Logic and Proof

Computer Science Tripos Part IB
Michaelmas Term

Lawrence C Paulson
Computer Laboratory
University of Cambridge

lcp@cl.cam.ac.uk

Copyright © 1999 by Lawrence C. Paulson

Contents

1 Introduction and Learning Guide **1**

2 Propositional Logic **2**

3 Proof Systems for Propositional Logic **11**

4 Ordered Binary Decision Diagrams **18**

5 First-order Logic **20**

6 Formal Reasoning in First-Order Logic **27**

7 Clause Methods for Propositional Logic **32**

8 Skolem Functions and Herbrand's Theorem **40**

9 Unification **47**

10 Applications of Unification **56**

11 Modal Logics **63**

12 Tableaux-Based Methods **68**

1 Introduction and Learning Guide

This course gives a brief introduction to logic, with emphasis on the resolution method of theorem-proving. Formal logic is used for specifying and verifying computer systems and (sometimes) for representing knowledge in Artificial Intelligence programs.

The course describes unification, which underlies the resolution method and Prolog. Unification has other applications, such as type checking in ML. This material is clearly relevant for the Prolog programming course. Its treatment of logic should be helpful for understanding courses such as Proving Programs. Try to avoid getting bogged down in the details of how the various proof methods work, since you must also acquire an intuitive feel for logical reasoning.

The course follows the classic text

> C.-L. Chang and R. C.-T. Lee, *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, 1973)

For more modern treatments, see

> Antony Galton, *Logic for Information Technology* (Wiley, 1990)
>
> Steve Reeves and Michael Clarke, *Logic for Computer Science* (Addison-Wesley, 1990)

Also relevant is

> Melvin Fitting, *First-Order Logic and Automated Theorem Proving* (Springer, 1990)

Many of these books are out of print, but there will be some copies in College libraries.

There are numerous exercises in these notes, and they are suitable for supervision purposes. Old examination questions for *Foundations of Logic Programming* (the former name of this course) are still relevant:

- 98 paper 5 question 10: OBDDs, sequent calculus, etc. (Lect. 4)
- 98 paper 6 question 10: modal logic (Lect. 11); resolution (Lect. 10)
- 97 paper 5 question 10: first-order logic (Lect. 5)
- 97 paper 6 question 10: sequent rules for quantifiers (Lect. 6)
- 96 paper 5 question 10: sequent calculus (Lect. 3, 6, 11)
- 96 paper 6 question 10: Davis-Putnam versus Resolution (Lect. 10)

- 95 paper 5 question 9: OBBDs (Lect. 4)
- 95 paper 6 question 9: outline logics; sequent calculus (Lect. 3, 6, 11)
- 94 paper 5 question 9: Resolution versus Prolog (Lect. 10)
- 94 paper 6 question 9: Herbrand models (Lect. 8)
- 94 paper 6 question 9: Most general unifiers and resolution (Lect. 10)
- 93 paper 3 question 3: Resolution and Prolog (Lect. 10)

Acknowledgements. Reuben Thomas pointed out numerous errors in a previous version of these notes. Ross Younger made detailed suggestions. Thanks also to Thomas Forster and Steve Payne.

2 Propositional Logic

Propositional logic deals with truth values and the logical connectives 'and,' 'or,' 'not,' etc. It has no variables of any kind and is unable to express anything but the simplest mathematical statements. It is studied because it is simple and because it is the basis of more powerful logics. Most of the concepts in propositional logic have counterparts in first-order logic. A logic comprises a *syntax*, which is a formal notation for writing assertions and a *semantics*, which gives a meaning to assertions. Its *proof theory* gives syntactic—and therefore mechanical—methods for reasoning about assertions.

2.1 Syntax of propositional logic

We take for granted a set of propositional symbols P, Q, R, $\ldots$, including the truth values $\mathbf{t}$ and $\mathbf{f}$. A formula consisting of a propositional symbol is called *atomic*.

Formulæ are constructed from atomic formulæ using the logical connectives

$\neg$ (not)
$\wedge$ (and)
$\vee$ (or)
$\rightarrow$ (implies)
$\leftrightarrow$ (if and only if)

These are listed in order of precedence; $\neg$ is highest. We shall suppress needless parentheses, writing, for example,

$$(((\neg P) \wedge Q) \vee R) \rightarrow ((\neg P) \vee Q) \quad \text{as} \quad \neg P \wedge Q \vee R \rightarrow \neg P \vee Q.$$

In the 'metalanguage' (these notes), the letters $A, B, C, \ldots$ stand for arbitrary formulæ. The letters $P, Q, R, \ldots$ stand for atomic formulæ.

Some authors use $\supset$ for the implies symbol and $\equiv$ for if-and-only-if.

2.2 Semantics

Propositional Logic is a formal language. Each formula has a meaning (or semantics) — either **t** or **f** — relative to the meaning of the propositional symbols it contains. The meaning can be calculated using the standard truth tables.

A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$
t	**t**	**f**	**t**	**t**	**t**	**t**
t	**f**	**f**	**f**	**t**	**f**	**f**
f	**t**	**t**	**f**	**t**	**t**	**f**
f	**f**	**t**	**f**	**f**	**t**	**t**

By inspecting the table, we can see that $A \rightarrow B$ is equivalent to $\neg A \vee B$ and that $A \leftrightarrow B$ is equivalent to $(A \rightarrow B) \wedge (B \rightarrow A)$. (The latter is also equivalent to $\neg(A \oplus B)$, where $\oplus$ is exclusive or.)

Note that we are using **t** and **f** in two distinct ways: as symbols on the printed page, and as the truth values themselves. In this simple case, there should be no confusion. When it comes to first-order logic, we shall spend some time on the distinction between symbols and their meanings.

We now make some definitions that will be needed throughout the course.

Definition 1 An *interpretation*, or *truth assignment*, for a set of formulæ is a function from its set of propositional symbols to {**t**, **f**}.

An interpretation *satisfies* a formula if the formula evaluates to **t** under the interpretation.

A set S of formulæ is *valid* (or a *tautology*) if every interpretation for S satisfies every formula in S.

A set S of formulæ is *satisfiable* (or *consistent*) if there is some interpretation for S that satisfies every formula in S.

A set S of formulæ is *unsatisfiable* (or *inconsistent*) if it is not satisfiable.

A set S of formulæ *entails* A if every interpretation that satisfies all elements of S, also satisfies A. Write $S \models A$.

