsto f(z)}{P(y),\, \neg P(f(y)),\, P(z),\, \neg P(f(z)) \Rightarrow}\,(\text{basic})}{P(y),\, \neg P(f(y)),\, P(z) \land \neg P(f(z)) \Rightarrow}\,(\land l)}{P(y),\, \neg P(f(y)),\, \forall x\,[P(x) \land \neg P(f(x))] \Rightarrow}\,(\forall l)}{P(y) \land \neg P(f(y)),\, \forall x\,[P(x) \land \neg P(f(x))] \Rightarrow}\,(\land l)}{\forall x\,[P(x) \land \neg P(f(x))] \Rightarrow}\,(\forall l)$$

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

## A Failed Proof

Try to prove $\forall x\,[P(x) \lor Q(x)] \to [\forall x\, P(x) \lor \forall x\, Q(x)]$

NNF: $\exists x\, \neg P(x) \land \exists x\, \neg Q(x) \land \forall x\,[P(x) \lor Q(x)] \Rightarrow$

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

$$\dfrac{\dfrac{\dfrac{y \mapsto a}{\neg P(a),\ \neg Q(b),\ P(y) \Rightarrow} \qquad \dfrac{y \mapsto b???}{\neg P(a),\ \neg Q(b),\ Q(y) \Rightarrow}}{\neg P(a),\ \neg Q(b),\ P(y) \lor Q(y) \Rightarrow}\,(\lor l)}{\neg P(a),\ \neg Q(b),\ \forall x\,[P(x) \lor Q(x)] \Rightarrow}\,(\forall l)$$

# The Various Tableau Calculi

Today we've seen two separate calculi:

1. First-order tableaux without unification
2. First-order tableaux with unification (free-variable tableau)

mentioned previously: connection tableaux

(related to the model elimination calculus)

All these lend themselves to compact implementations!

# 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).
```