Logic and Proof

Computer Science Tripos Part IB
Michaelmas Term

Lawrence C Paulson
Computer Laboratory
University of Cambridge

lp15@cam.ac.uk

Copyright © 2004 by Lawrence C. Paulson
<table>
<thead>
<tr>
<th></th>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Propositional Logic</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Gentzen’s Logical Calculi</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>Binary Decision Diagrams</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>First-Order Logic</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Formal Reasoning in First-Order Logic</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>Davis-Putnam &amp; Propositional Resolution</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>Skolem Functions and Herbrand’s Theorem</td>
<td>38</td>
</tr>
<tr>
<td>9</td>
<td>Unification</td>
<td>43</td>
</tr>
<tr>
<td>10</td>
<td>Resolution and Prolog</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>Modal Logics</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>Tableaux-Based Methods</td>
<td>59</td>
</tr>
</tbody>
</table>
Introduction to Logic

Logic concerns statements in some language.

The language can be informal (say English) 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

The meta-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 express general statements, or questions:

- What things are black?

Interpretations and Validity

An interpretation maps meta-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 $A$.

Lawrence C. Paulson

University of Cambridge
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:

- $\{X \text{ part of } Y, \ Y \text{ part of } Z, \ X \text{ NOT part of } Z\}$
- $\{n \text{ is a positive integer, } n \neq 1, \ n \neq 2, \ldots\}$

**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 $S \models A$.

- $\{X \text{ part of } Y, \ Y \text{ part of } Z\} \models X \text{ part of } Z$
- $\{n \neq 1, \ n \neq 2, \ldots\} \models n \text{ is NOT a positive integer}$

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

$\models A$ if and only if $A$ is valid, if and only if $\{\neg A\}$ is inconsistent.
We want to check \( A \) is valid.

Checking all interpretations can be effective — but what if there are infinitely many?

Let \( \{A_1, \ldots, A_n\} \models B \). If \( A_1, \ldots, A_n \) are true then \( B \) must be true. Write this as the inference rule

\[
\frac{A_1 \quad \ldots \quad A_n}{B}
\]

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 valid inference:

\[
\text{spoke part of wheel} \quad \text{wheel part of bike} \quad \text{spoke part of bike}
\]

An inference may be valid even if the premises are false!

\[
\text{cow part of chair} \quad \text{chair part of ant} \quad \text{cow part of ant}
\]
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.

Why Should the Language be Formal?

Consider this ‘definition’:
The least integer not definable using eight words
Greater than The number of atoms in the entire Universe
Also greater than The least integer not definable using eight words
• A formal language prevents AMBIGUITY.
### Syntax of Propositional Logic

- **P, Q, R, ...** propositional letter
- **t** true
- **f** false
- **¬A** not A
- **A ∧ B** A and B
- **A ∨ B** A or B
- **A → B** if A then B
- **A ↔ B** A if and only if B

### Semantics of Propositional Logic

¬, ∧, ∨, → and ↔ are truth-functional: functions of their operands.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>¬A</th>
<th>A ∧ B</th>
<th>A ∨ B</th>
<th>A → B</th>
<th>A ↔ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>f</td>
<td>t</td>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>t</td>
<td>f</td>
<td>t</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>t</td>
<td>f</td>
<td>f</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>
Interpretations of Propositional Logic

An interpretation is a function from the propositional letters to \{"t, f\}.

Interpretation \(I\) satisfies a formula \(A\) if the formula evaluates to \(t\).

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 \(\lnot A \lor B\).

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

\(A \models B\) if and only if \(\models A \rightarrow B\).

Equivalence

\(A \equiv B\) means \(A \models B\) and \(B \models A\).

\(A \equiv B\) if and only if \(\models A \leftrightarrow B\).
Equivalences

\[
\begin{align*}
A \land A & \simeq A \\
A \land B & \simeq B \land A \\
(A \land B) \land C & \simeq A \land (B \land C) \\
A \lor (B \land C) & \simeq (A \lor B) \land (A \lor C) \\
A \land \neg A & \simeq \bot \\
A \land t & \simeq A \\
A \land \neg A & \simeq \bot
\end{align*}
\]

Dual versions: exchange \(\land\) with \(\lor\) and \(t\) with \(\bot\) in any equivalence

Negation Normal Form

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

\[
\begin{align*}
A \leftrightarrow B & \simeq (A \rightarrow B) \land (B \rightarrow A) \\
A \rightarrow B & \simeq \neg A \lor B
\end{align*}
\]

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

\[
\begin{align*}
\neg\neg A & \simeq A \\
\neg (A \land B) & \simeq \neg A \lor \neg B \\
\neg (A \lor B) & \simeq \neg A \land \neg B
\end{align*}
\]
From NNF to Conjunctive Normal Form

3. Push disjunctions in, using distributive laws:

\[ A \lor (B \land C) \simeq (A \lor B) \land (A \lor C) \]
\[ (B \land C) \lor A \simeq (B \lor A) \land (C \lor A) \]

4. Simplify:
   - Delete any disjunction containing \( P \) and \( \neg P \)
   - Delete any disjunction that includes another: for example, in \((P \lor Q) \land P\), delete \( P \lor Q \).
   - Replace \((P \lor A) \land (\neg P \lor A)\) by \( A \)

Converting a Non-Tautology to CNF

\[ P \lor Q \rightarrow Q \lor R \]