Formulæ A and B are *equivalent*, $A \simeq B$, provided $A \models B$ and $B \models A$.

It is usual to write $A \models B$ instead of $\{A\} \models B$. We may similarly identify a one-element set with a formula in the other definitions.

Note that $\models$ and $\simeq$ are not logical connectives but relations between formulæ. They belong not to the logic but to the metalanguage: they are symbols we use to discuss the logic. They therefore have lower precedence than the logical connectives. No parentheses are needed in $A \wedge A \simeq A$ because the only possible reading is $(A \wedge A) \simeq A$. We may not write $A \wedge (A \simeq A)$ because $A \simeq A$ is not a formula.

In propositional logic, a valid formula is also called a *tautology*. Here are some examples of these definitions.

- The formulæ $A \to A$ and $\neg(A \wedge \neg A)$ are valid for every formula A.
- The formulæ P and $P \wedge (P \to Q)$ are satisfiable: they are both true under the interpretation that maps P and Q to **t**. But they are not valid: they are both false under the interpretation that maps P and Q to **f**.
- The formula $\neg A$ is unsatisfiable for every valid formula A. This set of formulæ is unsatisfiable: $\{P, Q, \neg P \vee \neg Q\}$

Exercise 1 Is the formula $A \to \neg A$ satisfiable? Is it valid?

2.3 Applications of propositional logic

Hardware design is the obvious example. Propositional logic is used to minimize the number of gates in a circuit, and to show the equivalence of combinational circuits. There now exist highly efficient tautology checkers, such as OBDDs (Ordered Binary Decision Diagrams), which have been used to verify complex combinational circuits. This is an important branch of hardware verification.

Chemical synthesis is a more offbeat example.[1] Under suitable conditions, the following chemical reactions are possible:

$$\mathrm{HCl} + \mathrm{NaOH} \to \mathrm{NaCl} + \mathrm{H_2O}$$
$$\mathrm{C} + \mathrm{O_2} \to \mathrm{CO_2}$$
$$\mathrm{CO_2} + \mathrm{H_2O} \to \mathrm{H_2CO_3}$$

Show we can make H_2CO_3 given supplies of HCl, NaOH, O_2, and C.

Chang and Lee formalize the supplies of chemicals as four axioms and prove that H_2CO_3 logically follows. The idea is to formalize each compound as a propo-

[1] Chang and Lee, page 21, as amended by Ross Younger

sitional symbol and express the reactions as implications:

$$\mathrm{HCl} \wedge \mathrm{NaOH} \to \mathrm{NaCl} \wedge \mathrm{H_2O}$$
$$\mathrm{C} \wedge \mathrm{O_2} \to \mathrm{CO_2}$$
$$\mathrm{CO_2} \wedge \mathrm{H_2O} \to \mathrm{H_2CO_3}$$

Note that this involves an ideal model of chemistry. What if the reactions can be inhibited by the presence of other chemicals? Proofs about the real world *always* depend upon general assumptions. It is essential to bear these in mind when relying on such a proof.

2.4 Equivalences

Note that $A \leftrightarrow B$ and $A \simeq B$ are different kinds of assertions. The formula $A \leftrightarrow B$ refers to some fixed interpretation, while the metalanguage statement $A \simeq B$ refers to all interpretations. On the other hand, $\models A \leftrightarrow B$ means the same thing as $A \simeq B$. Both are metalanguage statements, and $A \simeq B$ is equivalent to saying that the formula $A \leftrightarrow B$ is a tautology.

Similarly, $A \to B$ and $A \models B$ are are different kinds of assertions, while $\models A \to B$ and $A \models B$ mean the same thing. The formula $A \to B$ is a tautology if and only if $A \models B$.

Here is a listing of some of the more basic equivalences of propositional logic. They provide one means of reasoning about propositions, namely by transforming one proposition into an equivalent one. They are also needed to convert propositions into various normal forms.

idempotency laws

$$A \wedge A \simeq A$$
$$A \vee A \simeq A$$

commutative laws

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

associative laws

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

distributive laws

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

de Morgan laws

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

definitions of connectives

$$A \leftrightarrow B \simeq (A \rightarrow B) \wedge (B \rightarrow A)$$
$$\neg A \simeq A \rightarrow \mathbf{f}$$
$$A \rightarrow B \simeq \neg A \vee B$$

more negation laws

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

simplification

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

Propositional logic enjoys a principle of duality: for every equivalence $A \simeq B$ there is another equivalence $A' \simeq B'$, where A', B' are derived from A, B by exchanging $\wedge$ with $\vee$ and $\mathbf{t}$ with $\mathbf{f}$. Before applying this rule, remove all occurrences of $\rightarrow$ and $\leftrightarrow$, since they implicitly involve $\wedge$ and $\vee$.

Exercise 2 Verify some of the equivalences using truth tables.

2.5 Normal forms

The language of propositional logic is redundant: many of the connectives can be defined in terms of others. By repeatedly applying certain equivalences, we can transform a formula into a *normal form*. A typical normal form eliminates certain connectives entirely, and uses others in a restricted manner. The restricted structure makes the formula easy to process, although the normal form may be exponentially larger than the original formula. Most normal forms are unreadable, although Negation Normal Form is not too bad.

Definition 2 A *literal* is an atomic formula or its negation. Let K, L, L', ... stand for literals.

A *maxterm* is a literal or a disjunction of literals.

A *minterm* is a literal or a conjunction of literals.

A formula is in *Negation Normal Form* (NNF) if the only connectives in it are $\wedge$, $\vee$, and $\neg$, where $\neg$ is only applied to atomic formulæ.

A formula is in *Conjunctive Normal Form* (CNF) if it has the form $A_1 \wedge \cdots \wedge A_m$, where each A_i is maxterm.

A formula is in *Disjunctive Normal Form* (DNF) if it has the form $A_1 \vee \cdots \vee A_m$, where each A_i is a minterm.

An atomic formula like P is in all the normal forms NNF, CNF, and DNF. The formula

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

is in CNF. To get an example of a DNF formula, exchange $\wedge$ and $\vee$ above. Every formula in CNF or DNF is also in NNF, but the NNF formula

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

is neither CNF nor DNF.

One advantage of NNF is that it reveals the true nature of a formula. For example, converting $\neg(A \rightarrow B)$ to NNF yields $A \wedge \neg B$. This reveals that the original formula was effectively a conjunction.

2.6 Translation to normal form

Every formula can be translated into an equivalent formula in NNF, CNF, or DNF by means of the following steps.

Step 1. Eliminate $\leftrightarrow$ and $\rightarrow$ by repeatedly applying the following equivalences:

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

Step 2. Push negations in until they apply only to atoms, repeatedly replacing by the equivalences

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

At this point, the formula is in Negation Normal Form.

Step 3. To obtain CNF, push disjunctions in until they apply only to literals. Repeatedly replace by the equivalences

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

Note that the second replacement is the same distributive law with the operands reversed (using the commutative law).

Step 4. Simplify the resulting CNF by deleting any maxterm that contains both P and $\neg P$, since it is equivalent to $\mathbf{t}$. Also delete any maxterm that includes another maxterm (meaning, every literal in the latter is also present in the former). This is correct because $A \wedge (A \vee B) \simeq A$. Finally, two maxterms of the form $P \vee A$ and $\neg P \vee A$ can be replaced by A, thanks to the equivalence

$$(P \vee A) \wedge (\neg P \vee A) \simeq A.$$

This simplification is related to the resolution rule, which we shall study later.

Since $\vee$ is commutative, saying 'a maxterm of the form $A \vee B$' refers to any possible way of arranging the literals into two parts. This includes $A \vee \mathbf{f}$, since one of those parts may be empty and the empty disjunction is false. So in the last simplification above, two maxterms of the form P and $\neg P$ can be replaced by $\mathbf{f}$.

Steps 3' and 4'. To obtain DNF, apply instead the other distributive law:

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

Exactly the same simplifications can be performed for DNF as for CNF, exchanging the roles of $\wedge$ and $\vee$.

2.7 Tautology checking using CNF

Here is a method of proving theorems in propositional logic. To prove A, reduce it to CNF. If the simplified CNF formula is **t** then A is valid because each transformation preserves logical equivalence. And if the CNF formula is not **t**, then A is not valid.

To see why, suppose the CNF formula is $A_1 \wedge \cdots \wedge A_m$. If A is valid then each A_i must also be valid. Write A_i as $L_1 \vee \cdots \vee L_n$, where the L_j are literals. We can make an interpretation I that falsifies every L_j, and therefore falsifies A_i. Define I such that, for every propositional letter P,

$$I(P) = \begin{cases} \mathbf{t} & \text{if } L_j \text{ is } P \text{ for some } j \\ \mathbf{f} & \text{if } L_j \text{ is } \neg P \text{ for some } j \end{cases}$$

This definition is legitimate because there cannot exist literals L_j and L_k such that L_j is $\neg L_k$; if there did, then simplification would have deleted the disjunction A_i.

The powerful OBDD method is related to this CNF method. It uses an if-then-else data structure, an ordering on the propositional letters, and some standard algorithmic techniques (such as hashing) to gain efficiency.

Example 1 Start with

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

Step 1, eliminate $\rightarrow$, gives

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

Step 2, push negations in, gives

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

Step 3, push disjunctions in, gives

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

Simplifying yields

$$(\neg P \vee Q \vee R) \wedge \mathbf{t}$$

$$\neg P \vee Q \vee R$$

The interpretation $P \mapsto \mathbf{t},\ Q \mapsto \mathbf{f},\ R \mapsto \mathbf{f}$ falsifies this formula, which is equivalent to the original formula. So the original formula is not valid.

Example 2 Start with

$$P \wedge Q \rightarrow Q \wedge P$$

Step 1, eliminate $\rightarrow$, gives

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

Step 2, push negations in, gives

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

Step 3, push disjunctions in, gives

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

Simplifying yields $\mathbf{t} \wedge \mathbf{t}$, which is $\mathbf{t}$. Both conjuncts are valid since they contain a formula and its negation. Thus $P \wedge Q \rightarrow Q \wedge P$ is valid.

Example 3 Peirce's law is another example. Start with

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

Step 1, eliminate $\rightarrow$, gives

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

Step 2, push negations in, gives

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

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

Step 3, push disjunctions in, gives

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

Simplifying again yields $\mathbf{t}$. Thus Peirce's law is valid.

There is a dual method of refuting A (proving inconsistency). To refute A, reduce it to DNF, say $A_1 \vee \cdots \vee A_m$. If A is inconsistent then so is each A_i. Suppose A_i is $L_1 \wedge \cdots \wedge L_n$, where the L_j are literals. If there is some literal L' such that the L_j include both L' and $\neg L'$, then A_i is inconsistent. If not then there is an interpretation that verifies every L_j — and so A_i.

To prove A, we can use the DNF method to refute $\neg A$. The steps are exactly the same as the CNF method because the extra negation swaps every $\vee$ and $\wedge$. Gilmore implemented a theorem prover based upon this method in 1960 (see Chang and Lee, page 62).

Exercise 3 Each of the following formulæ is satisfiable but not valid. Exhibit an interpretation that makes the formula true and another interpretation that makes the formula false.

$$A \to B \qquad A \vee B \to A \wedge B$$
$$\neg(A \vee B \vee C) \qquad \neg(A \wedge B) \wedge \neg(B \vee C) \wedge (A \vee C)$$

Exercise 4 Convert of the following propositional formulæ into Conjunctive Normal Form and also into Disjunctive Normal Form. For each formula, state whether it is valid, satisfiable, or unsatisfiable; justify each answer.

$$(A \to B) \wedge (B \to A)$$
$$((A \wedge B) \vee C) \wedge (\neg((A \vee C) \wedge (B \vee C)))$$
$$\neg(A \vee B \vee C) \vee ((A \wedge B) \vee C)$$
$$\neg(A \vee B \to C) \wedge (A \to C) \wedge (B \to C)$$

Exercise 5 Using ML or Lisp, define data structures for representing propositions and interpretations. Write a function to test whether or not a proposition holds under an interpretation (both supplied as arguments). Write a function to convert a proposition to Negation Normal Form.

3 Proof Systems for Propositional Logic

We can verify any tautology by checking all possible interpretations, using the truth tables. This is a *semantic* approach, since it appeals to the meanings of the connectives.

The *syntactic* approach is formal proof: generating theorems, or reducing a conjecture to a known theorem, by applying syntactic transformations of some sort. We have already seen a proof method based on CNF. Most proof methods are based on axioms and inference rules.