1. Elim \( \rightarrow \): \( \neg(P \lor Q) \lor (Q \lor R) \)
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 \( P \mapsto t, \ Q \mapsto f, \ R \mapsto f \)
Tautology checking using CNF

\[((P \rightarrow Q) \rightarrow P) \rightarrow P\]

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

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

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

4. Simplify: $t \land t$

$t$ It's a tautology!
Slide 301

A Simple Proof System

Axiom Schemes

K \[ A \rightarrow (B \rightarrow A) \]
S \[ (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \]
DN \[ \neg\neg A \rightarrow A \]

Inference Rule: Modus Ponens

\[
\begin{array}{c}
A \rightarrow B \\
A \\
\hline
B
\end{array}
\]

Slide 302

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

(\( A \rightarrow ((D \rightarrow A) \rightarrow A) \)) \rightarrow \hspace{1cm} (1)

(\( (A \rightarrow (D \rightarrow A)) \rightarrow (A \rightarrow A) \)) \hspace{1cm} \text{by S}

(\( A \rightarrow ((D \rightarrow A) \rightarrow A) \)) \hspace{1cm} \text{by K} \hspace{1cm} (2)

(\( A \rightarrow (D \rightarrow A) \)) \rightarrow (A \rightarrow A) \hspace{1cm} \text{by MP, (1), (2)} \hspace{1cm} (3)

A \rightarrow (D \rightarrow A) \hspace{1cm} \text{by K} \hspace{1cm} (4)

A \rightarrow A \hspace{1cm} \text{by MP, (3), (4)} \hspace{1cm} (5)

Lawrence C. Paulson  
University of Cambridge
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 $\land$ shows how to deduce $A \land B$:

\[
\begin{array}{c}
A \\
B
\end{array} \quad \frac{}{A \land B}
\]

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

\[
\begin{array}{c}
A \land B
\end{array} \quad \frac{}{A} \quad \frac{}{B}
\]
The Sequent Calculus

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

if $A_1 \land \ldots \land A_m$ then $B_1 \lor \ldots \lor B_n$

$A_1, \ldots, A_m$ are assumptions; $B_1, \ldots, B_n$ are goals

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

The sequent $A, \Gamma \Rightarrow A, \Delta$ is trivially true (basic sequent).

Sequent Calculus Rules

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

\[
\frac{\Gamma \Rightarrow \Delta, A}{-A, \Gamma \Rightarrow \Delta} \quad \frac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, -A} \quad \text{(-l)}
\]

\[
\frac{A, B, \Gamma \Rightarrow \Delta}{A \land B, \Gamma \Rightarrow \Delta} \quad \frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, A \land B} \quad \text{(-r)}
\]

Lawrence C. Paulson University of Cambridge
More Sequent Calculus Rules

\[
\begin{align*}
A, \Gamma \Rightarrow \Delta & \quad B, \Gamma \Rightarrow \Delta \\ & \quad \frac{A \lor B, \Gamma \Rightarrow \Delta}{\text{\textcopyright{\textmu{l}}}}\quad \frac{\Gamma \Rightarrow \Delta, A, B}{\text{\textcopyright{\textmu{r}}}} \\
\Gamma \Rightarrow \Delta, A & \quad B, \Gamma \Rightarrow \Delta \\ & \quad \frac{A \Rightarrow B, \Gamma \Rightarrow \Delta}{\text{\textcopyright{\textl{l}}}}\quad \frac{A, \Gamma \Rightarrow \Delta, B}{\text{\textcopyright{\textl{r}}}} \\
\end{align*}
\]

Easy Sequent Calculus Proofs

\[
\begin{align*}
A, B \Rightarrow A & \quad \frac{\text{\textcopyright{\textl{l}}}}{A \land B \Rightarrow A} \\
& \quad \frac{\Rightarrow (A \land B) \Rightarrow A}{\text{\textcopyright{\textl{r}}}} \quad (\Rightarrow) \\
\end{align*}
\]

\[
\begin{align*}
A, B \Rightarrow B, \bar{A} & \quad \frac{\text{\textcopyright{\textl{l}}}}{A \Rightarrow B, B \Rightarrow A} \\
& \quad \frac{\Rightarrow (A \Rightarrow B) \lor (B \Rightarrow A)}{\text{\textcopyright{\textl{r}}}} \quad (\Rightarrow) \\
\end{align*}
\]
Part of a Distributive Law

\[
\begin{align*}
A & \Rightarrow A, B \\
B & \Rightarrow A, B \\
B \land C & \Rightarrow A, B \\
A \lor (B \land C) & \Rightarrow A, B \\
A \lor (B \land C) & \Rightarrow A \lor B \\
A \lor (B \land C) & \Rightarrow (A \lor B) \land (A \lor C) \\
\end{align*}
\]

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

A Failed Proof

\[
\begin{align*}
A & \Rightarrow B, C \\
B & \Rightarrow B, C \\
A \lor B & \Rightarrow B, C \\
A \lor B & \Rightarrow B \lor C \\
\Rightarrow (A \lor B) \Rightarrow (B \lor C) \\
\end{align*}
\]

\( A \Rightarrow t, B \Rightarrow f, C \Rightarrow f \) falsifies unproved sequent!
BDDs: Binary Decision Diagrams

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

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

Detects if a formula is tautologous (t) or inconsistent (f).
Exhibits models if the formula is satisfiable.
Excellent for verifying digital circuits, with many other applications.