What about efficiency? Deciding whether a propositional formula is satisfiable is an NP-complete problem (Aho, Hopcroft and Ullman 1974, pages 377–383). Thus all approaches are likely to be exponential in the length of the formula.

3.1 A Hilbert-style proof system

Here is a simple proof system for propositional logic. The following definitions apply to this section only! There are countless similar proof systems. They are often called *Hilbert-style* after the logician David Hilbert, although they existed before him.

Note that $A \to (B \to A)$ is a tautology. Call it Axiom K. Also,

$$(A \to (B \to C)) \to ((A \to B) \to (A \to C))$$

is a tautology. Call it Axiom S. The Double-Negation Law $\neg\neg A \to A$, is a tautology. Call it Axiom DN.

These axioms are more properly called *axiom schemes*, since we assume all instances of them that can be obtained by substituting formulæ for A, B and C.

Let us define the other logical connectives in terms of implication:

$$\neg A \stackrel{\text{def}}{=} A \to \mathbf{f}$$
$$A \vee B \stackrel{\text{def}}{=} \neg A \to B$$
$$A \wedge B \stackrel{\text{def}}{=} \neg(\neg A \vee \neg B)$$

Whenever $A \to B$ and A are both valid, it follows that B is valid. We write this as the inference rule

$$\frac{A \to B \quad A}{B.}$$

This rule is traditionally called Modus Ponens. Together with Axioms K, S, and DN and the definitions, it suffices to prove all tautologies of (classical) propositional logic.[2] However, this formalization of propositional logic is inconvenient to use. For example, try proving $A \to A$!

A variant of this proof system replaces the Double-Negation Law by the Contrapositive Law:

$$(\neg B \to \neg A) \to (A \to B)$$

Another formalization of propositional logic consists of the Modus Ponens rule plus the following axioms:

$$A \vee A \to A$$
$$B \to A \vee B$$
$$A \vee B \to B \vee A$$
$$(B \to C) \to (A \vee B \to A \vee C)$$

Here $A \wedge B$ and $A \to B$ are defined in terms of $\neg$ and $\vee$.

[2] If the Double-Negation Law is omitted, only the intuitionistic tautologies are provable. This axiom system is connected with the combinators S and K and the λ-calculus.

3.2 Gentzen's Natural Deduction Systems

Natural proof systems do exist. Natural deduction, devised by Gerhard Gentzen, is based upon three principles:

1. Proof takes place within a varying context of assumptions.
2. Each logical connective is defined independently of the others. (This is possible because item 1 eliminates the need for tricky uses of implication.)
3. Each connective is defined by *introduction* and *elimination* rules.

For example, the *introduction* rule for $\wedge$ describes how to deduce $A \wedge B$:

$$\frac{A \quad B}{A \wedge B} \ (\wedge i)$$

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

$$\frac{A \wedge B}{A} \ (\wedge e1) \qquad \frac{A \wedge B}{B} \ (\wedge e2)$$

The elimination rule for $\rightarrow$ says what to deduce from $A \rightarrow B$. It is just Modus Ponens:

$$\frac{A \rightarrow B \quad A}{B} \ (\rightarrow e)$$

The introduction rule for $\rightarrow$ says that $A \rightarrow B$ is proved by assuming A and deriving B:

$$\frac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \rightarrow B} \ (\rightarrow i)$$

For simple proofs, this notion of assumption is pretty intuitive. Here is a proof of the formula $A \wedge B \rightarrow A$:

$$\frac{\dfrac{[A \wedge B]}{A} \ (\wedge e1)}{A \wedge B \rightarrow A} \ (\rightarrow i)$$

The key point is that rule $(\rightarrow i)$ *discharges* its assumption: the assumption could be used to prove A from $A \wedge B$, but is no longer available once we conclude $A \wedge B \rightarrow A$.

The introduction rules for $\vee$ are straightforward:

$$\frac{A}{A \vee B}\ (\vee i1) \qquad \frac{B}{A \vee B}\ (\vee i2)$$

The elimination rule says that to show some C from $A \vee B$ there are two cases to consider, one assuming A and one assuming B:

$$\frac{A \vee B \qquad \begin{array}{c}[A] \\ \vdots \\ C\end{array} \qquad \begin{array}{c}[B] \\ \vdots \\ C\end{array}}{C}\ (\vee e)$$

The scope of assumptions can get confusing in complex proofs. Let us switch attention to the sequent calculus, which is similar in spirit but easier to use.

3.3 The sequent calculus

The *sequent calculus* resembles natural deduction, but it makes the set of assumptions explicit. Thus, it is more concrete.

A *sequent* has the form $\Gamma \Rightarrow \Delta$, where Γ and Δ are finite sets of formulæ.[3] These sets may be empty. The sequent

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

is *true* if $A_1 \wedge \ldots \wedge A_m$ implies $B_1 \vee \ldots \vee B_n$. In other words, we assume that each of $A_1, \ldots, A_m$ are true and try to show that at least one of $B_1, \ldots, B_n$ is true.

A *basic* sequent is one in which the same formula appears on both sides, as in $P, B \Rightarrow B, R$. This sequent is true because, if all the formulæ on the left side are true, then in particular B is; so, least one right-side formula (B again) is true. Our calculus therefore regards all basic sequents as proved.

Every basic sequent might be written in the form $\{A\} \cup \Gamma \Rightarrow \{A\} \cup \Delta$, where A is the common formula and Γ and Δ are the other left- and right-side formulæ, respectively. The sequent calculus identifies the one-element set $\{A\}$ with its element A and denotes union by a comma. Thus, the correct notation for the general form of a basic sequent is $A, \Gamma \Rightarrow A, \Delta$.

Sequent rules are almost always used backward. We start with the sequent that we would like to prove and, working backwards, reduce it to simpler sequents in the hope of rendering them trivial. The forward direction would be to start with known facts and derive new facts, but this approach tends to generate random theorems rather than ones we want.

[3] With minor changes, sequents can instead be lists or multisets.

Sequent rules are classified as *right* or *left*, indicating which side of the $\Rightarrow$ symbol they operate on. Rules that operate on the right side are analogous to natural deduction's introduction rules, and left rules are analogous to elimination rules.

The sequent calculus analogue of $(\to i)$ is the rule

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

Working backwards, this rule breaks down some implication on the right side of a sequent; Γ and Δ stand for the sets of formulæ that are unaffected by the inference. The analogue of the pair $(\vee i1)$ and $(\vee i2)$ is the single rule

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

This breaks down some disjunction on the right side, replacing it by both disjuncts. Thus, the sequent calculus is a kind of multiple-conclusion logic. Figure 1 summarises the rules.

basic sequent: $A, \Gamma \Rightarrow A, \Delta$

Negation rules:

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

Conjunction rules:

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

Disjunction rules:

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

Implication rules:

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

Figure 1: Sequent Rules for Propositional Logic

To illustrate the use of multiple formulæ on the right, let us prove the classical theorem $(A \to B) \vee (B \to A)$. Working backwards (or upwards), we reduce this

formula to a basic sequent:

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

The basic sequent has a line over it to emphasize that it is provable.

This example is typical of the sequent calculus: start with the desired theorem and apply rules *backwards* in a fairly arbitrary manner. This yields a surprisingly effective proof procedure.

Here is part of a proof of the distributive law $A \vee (B \wedge C) \simeq (A \vee B) \wedge (A \vee C)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \Rightarrow A, B} \quad \dfrac{\overline{B, C \Rightarrow A, B}}{B \wedge C \Rightarrow A, B}\,(\wedge l)}{A \vee (B \wedge C) \Rightarrow A, B}\,(\vee l)}{A \vee (B \wedge C) \Rightarrow A \vee B}\,(\vee r) \qquad \text{similar}}{A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (A \vee C)}\,(\wedge r)$$

The second, omitted proof tree proves $A \vee (B \wedge C) \Rightarrow A \vee C$ similarly.

Finally, here is a failed proof of the invalid formula $A \vee B \to B \vee C$.

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

The sequent $A \Rightarrow B, C$ has no line over it because it is not valid! The interpretation $A \mapsto \mathbf{t},\ B \mapsto \mathbf{f},\ C \mapsto \mathbf{f}$ falsifies it. We have already seen this as Example 1 (page 9).

Structural rules concern sequents in general rather than particular connectives. The *weakening* rules allow additional formulas to be inserted on the left or right side. Obviously, if $\Gamma \Rightarrow \Delta$ holds then the sequent continues to hold after further assumptions or goals are added:

$$\dfrac{\Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\,(weaken{:}l) \qquad \dfrac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, A}\,(weaken{:}r)$$

Exchange rules allow formulas in a sequent to be re-ordered. We don't need them because our sequents are sets rather than lists. Contraction rules allow formulas to be used more than once:

$$\dfrac{A, A, \Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\,(contract{:}l) \qquad \dfrac{\Gamma \Rightarrow \Delta, A, A}{\Gamma \Rightarrow \Delta, A}\,(contract{:}r)$$

Because the sets $\{A\}$ and $\{A, A\}$ are identical, we don't need contraction rules either. Moreover, it turns out that we almost never need to use a formula more than once. Exceptions are $\forall x\ A$ (when it appears on the left) and $\exists x\ A$ (when it appears on the right).

The *cut rule* allows the use of lemmas. Some formula A is proved in the first premise, and assumed in the second premise. A famous result, the *cut-elimination theorem*, states that this rule is not required. All uses of it can be removed from any proof, but the proof could get exponentially larger.

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

This special case of cut may be easier to understand. We prove lemma A from Γ and use A and Γ together to reach the conclusion B.

$$\frac{\Gamma \Rightarrow B, A \quad A, \Gamma \Rightarrow B}{\Gamma \Rightarrow B}$$

Since Γ contains as much information as A, it is natural to expect that such lemmas should not be necessary, but the cut-elimination theorem is quite hard to prove.

Note On the Thor directory `/group/clteach/lcp/Logic` there is a simple theorem prover called `folderol.ML`. It can prove easy first-order theorems using the sequent calculus, and outputs a summary of each proof. The file begins with very basic instructions describing how to run it. The file `testsuite.ML` contains further instructions and numerous examples.

Exercise 6 Prove the following sequents:

$$\neg\neg A \Rightarrow A$$
$$A \wedge B \Rightarrow B \wedge A$$
$$A \vee B \Rightarrow B \vee A$$

Exercise 7 Prove the following sequents:

$$(A \wedge B) \wedge C \Rightarrow A \wedge (B \wedge C)$$
$$(A \vee B) \wedge (A \vee C) \Rightarrow A \vee (B \wedge C)$$
$$\neg(A \vee B) \Rightarrow \neg A \wedge \neg B$$

4 Ordered Binary Decision Diagrams

A binary decision tree represents a propositional formulæ by binary decisions, namely if-then-else expressions over the propositional letters. (In the relevant literature, propositional letters are called *variables*.) A tree may contain much redundancy; a binary decision *diagram* is a directed graph, sharing identical subtrees. An *ordered* binary decision diagram (OBDD) is based upon giving an ordering $<$ to the variables: they must be tested in order. Further refinements ensure that each propositional formula is mapped to a unique OBDD, for a given ordering.

An OBDD representation must satisfy the following conditions:

- *ordering*: if P is tested before Q, then $P < Q$
  (thus in particular, P cannot be tested more than once on a single path)
- *uniqueness*: identical subgraphs are stored only once
  (to do this efficiently, hash each node by its variable and pointer fields)
- *irredundancy*: no test leads to identical subgraphs in the **t** and **f** cases
  (thanks to uniqueness, redundant tests can be detected by comparing pointers)

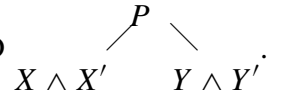
Because the OBDD representation of each formula is unique, it is called a *canonical form*. Canonical forms usually lead to good algorithms — for a start, you can test whether two things are equivalent by comparing their canonical forms.

The OBDD form of any tautology is **t**. Similarly, that of any inconsistent formula is **f**. To check whether two formulæ are logically equivalent, convert both to OBDD form and then — thanks to uniqueness — simply compare the pointers.

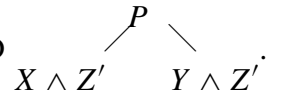
A recursive algorithm converts a formula to an OBDD. All the logical connectives can be handled directly, including $\rightarrow$ and $\leftrightarrow$. (Exclusive 'or' is also used, especially in hardware examples.) The expensive transformation of $A \leftrightarrow B$ into $(A \rightarrow B) \wedge (B \rightarrow A)$ is completely unnecessary.

Here is how to convert a conjunction $A \wedge A'$ to an OBDD.