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

- Lawrence C. Paulson
- University of Cambridge
Converting a Decision Diagram to a BDD

No duplicates  No redundant tests

Building BDDs Efficiently

Do not construct full tree!
Do not expand $\to$, $\leftrightarrow$, $\oplus$ (exclusive OR) to other connectives.
Treat $\neg Z$ as $Z \to f$ or $Z \oplus t$.
Recursively convert operands to BDDs.
Combine operand BDDs, respecting the ordering and sharing.
Delete redundant variable tests.

Lawrence C. Paulson  University of Cambridge
**Canonical Form Algorithm**

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

*Trivial if either operand is $t$ or $f$.*

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

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

---

**Canonical Forms of Other Connectives**

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

Some cases, like $Z \rightarrow t$, 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 $Z = t$ then return $f$, and if $Z = f$ then return $t$.

In effect we copy the BDD but swap $t$ and $f$ at the leaves.
Canonical Form (that is, BDD) of $P \lor Q$

Slide 407

Canonical Form of $P \lor Q \rightarrow Q \lor R$

Slide 408

Lawrence C. Paulson
University of Cambridge
**Optimisations Based On Hash Tables**

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

- If $X \leftrightarrow Y$, then the addresses of $X$ and $Y$ are equal.
- Can see if $H(P, X, Y)$ is redundant by checking if $X = Y$.
- Can quickly simplify special cases like $X \land X$.

*Never convert $X \land Y$ twice*, but keep a table of known canonical forms.

---

**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.

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).

---

Lawrence C. Paulson  
University of Cambridge
Outline of First-Order Logic

Reasons about functions and relations over a set of individuals:
\[
\text{father}(\text{father}(x)) = \text{father}(\text{father}(y))
\]
\[
\text{cousin}(x, y)
\]

Reasons about all and some individuals:
- All men are mortal
- Socrates is a man
- 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
\[
f(t_1, \ldots, t_n)
\]

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.

Lawrence C. Paulson

University of Cambridge
**Relation Symbols; Formulae**

Each relation symbol stands for an n-place relation.

*Equality* is the 2-place relation symbol $\equiv$.

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

A formula is built up from atomic formulæ using $\neg$, $\land$, $\lor$, and so forth. Later, we can add quantifiers.

---

**The Power of Quantifier-Free FOL**

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

It is easy to equivalence of mathematical functions:

\[
\begin{align*}
    p(z, 0) &= 1 \\
    p(z, n + 1) &= p(z, n) \times z \\
    q(z, 1) &= z \\
    q(z, 2 \times n) &= q(z \times z, n) \\
    q(z, 2 \times n + 1) &= q(z \times z, n) \times z
\end{align*}
\]

The prover ACL2 uses this logic and has been used in major hardware proofs.

Lawrence C. Paulson
University of Cambridge
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 \( \forall x \forall y \forall z A \)
\( \forall z . A \land B \) 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:*

\( \forall x [\text{man}(x) \rightarrow \text{mortal}(x)] \)

*All mothers are female:*

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

*There exists a unique \( x \) such that \( A \), sometimes written \( \exists! x A \):

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

Lawrence C. Paulson

University of Cambridge
How do we interpret \( \text{mortal}(\text{Socrates}) \)?

Take an interpretation \( \mathcal{I} = (D, I) \) of our first-order language.

- \( D \) is a non-empty set, called the \textit{domain} or \textit{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 \( I[P] \subseteq D^n \)

How do we interpret \( \text{cousin}(\text{Charles}, y) \)?

A \textit{valuation} supplies the values of free variables.

It is a function \( V : \text{variables} \rightarrow D \).

\( \mathcal{I}_V[t] \) extends \( V \) to a term \( t \) by the obvious recursion:

- \( \mathcal{I}_V[x] \overset{\text{def}}{=} V(x) \) if \( x \) is a variable
- \( \mathcal{I}_V[c] \overset{\text{def}}{=} I[c] \)
- \( \mathcal{I}_V[f(t_1, \ldots, t_n)] \overset{\text{def}}{=} I[f](\mathcal{I}_V[t_1], \ldots, \mathcal{I}_V[t_n]) \)
For interpretation $I$ and valuation $V$, define $|=_{I,V}$ by recursion.

$|=_{I,V} P(t)$ if $I_V[t] \in I[P]$ holds

$|=_{I,V} t = u$ if $I_V[t]$ equals $I_V[u]$.

$|=_{I,V} A \land B$ if $|=_{I,V} A$ and $|=_{I,V} B$.

$|=_{I,V} \exists x A$ if $|=_{I,V(m/x)} A$ holds for some $m \in D$.

Finally, we define

$|=_I A$ if $|=_{I,V} A$ holds for all $V$.

Formula $A$ is satisfiable if $|=_I A$ for some $I$. 

Lawrence C. Paulson
University of Cambridge
**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.

May **rename** bound variables:

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

---

**Substitution for Free Variables**

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

- $(B \land C)[t/x]$ is $B[t/x] \land C[t/x]$
- $(\forall x B)[t/x]$ is $\forall x B$
- $(\forall y B)[t/x]$ is $\forall y B[t/x]$  $(x \neq y)$
- $(P(u))[t/x]$ is $P[u[t/x]]$

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

$(\forall y (x = y))[y/x]$ is **not equivalent to** $\forall y (y = y)$
Some Equivalences for Quantifiers

\[ \neg (\forall x \ A) \simeq \exists x \neg A \]
\[ \forall x \ A \simeq \forall x \ A \land A[t/x] \]
\[ (\forall x \ A) \land (\forall x \ B) \simeq \forall x (A \land B) \]

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 \(B\).

\[ (\forall x \ A) \land B \simeq \forall x (A \land B) \]
\[ (\forall x \ A) \lor B \simeq \forall x (A \lor B) \]
\[ (\forall x \ A) \rightarrow B \simeq \exists x (A \rightarrow B) \]

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

Lawrence C. Paulson
University of Cambridge
### Reasoning by Equivalences

\[
\exists x (x = a \land P(x)) \simeq \exists x (x = a \land P(a)) \\
\simeq \exists x (x = a) \land P(a) \\
\simeq P(a)
\]

\[
\exists z (P(z) \to P(a) \land P(b)) \\
\simeq \forall z P(z) \to P(a) \land P(b) \\
\simeq \forall z P(z) \land P(a) \land P(b) \to P(a) \land P(b) \\
\simeq \top
\]

### Sequent Calculus Rules for \( \forall \)

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

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

\[
\frac{P(y) \Rightarrow P(y)}{P(y) \Rightarrow \forall y P(y)} \quad (\forall r) \quad \text{THIS IS NONSENSE!}
\]

Lawrence C. Paulson  
University of Cambridge
**A Simple Example of the \( \forall \) Rules**

\[
\begin{align*}
P(f(y)) & \implies P(f(y)) \\
\forall x P(x) & \implies P(f(y)) \\
\forall x P(x) & \implies \forall y P(f(y))
\end{align*}
\]

**A Not-So-Simple Example of the \( \forall \) Rules**

\[
\begin{align*}
P & \implies Q(y), \overline{P} & \quad P, Q(y) & \implies Q(y) \quad (\rightarrow l) \\
P, P \implies Q(y) & \implies Q(y) \quad (\rightarrow l) \\
\overline{P}, \forall x (P \implies Q(x)) & \implies Q(y) \quad (\forall l) \\
\overline{P}, \forall x (P \implies Q(x)) & \implies \forall y Q(y) \quad (\forall r) \\
\forall x (P \implies Q(x)) & \implies P \implies \forall y Q(y) \quad (\rightarrow r)
\end{align*}
\]

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

Lawrence C. Paulson  
University of Cambridge
Sequent Calculus Rules for $\exists$

Slide 609

$$\Gamma \Rightarrow \Delta$$

$$\exists x \ A, \Gamma \Rightarrow \Delta$$

$$(\exists l)$$

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:

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

Part of the $\exists$ Distributive Law

Slide 610

$$P(x) \Rightarrow P(x), Q(x)$$

$$(\lor r)$$

$$P(x) \Rightarrow P(x) \lor Q(x)$$

$$(\exists r)$$

$$P(x) \Rightarrow \exists y (P(y) \lor Q(y))$$

$$(\exists l)$$

$$\exists x P(x) \Rightarrow \exists y (P(y) \lor Q(y))$$

$$(\exists l)$$

$$\exists x P(x) \lor \exists x Q(x) \Rightarrow \exists y (P(y) \lor Q(y))$$

$$(\lor l)$$

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

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

Lawrence C. Paulson  
University of Cambridge
A Failed Proof

\[ P(x), Q(y) \Rightarrow P(x) \land Q(x) \]
\[ P(x), Q(y) \Rightarrow \exists z \ (P(z) \land Q(z)) \quad \{\exists r\} \]
\[ P(x), \exists x Q(x) \Rightarrow \exists z \ (P(z) \land Q(z)) \quad \{\exists l\} \]
\[ \exists x P(x), \exists x Q(x) \Rightarrow \exists z \ (P(z) \land Q(z)) \quad \{\exists l\} \]
\[ \exists x P(x) \land \exists x Q(x) \Rightarrow \exists z \ (P(z) \land Q(z)) \quad \{\land l\} \]

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: \[ K_1, \ldots, K_m \rightarrow L_1, \ldots, L_n \]
\[ L_1, \ldots, L_n \leftarrow K_1, \ldots, K_m \]

Empty clause:

\[ \Box \]

Empty clause is equivalent to \( f \), meaning CONTRADICTION!

Outline of Clause Form Methods

To prove \( A \), obtain a contradiction from \( \neg A \):

1. Translate \( \neg A \) into CNF as \( A_1 \land \cdots \land A_m \)
2. This is the set of clauses \( A_1, \ldots, A_m \)
3. Transform the clause set, preserving consistency

Deducing the empty clause refutes \( \neg A \).

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

The basis for SAT SOLVERS and RESOLUTION PROVERS.