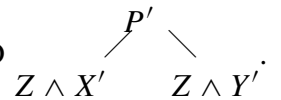
1. Recursively convert A and A' to OBDDs Z and Z'.
2. Check for trivial cases. If $Z = Z'$ (pointer comparison) then the result is Z; if either operand is **f**, then the result is **f**; if either operand is **t**, then the result is the other operand.
3. In the general case, let $Z = \begin{array}{ccc} & P & \\ X & & Y \end{array}$ and $Z' = \begin{array}{ccc} & P' & \\ X' & & Y' \end{array}$. There are three possibilities:

(a) If $P = P'$ then recursively build the OBDD 

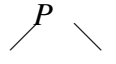
.
This means convert $X \wedge X'$ and $Y \wedge Y'$ to OBDDs U and U', then attempt to construct a new decision node from P to them. The usual simplifications are done. If $U = U'$ then the resulting OBDD for the conjunction is U. If an identical decision node from P to (U, U') has been created previously, then that existing node is used instead of creating a new one.

(b) If $P < P'$ then recursively build the OBDD 

.

(c) If $P > P'$ then recursively build the OBDD 

.

Other connectives are treated similarly; they differ only in the trivial cases. The negation of the OBDD 

is

. In essence we copy the OBDD, and when we reach the leaves, exchange **t** and **f**.

During this processing, the same input (consisting of a connective and two OBDDs) may be transformed into an OBDD repeatedly. Efficient implementations therefore have an additional hash table, which associates inputs to the corresponding OBDDs. The result of every transformation is stored in the hash table so that it does not have to be computed again.

For more details, please see Bryant (1992), part of which is attached as an appendix to these notes. Section 3.1 describes a construction method that can be done using pencil and paper. In essence, the idea goes like this:

- Draw the two OBDDs to be combined by a connective such as $\wedge$.
- Label the nodes of the first one $A_1, \ldots, A_m$ and of the second one $B_1, \ldots, B_n$.
- Draw the graph that results from merging the two OBDDs while respecting the variable ordering. Each node of the new graph will have a label of the form $A_i B_j$, and will be a decision on the earlier of the two Boolean variables.
- Apply the connective to the zeroes and ones in the leaves.
- Simplify the new OBDD by deleting redundant nodes.

You will need to read Section 3.1 of Bryant (1992) in order to understand this method properly.

Exercise 8 Compute the OBDD for each of the following formulæ, taking the variables as alphabetically ordered:

$$P \wedge Q \to Q \wedge P \qquad P \vee Q \to P \wedge Q$$
$$\neg(P \vee Q) \vee P \qquad \neg(P \wedge Q) \leftrightarrow (P \vee R)$$

Exercise 9 Verify the following equivalences using OBDDs:

$$(P \wedge Q) \wedge R \simeq P \wedge (Q \wedge R)$$
$$(P \vee Q) \vee R \simeq P \vee (Q \vee R)$$
$$P \vee (Q \wedge R) \simeq (P \vee Q) \wedge (P \vee R)$$
$$P \wedge (Q \vee R) \simeq (P \wedge Q) \vee (P \wedge R)$$

Exercise 10 Verify the following equivalences using OBDDs:

$$\neg(P \wedge Q) \simeq \neg P \vee \neg Q$$
$$(P \leftrightarrow Q) \leftrightarrow R \simeq P \leftrightarrow (Q \leftrightarrow R)$$
$$(P \vee Q) \to R \simeq (P \to R) \wedge (Q \to R)$$

5 First-order Logic

First-order logic (FOL) extends propositional logic to allow reasoning about the members (such as numbers) of some non-empty universe. It uses the quantifiers $\forall$ ('for all') and $\exists$ ('there exists'). First-order logic has variables ranging over 'individuals,' but not over functions or predicates; such variables are found in second- or higher-order logic.

5.1 Syntax of first-order Logic

Terms stand for individuals while *formulæ* stand for truth values. We assume there is an infinite supply of *variables* x, y, ... that range over individuals. A *first-order language* specifies symbols that may appear in terms and formulæ. A first-order language $\mathcal{L}$ contains, for all $n \geq 0$, a set of n-place *function symbols* f, g, ... and n-place *predicate symbols* P, Q, ... These sets may be empty, finite, or infinite.

Constant symbols a, b, ... are simply 0-place function symbols. Predicate symbols are also called *relation symbols*. Prolog programmers refer to function symbols as *functors*.

Definition 3 The *terms* $t, u, \ldots$ of a first-order language are defined recursively as follows:

- A variable is a term.
- A constant symbol is a term.
- If $t_1, \ldots, t_n$ are terms and f is an n-place function symbol then $f(t_1, \ldots, t_n)$ is a term.

Definition 4 The *formulæ* $A, B, \ldots$ of a first-order language are defined recursively as follows:

- If $t_1, \ldots, t_n$ are terms and P is an n-place predicate symbol then $P(t_1, \ldots, t_n)$ is a formula (called an *atomic formula*).
- If A and B are formulæ then $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$ are also formulæ.
- If x is a variable and A is a formula then $\forall x\ A$ and $\exists x\ A$ are also formulæ.

Brackets are used in the conventional way for grouping. Terms and formulæ are tree-like data structures, not strings.

The quantifiers $\forall x\ A$ and $\exists x\ A$ bind tighter than the binary connectives; thus $\forall x\ A \wedge B$ is equivalent to $(\forall x\ A) \wedge B$. Sometimes you will see an alternative quantifier syntax, $\forall x\ .\ A$ and $\exists x\ .\ B$, which binds looser than the binary connectives; thus $\forall x\ .\ A \wedge B$ is equivalent to $\forall x\ .\ (A \wedge B)$. The dot is the giveaway; be careful!

Nested quantifications such as $\forall x\ \forall y\ A$ are abbreviated to $\forall x y\ A$.

Example 4 A language for arithmetic might have the constant symbols 0, 1, 2, ..., and function symbols $+$, $-$, $\times$, $/$, and the predicate symbols $=$, $<$, $>$, ... We informally may adopt an infix notation for the function and predicate symbols. Terms include 0 and $(x + 3) - y$; formulæ include $y = 0$ and $x + y < y + z$.

5.2 Examples of statements in first-order logic

Here are some sample formulæ with a rough English translation. English is easier to understand but is too ambiguous for long derivations.

All professors are brilliant:

$$\forall x\ (\text{professor}(x) \rightarrow \text{brilliant}(x))$$

The income of any banker is greater than the income of any bedder:

$$\forall x\, y\ (\text{banker}(x) \wedge \text{bedder}(y) \rightarrow \text{income}(x) > \text{income}(y))$$

Note that $>$ is a 2-place relation symbol. The infix notation is simply a convention.

Every student has a supervisor:

$$\forall x\ (\text{student}(x) \rightarrow \exists y\ \text{supervises}(y, x))$$

This does not preclude a student having several supervisors.

Every student's tutor is a member of the same College:

$$\forall x\, y\ (\text{student}(x) \wedge \text{college}(y) \wedge \text{member}(x, y) \rightarrow \text{member}(\text{tutor}(x), y))$$

The use of a function 'tutor' incorporates the assumption that every student has exactly *one* tutor.

A mathematical example: *there exist infinitely many Pythagorean triples*:

$$\forall n\ \exists i\, j\, k\ (i > n \wedge i^2 + j^2 = k^2)$$

Here the superscript 2 refers to the squaring function. Equality (=) is just another relation symbol (satisfying suitable axioms) but there are many special techniques for it.

First-order logic assumes a non-empty domain: thus $\forall x\ P(x)$ implies $\exists x\ P(x)$. If the domain could be empty, even $\exists x\, \mathbf{t}$ could fail to hold. Note also that $\forall x\ \exists y\ y^2 = x$ is true if the domain is the complex numbers, and is false if the domain is the integers or reals. We determine properties of the domain by asserting the set of statements it must satisfy.

There are many other forms of logic. *Many-sorted first-order logic* assigns types to each variable, function symbol and predicate symbol, with straightforward type checking; types are called *sorts* and denote non-empty domains. *Second-order logic* allows quantification over functions and predicates. It can express mathematical induction by

$$\forall P\ [P(0) \wedge \forall k\ (P(k) \rightarrow P(k+1)) \rightarrow \forall n\ P(n)],$$

using quantification over the unary predicate P. In second-order logic, these functions and predicates must themselves be first-order, taking no functions or predicates as arguments. *Higher-order logic* allows unrestricted quantification over functions and predicates of any order. The list of logics could be continued indefinitely.

5.3 Formal semantics of first-order logic

Let us rigorously define the meaning of formulæ. An interpretation of a language maps its function symbols to actual functions, and its relation symbols to actual relations. For example, the predicate symbol 'student' could be mapped to the set of all students currently enrolled at the University.

Definition 5 Let $\mathcal{L}$ be a first-order language. An *interpretation* $\mathcal{I}$ of $\mathcal{L}$ is a pair (D, I). Here D is a nonempty set, the *domain* or *universe*. The operation I maps symbols to individuals, functions or sets:

- if c is a constant symbol (of $\mathcal{L}$) then $I[c] \in D$
- if f is an n-place function symbol then $I[f] \in D^n \to D$ (which means $I[f]$ is an n-place function on D)
- if P is an n-place relation symbol then $I[P] \subseteq D^n$ (which means $I[P]$ is an n-place relation on D)

There are various ways of talking about the values of variables under an interpretation. One way is to 'invent' a constant symbol for every element of D. More natural is to represent the values of variables using an environment, known as a *valuation*.

Definition 6 A *valuation* V of $\mathcal{L}$ over D is a function from the variables of $\mathcal{L}$ into D. Write $\mathcal{I}_V[t]$ for the value of t with respect to $\mathcal{I}$ and V, defined by

$$\begin{aligned} \mathcal{I}_V[x] &\stackrel{\text{def}}{=} V(x) \qquad \text{if } x \text{ is a variable} \\ \mathcal{I}_V[c] &\stackrel{\text{def}}{=} I[c] \\ \mathcal{I}_V[f(t_1, \ldots, t_n)] &\stackrel{\text{def}}{=} I[f](\mathcal{I}_V[t_1], \ldots, \mathcal{I}_V[t_n]) \end{aligned}$$

Write $V\{a/x\}$ for the valuation that maps x to a and is otherwise the same as V. Typically, we modify a valuation one variable at a time. This is a semantic analogue of substitution for the variable x.

5.4 What is truth?

We now can define truth itself. (First-order truth, that is!) This formidable definition formalizes the intuitive meanings of the connectives. Thus it almost looks like a tautology. It effectively specifies each connective by English descriptions. Valuations help specify the meanings of quantifiers. Alfred Tarski first defined truth in this manner.

Definition 7 Let A be a formula. Then for an interpretation $\mathcal{I} = (D, I)$ write $\models_{\mathcal{I},V} A$ to mean 'A is true in $\mathcal{I}$ under V.' This is defined by cases on the construction of the formula A:

$\models_{\mathcal{I},V} P(t_1, \ldots, t_n)$ if $I[P](\mathcal{I}_V[t_1], \ldots, \mathcal{I}_V[t_n])$ holds (that is, the actual relation $I[P]$ holds of the values of the arguments)

$\models_{\mathcal{I},V} t = u$ if $\mathcal{I}_V[t]$ equals $\mathcal{I}_V[u]$ (if $=$ is a predicate symbol of the language, then we insist that it really denotes equality)

$\models_{\mathcal{I},V} \neg B$ if $\models_{\mathcal{I},V} B$ does not hold

$\models_{\mathcal{I},V} B \wedge C$ if $\models_{\mathcal{I},V} B$ and $\models_{\mathcal{I},V} C$

$\models_{\mathcal{I},V} B \vee C$ if $\models_{\mathcal{I},V} B$ or $\models_{\mathcal{I},V} C$

$\models_{\mathcal{I},V} B \rightarrow C$ if $\models_{\mathcal{I},V} B$ does not hold or $\models_{\mathcal{I},V} C$

$\models_{\mathcal{I},V} B \leftrightarrow C$ if $\models_{\mathcal{I},V} B$ and $\models_{\mathcal{I},V} C$ both hold or neither hold

$\models_{\mathcal{I},V} \exists x\, B$ if there exists $m \in D$ such that $\models_{\mathcal{I},V\{m/x\}} B$ holds (that is, B holds when x has the value m)

$\models_{\mathcal{I},V} \forall x\, B$ if for all $m \in D$ we have that $\models_{\mathcal{I},V\{m/x\}} B$ holds

The cases for $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$ follow the propositional truth tables.