Lawrence C. Paulson University of Cambridge
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

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

---

Davis-Putnam on a Non-Tautology

Consider \( P \lor Q \rightarrow Q \lor R \)

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

\[
\begin{align*}
\{P, Q\} & \quad \{\neg Q\} \quad \{\neg R\} \quad \text{initial clauses} \\
\{P\} & \quad \{\neg R\} \quad \text{unit } \neg Q \\
\{\neg R\} & \quad \text{unit } P \quad \text{(also pure)} \\
\{\neg R\} & \quad \text{unit } \neg R \quad \text{(also pure)}
\end{align*}
\]

Clauses satisfiable by \( P \mapsto t, \: Q \mapsto f, \: R \mapsto f \)
Example of a Case Split on $P$

\[
\begin{align*}
\{&\neg Q, R\} \quad \{&\neg R, P\} \quad \{&\neg R, Q\} \quad \{&\neg P, Q, R\} \quad \{P, Q\} \quad \{&\neg P, \neg Q\} \\
\{&\neg Q, R\} \quad \{&\neg R, Q\} \quad \{&Q, R\} \quad \{&\neg Q\} & \text{if } P \text{ is true} \\
\{&\neg R\} \quad \{&R\} & \text{unit } \neg Q \\
\Box & \quad \text{unit } R \\
\{&\neg Q, R\} \quad \{&\neg R\} \quad \{&\neg R, Q\} \quad \{&Q\} & \text{if } P \text{ is false} \\
\{&\neg Q\} \quad \{&Q\} & \text{unit } \neg R \\
\Box & \quad \text{unit } \neg Q
\end{align*}
\]

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).
- Typical approach: approximate the problem with a finite model; encode it using Boolean logic; supply to a SAT solver.
The Resolution Rule

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

In set notation,

\[
\begin{array}{c}
\{B, A_1, \ldots, A_m\} \\
\{\neg B, C_1, \ldots, C_n\}
\end{array}
\equiv
\begin{array}{c}
\{A_1, \ldots, A_m, C_1, \ldots, C_n\}
\end{array}
\]

Some special cases:

\[
\begin{array}{c}
\{B\} \\
\{\neg B, C_1, \ldots, C_n\}
\end{array}
\equiv
\begin{array}{c}
\{C_1, \ldots, C_n\}
\end{array}
\]

\[
\begin{array}{c}
\{B\} \\
\{\neg B\}
\end{array}
\equiv
\square
\]

Simple Example: Proving $P \land Q \rightarrow Q \land P$

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

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

Clauses:

\[
\begin{array}{c}
\{P\} \\
\{Q\} \\
\{\neg Q, \neg P\}
\end{array}
\]

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

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

Refute $\neg[(P \lor Q) \land (P \lor R) \rightarrow P \lor (Q \land R)]$

From $(P \lor Q) \land (P \lor R)$, get clauses $\{P, Q\}$ and $\{P, R\}$.

From $\neg[P \lor (Q \land R)]$ 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 $\Box$, 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.
Refinements of Resolution

Subsumption: deleting redundant clauses
Preprocessing: removing tautologies, symmetries...
Indexing: elaborate data structures for speed
Ordered resolution: restrictions to focus the search
Weighting: giving priority to the smallest clauses
Set of Support: working on the goal, not the axioms
Reducing FOL to Propositional Logic

**Prenex:** Move quantifiers to the front

**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

---

**Prenex Normal Form**

Convert to Negation Normal Form using additionally

\[
\neg (\forall x \ A) \simeq \exists x \neg A \\
\neg (\exists x \ A) \simeq \forall x \neg A
\]

Move quantifiers to the front using (provided \(x\) is not free in \(B\))

\[
(\forall x \ A) \land B \simeq \forall x (A \land B) \\
(\forall x \ A) \lor B \simeq \forall x (A \lor B)
\]

and the similar rules for \(\exists\)
Skolemization, or Getting Rid of $\exists$

Start with a formula of the form

$$\forall x_1 \forall x_2 \cdots \forall x_k \exists y \ A$$

(Can have $k = 0$).

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

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

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

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

$\forall x \ \exists y \ [P(x) \land \neg P(y)]$ pulling $\exists$ out

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

$\{P(x)\} \ \{\neg P(f(x))\}$ Final clauses
**Correctness of Skolemization**

The formula \( \forall x \exists y A \) is consistent

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

\[\iff\] for all \( x \in \mathcal{D} \) there is some \( y \in \mathcal{D} \) such that \( A \) holds

\[\iff\] some function \( \hat{f} \) in \( \mathcal{D} \to \mathcal{D} \) yields suitable values of \( y \)

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

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

Don’t panic if you can’t follow this reasoning!

---

**The Herbrand Universe for a set of clauses \( S \)**

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

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

\[ H \overset{\text{def}}{=} \bigcup_{i \geq 0} H_i \quad \text{Herbrand Universe} \]

\( H \) contains the terms expressible using the function symbols of \( S \).

\( H_i \) contains just the terms with at most \( i \) nested function applications.

Lawrence C. Paulson  
University of Cambridge
Herbrand Interpretations for a set of clauses $S$

$$\text{HB} \overset{\text{def}}{=} \{ P(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in H \quad \text{and} \quad P \text{ is an } n\text{-place predicate symbol in } S \}$$

$\text{HB}$ contains all applications of predicates to elements of $H$.

*Each subset of $\text{HB}$ defines the cases where the predicates are true.*

A Herbrand model will interpret the predicates by some subset of $\text{HB}$.

It will interpret function symbols by term-forming operations:

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

---

Example of an Herbrand Model

\[
\begin{aligned}
\neg \text{even}(1) \\
\text{even}(2) \\
\text{even}(X \cdot Y) \leftarrow \text{even}(X), \text{even}(Y)
\end{aligned}
\]

\[
\begin{aligned}
\text{clauses}
\end{aligned}
\]

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

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

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

(for model where $\cdot$ means product; could instead use sum!)

Lawrence C. Paulson

University of Cambridge
A Key Fact about Herbrand Interpretations

Let $S$ be a set of clauses.

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

- Holds because some Herbrand model mimics every ‘real’ model
- We must consider only a small class of models
- Herbrand models are syntactic, easily processed by computer

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 $\{P(x), \neg P(f(y))\}$, 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**
- Tools for satisfying multiple **constraints**
- Polymorphic type-checking (**ML** and other functional languages)

It's an intuitive generalization of pattern-matching.

---

Substitutions: A Mathematical Treatment

A substitution is a finite set of replacements

\[ \theta = \left\{ t_1/x_1, \ldots, t_k/x_k \right\} \]

where \( x_1, \ldots, x_k \) are distinct variables and \( t_1 \neq x_i \).

- \( f(t, u)\theta = f(t\theta, u\theta) \) (substitution in terms)
- \( P(t, u)\theta = P(t\theta, u\theta) \) (in literals)
- \( \{L_1, \ldots, L_m\}\theta = \{L_1\theta, \ldots, L_m\theta\} \) (in clauses)

---

Lawrence C. Paulson       University of Cambridge
Composing Substitutions

Composition of φ and θ, written φ ◦ θ, satisfies for all terms t

\[ t(φ ◦ θ) = (tφ)θ \]

It is defined by (for all relevant x)

\[ φ ◦ θ \overset{\text{def}}{=} [(xφ)θ / x, \ldots] \]

Consequences include θ ◦ [] = θ, and associativity:

\[ (φ ◦ θ) ◦ σ = φ ◦ (θ ◦ σ) \]

Most General Unifiers

θ is a unifier of terms t and u if tθ = uθ.

θ is more general than φ if φ = θ ◦ σ for some substitution σ.

θ is most general if it is more general than every other unifier.

If θ unifies t and u then so does θ ◦ σ:

\[ t(θ ◦ σ) = tθσ = uθσ = u(θ ◦ σ) \]

A most general unifier of \( f(a, x) \) and \( f(y, g(z)) \) is \([a/y, g(z)/x]\).

The common instance is \( f(a, g(z)) \).

Lawrence C. Paulson

University of Cambridge
The Unification Algorithm

Represents terms by binary trees.

Each term is a Variable $x$, $y$, ..., Constant $a$, $b$, ..., or Pair $(t, t')$

Sketch of the Algorithm.

- Constants do not unify with different Constants.
- Constants do not unify with Pairs.
- Variable $x$ and term $t$: unifier is $[t/x]$, unless $x$ occurs in $t$

Cannot unify $f(x)$ with $x$!

The Unification Algorithm: The Case of Two Pairs

$\theta \circ \theta'$ unifies $(t, t')$ with $(u, u')$

If $\theta$ unifies $t$ with $u$ and $\theta'$ unifies $t' \theta$ with $u' \theta$.

We unify the left sides, then the right sides.

In an implementation, substitutions are formed by updating pointers.

Composition happens automatically as more pointers are updated.

Lawrence C. Paulson
University of Cambridge
It's easy to check that $\theta \circ \theta'$ unifies $(t, t')$ with $(u, u')$:

$$(t, t')|\theta \circ \theta'| = (t, t')\theta \theta'$$

$$= (t\theta', t'\theta')$$

$$= (u\theta', u'\theta')$$

$$= (u, u')\theta \theta'$$

$$= (u, u')(\theta \circ \theta')$$

$\theta \circ \theta'$ is even a most general unifier, if $\theta$ and $\theta'$ are!
**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(\alpha, a)\} and \{\neg R(b, y)\}.
The literals \(R(x, \alpha)\) and \(R(b, y)\) have unifier \([b/x, \alpha/y]\).
We have the contradiction \(R(b, \alpha)\) and \(\neg R(b, s)\).

**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(\alpha, 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!**
Variations on Unification

Efficient unification algorithms: near-linear time

Indexing & Discrimination networks: fast retrieval of a unifiable term

Associative/commutative unification

- Example: unify \( a + (y + c) \) with \( (c + x) + b \), get \([a/x, b/y]\)
- Algorithm is very complicated
- The number of unifiers can be exponential

Unification in many other theories (often undecidable!)

Lawrence C. Paulson University of Cambridge
The Binary Resolution Rule

\[ \begin{array}{c}
\{B, A_1, \ldots, A_m\} \\
\{\neg D, C_1, \ldots, C_n\}
\end{array} \xrightarrow{provided \ B \sigma = D \sigma} \begin{array}{c}
\{A_1, \ldots, A_m, C_1, \ldots, C_n\} \sigma
\end{array} \]

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

\[ \{P(x)\} \quad \text{and} \quad \{\neg P(g(x))\} \]

to rename \(x\) in one of the clauses before attempting unification.

Always use a most general unifier (MGU).

The Factoring Rule

This inference collapses unifiable literals in one clause:

\[ \begin{array}{c}
\{B_1, B_k, A_1, \ldots, A_m\} \\
\{B_1, A_1, \ldots, A_m\} \sigma
\end{array} \xrightarrow{provided \ B_1 \sigma = \cdots = B_k \sigma} \]

Example: Prove \(\forall x \exists y \neg (P(y, x) \iff \neg P(y, y))\)

The clauses are

\[ \{\neg P(y, a), \neg P(y, y)\} \quad \{P(y, y), P(y, a)\} \]

Factoring yields

\[ \{\neg P(y, a)\} \quad \{P(y, a)\} \]

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)\}, \{-P, \neg Q(x)\}, \{-P, Q(a)\}

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

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

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

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

Resolve \{P\} with \{-P\} getting \Box

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**

\[
\begin{align*}
\frac{B[t'], A_1, \ldots, A_m}{B[u], A_1, \ldots, A_m, C_1, \ldots, C_n} & \quad (\text{if } t \sigma = t' \sigma)
\end{align*}
\]

Lawrence C. Paulson 
University of Cambridge
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)

Lawrence C. Paulson

University of Cambridge
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
* :- parent(elizabeth, Y5), parent(Y5, Y).
Y = beatrice
* = 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 \( \{ A_1, \ldots, A_m \} \) 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

Saturation (that is, resolution): E, Gandalf, SPASS, Vampire, …

Higher-Order Logic: TPS, LEO

Model Elimination: Prolog Technology Theorem Prover, SETHEO

Parallel ME: PARTHENON, PARTHEO

Tableau (sequent) based: LeanTAP, 3TAP, …
Modal Operators

\( W \): set of possible worlds (machine states, future times, \ldots )

\( R \): accessibility relation between worlds

\( (W, R) \) is called a modal frame

\[ \square A \text{ means } A \text{ is necessarily true} \]

\[ \Diamond A \text{ means } A \text{ is possibly true} \]

\[ \neg \Diamond A \equiv \Box \neg A \]

\( A \) 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 \( W \)

\[ w \Vdash A \text{ means } A \text{ is true in world } w \]

\[ w \Vdash P \iff w \in I(P) \]

\[ w \Vdash A \land B \iff w \Vdash A \text{ and } w \Vdash B \]

\[ w \Vdash \square A \iff \forall v \text{ such that } R(w,v) \]

\[ w \Vdash \Diamond A \iff \exists v \text{ such that } R(w,v) \]

Lawrence C. Paulson

University of Cambridge
Truth and Validity in Modal Logic

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

\[ w \vDash A \quad \text{means } A \text{ is true in world } w \]

\[ \models_{W,R,I} A \quad \text{means } w \vDash A \text{ for all } w \text{ in } W \]

\[ \models_{W,R} A \quad \text{means } w \vDash A \text{ for all } w \text{ and all } I \]

\[ \models A \quad \text{means } \models_{W,R} A \text{ for all frames; } A \text{ is universally valid} \]

\[ \ldots \text{but typically we constrain } R \text{ to be, say, transitive} \]

All 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

\[ \Diamond A \overset{\text{def}}{=} \neg \Box \neg A \]
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 $T$
- $4$: $A \rightarrow \Box \Box A$ (transitive) logic $S4$
- $B$: $A \rightarrow \Box \Diamond A$ (symmetric) logic $S5$

And countless others!

We shall mainly look at $S4$

---

Extra Sequent Calculus Rules for $S4$

- $\frac{A, \Gamma \Rightarrow \Delta}{\Box A, \Gamma \Rightarrow \Delta}$ \hspace{1cm} $\frac{\Gamma^* \Rightarrow \Delta^*, A}{\Gamma \Rightarrow \Delta, \Box A}$ ($\Box_l$)
- $\frac{A, \Gamma^* \Rightarrow \Delta^*}{\Diamond A, \Gamma \Rightarrow \Delta}$ \hspace{1cm} $\frac{\Gamma \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \Diamond A}$ ($\Diamond_l$)

- $\Gamma^* \overset{\text{def}}{=} \{ \Box B \mid \Box B \in \Gamma \}$ Erase non-$\Box$ assumptions
- $\Delta^* \overset{\text{def}}{=} \{ \Diamond B \mid \Diamond B \in \Delta \}$ Erase non-$\Diamond$ goals!
A Proof of the Distribution Axiom

\[\begin{align*}
A \Rightarrow B, A & \quad B, A \Rightarrow B \\
A \Rightarrow B, A & \Rightarrow B \quad (\sim_1) \\
A \Rightarrow B, \Box A & \Rightarrow B \quad (\Box_1) \\
\Box(A \Rightarrow B), \Box A & \Rightarrow B \quad (\Box_1) \\
\Box(A \Rightarrow B), \Box A & \Rightarrow \Box B \quad (\Box r) \\
\end{align*}\]

And thus \(\Box(A \Rightarrow B) \Rightarrow (\Box A \Rightarrow \Box B)\)

Must apply \((\Box r)\) first!

Part of an Operator String Equivalence

\[\begin{align*}
\Diamond A \Rightarrow \Diamond A & \quad (\Diamond 1) \\
\Box \Diamond A \Rightarrow \Diamond A & \quad (\Box \Diamond 1) \\
\Diamond \Diamond \Diamond A \Rightarrow \Diamond A & \quad (\Diamond \Diamond 1) \\
\Box \Diamond \Diamond A \Rightarrow \Box \Diamond A & \quad (\Box \Diamond r) \\
\end{align*}\]

In fact, \(\Box \Diamond \Diamond A \simeq \Diamond \Diamond A\) also \(\Box \Box A \simeq \Box A\)

The S4 operator strings are \(\Box \Diamond \Box \Box \Diamond \Diamond \Diamond \Diamond \Diamond \Box \)

Lawrence C. Paulson University of Cambridge
Two Failed Proofs

\[
\begin{align*}
& \Rightarrow A \\
& \Rightarrow \Diamond A \quad (\lor) \\
& A \Rightarrow \Box \Diamond A \quad (\Box \lor)
\end{align*}
\]

\[
\begin{align*}
& B \Rightarrow A \wedge B \\
& \therefore B \Rightarrow \Diamond (A \wedge B) \quad (\Box \lor) \\
& \therefore \Diamond A, \Diamond B \Rightarrow \Diamond (A \wedge B) \quad (\Box \lor)
\end{align*}
\]

Can extract a countermodel from the proof attempt
Simplifying the Sequent Calculus

7 connectives (or 9 for modal logic):
\[ \neg \land \lor \rightarrow \leftrightarrow \forall \exists (\Box \Diamond) \]

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

Idea! Work in Negation Normal Form

Fewer connectives: \[ \land \lor \forall \exists (\Box \Diamond) \]

Sequents need one side only!

Simplified Calculus: Left-Only

\[
\begin{align*}
\neg A, A, \Gamma & \Rightarrow \quad (\text{basic}) \quad \neg A, \Gamma \Rightarrow A, \Gamma \Rightarrow \\
A, B, \Gamma & \Rightarrow \quad (\land I) \quad A, \Gamma \Rightarrow B, \Gamma \Rightarrow \\
\Gamma & \Rightarrow \quad (\text{cut}) \\
A \land B, \Gamma & \Rightarrow \quad (\lor I) \quad A \lor B, \Gamma \Rightarrow \\
A[t/x], \Gamma & \Rightarrow \quad (\forall I) \quad A, \Gamma \Rightarrow \exists x A, \Gamma \Rightarrow \\
\forall x A, \Gamma & \Rightarrow \quad (\exists I)
\end{align*}
\]

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

Lawrence C. Paulson 
University of Cambridge
**Left-Only Sequent Rules for S4**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A, \Gamma \Rightarrow$</td>
<td>$\Box A, \Gamma \Rightarrow$ (O1)</td>
</tr>
<tr>
<td>$A, \Gamma^* \Rightarrow$</td>
<td>$\Diamond A, \Gamma \Rightarrow$ (O1)</td>
</tr>
</tbody>
</table>

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

---

**Proving $\forall x (P \rightarrow Q(x)) \Rightarrow P \rightarrow \forall y Q(y)$**

Move the right-side formula to the left and convert to NNF:

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

\[
\begin{align*}
P, \neg Q(y), \neg P & \Rightarrow \quad P, \neg Q(y), \neg P \lor Q(y) \Rightarrow \quad [\lor r] \\
P, \neg Q(y), \neg P \lor Q(y) & \Rightarrow \quad [\lor l] \\
P, \neg Q(y), \forall x (\neg P \lor Q(x)) & \Rightarrow \quad [\exists l] \\
P, \exists y \neg Q(y), \forall x (\neg P \lor Q(x)) & \Rightarrow \quad [\exists l] \\
\end{align*}
\]

Lawrence C. Paulson  
University of Cambridge
Adding Unification

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

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

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 $(\exists l)$? Skolemize!

Skolemization from NNF

Don’t pull quantifiers out! Skolemize.

$$(\forall y \exists z Q(y, z)) \land \exists x P(x) \Rightarrow (\forall y Q(y, f(y))) \land P(a)$$

It’s better to push quantifiers in (called miniscoping).

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

1. Negate; convert to NNF: $\forall x \exists y [P(x) \land \neg P(y)]$
2. Push in the $\exists y$: $\forall x [P(x) \land \exists y \neg P(y)]$
3. Push in the $\forall x$: $(\forall x P(x)) \land (\exists y \neg P(y))$
4. Skolemize: $\forall x P(x) \land \neg P(a)$
A Proof of $\exists x \forall y [P(x) \rightarrow P(y)]$

\[
\begin{align*}
y \mapsto f(z) \\
P(y), \neg P(f(y)), P(z), \neg P(f(z)) \Rightarrow & \quad \text{(basic)} \\
P(y), \neg P(f(y)), P(z) \land \neg P(f(z)) \Rightarrow & \quad \text{\scriptsize$(\land I)$} \\
P(y), \neg P(f(y)), \forall x [P(x) \land \neg P(f(x))] \Rightarrow & \quad \text{\scriptsize$(\forall I)$} \\
P(y) \land \neg P(f(y)), \forall x [P(x) \land \neg P(f(x))] \Rightarrow & \quad \text{\scriptsize$(\forall I)$} \\
\forall x [P(x) \land \neg P(f(x))] \Rightarrow & \quad \text{Unification chooses the term for $(\forall I)$}
\end{align*}
\]

A Failed Proof

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

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

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

\[
\begin{align*}
y \mapsto a \\
\neg P(a), \neg Q(b), P(y) \Rightarrow & \quad \text{y \mapsto b??} \\
\neg P(a), \neg Q(b), Q(y) \Rightarrow & \quad \text{\scriptsize$(\lor I)$} \\
\neg P(a), \neg Q(b), P(y) \lor Q(y) \Rightarrow & \quad \text{\scriptsize$(\lor I)$} \\
\neg P(a), \neg Q(b), \forall x [P(x) \lor Q(x)] \Rightarrow & \quad \text{\scriptsize$(\forall I)$} \\
\neg P(a) \land \neg Q(b), \forall x [P(x) \lor Q(x)] \Rightarrow & \quad \text{\scriptsize$(\land I)$}
\end{align*}
\]

Lawrence C. Paulson
University of Cambridge
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).