Write $\models_{\mathcal{I}} A$ provided $\models_{\mathcal{I},V} A$ for all V. Clearly, if A is closed (contains no free variables) then its truth is independent of the valuation. If A contains free variables $x_1, \ldots, x_n$ then these in effect are universally quantified:

$$\models_{\mathcal{I}} A \quad \text{if and only if} \quad \models_{\mathcal{I}} \forall x_1 \cdots \forall x_n\, A$$

The definitions of valid, satisfiable, etc. carry over almost verbatim from Section 2.2.

Definition 8 An interpretation $\mathcal{I}$ *satisfies* a formula if $\models_{\mathcal{I}} A$ holds.

A set S of formulæ is *valid* if every interpretation of S satisfies every formula in S.

A set S of formulæ is *satisfiable* (or *consistent*) if there is some interpretation of S that satisfies every formula in S.

A set S of formulæ is *unsatisfiable* (or *inconsistent*) if it is not satisfiable. (Each interpretation falsifies some formula of S.)

A *model* of a set S of formulæ is an interpretation that satisfies every formula in S. We also consider models that satisfy a single formula.

Unlike in propositional logic, models can be infinite and there can be an infinite number of models. There is no chance of proving validity by checking all models. We must rely on proof.

Example 5 The formula $P(a) \wedge \neg P(b)$ is satisfiable. Consider the interpretation with $D = \{0, 1\}$ and I defined by

$$I[a] = 0$$
$$I[b] = 1$$
$$I[P] = \{0\}$$

On the other hand, $P(x) \wedge \neg P(y)$ is unsatisfiable. Its free variables are taken to be universally quantified, so it is equivalent to $\forall xy\,(P(x) \wedge \neg P(y))$.

The formula $(\exists x\ P(x)) \rightarrow P(c)$ holds in the interpretation (D, I) where $D = \{0, 1\}$, $I[P] = \{0\}$, and $I[c] = 0$. (Thus $P(x)$ means 'x equals 0' and c denotes 0.) If we modify this interpretation by making $I[c] = 1$ then the formula no longer holds. Thus it is satisfiable but not valid.

The formula $(\forall x\ P(x)) \rightarrow (\forall x\ P(f(x)))$ is valid, for let (D, I) be an interpretation. If $\forall x\ P(x)$ holds in this interpretation then $P(x)$ holds for all $x \in D$, thus $I[P] = D$. The symbol f denotes some actual function $I[f] \in D \rightarrow D$. Since $I[P] = D$ and $I[f](x) \in D$ for all $x \in D$, formula $\forall x\ P(f(x))$ holds.

The formula $\forall xy\ x = y$ is satisfiable but not valid; it is true in every domain that consists of exactly one element. (The empty domain is not allowed in first-order logic.)

Example 6 Let $\mathcal{L}$ be the first-order language consisting of the constant 0 and the (infix) 2-place function symbol $+$. An interpretation $\mathcal{I}$ of this language is any non-empty domain D together with values $I[0]$ and $I[+]$, with $I[0] \in D$ and $I[+] \in D \times D \rightarrow D$. In the language $\mathcal{L}$ we may express the following axioms:

$$x + 0 = x$$
$$0 + x = x$$
$$(x + y) + z = x + (y + z)$$

(Remember, free variables in effect are universally quantified, by the definition of $\models_{\mathcal{I}} A$.) One model of these axioms is the set of natural numbers, provided we give 0 and $+$ the obvious meanings. But the axioms have many other models.[4] Below, let A be some set.

1. The set of all strings (in ML say) letting 0 denote the empty string and $+$ string concatenation.
2. The set of all subsets of A, letting 0 denote the empty set and $+$ union.
3. The set of functions in $A \rightarrow A$, letting 0 denote the identity function and $+$ composition.

[4]Models of these axioms are called *monoids*.

Exercise 11 To test your understanding of quantifiers, consider the following formulæ: *everybody loves somebody* vs *there is somebody that everybody loves*:

$$\forall x\, \exists y\, \text{loves}(x, y) \tag{1}$$
$$\exists y\, \forall x\, \text{loves}(x, y) \tag{2}$$

Does (1) imply (2)? Does (2) imply (1)? Consider both the informal meaning and the formal semantics defined above.

Exercise 12 Describe a formula that is true in precisely those domains that contain at least m elements. (We say it *characterises* those domains.) Describe a formula that characterises the domains containing at most m elements.

Exercise 13 Let $=$ be a 2-place predicate symbol, which we write using infix notation: for instance, $x = y$ rather than $=(x, y)$. Consider the following axioms:

$$\forall x\, x = x \tag{1}$$
$$\forall xy\, (x = y \to y = x) \tag{2}$$
$$\forall xyz\, (x = y \wedge y = z \to x = z) \tag{3}$$

Let the universe be the set of natural numbers, $N = \{0, 1, 2, \dots\}$. Which axioms hold if $I[=]$ is

- the empty relation, ϕ?
- the universal relation, $\{(x, y) \mid x, y \in N\}$?
- the relation $\{(x, x) \mid x \in N\}$?
- the relation $\{(x, y) \mid x, y \in N \wedge x + y \text{ is even}\}$?
- the relation $\{(x, y) \mid x, y \in N \wedge x + y = 100\}$?
- the relation $\{(x, y) \mid x, y \in N \wedge x \equiv y \pmod{16}\}$?

Exercise 14 Taking $=$ and R as 2-place relation symbols, consider the following axioms:

$$\forall x\, \neg R(x, x) \tag{1}$$
$$\forall xy\, \neg(R(x, y) \wedge R(y, x)) \tag{2}$$
$$\forall xyz\, (R(x, y) \wedge R(y, z) \to R(x, z)) \tag{3}$$
$$\forall xy\, (R(x, y) \vee (x = y) \vee R(y, x)) \tag{4}$$
$$\forall xz\, (R(x, z) \to \exists y\, (R(x, y) \wedge R(y, z))) \tag{5}$$

Exhibit two interpretations that satisfy axioms 1–3 and falsify axioms 4 and 5. Exhibit two interpretations that satisfy axioms 1–4 and falsify axiom 5. Exhibit two interpretations that satisfy axioms 1–5. Consider only interpretations that make $=$ denote the equality relation. (This exercise asks whether you can make the connection between the axioms and typical mathematical objects satisfying them.)

6 Formal Reasoning in First-Order Logic

This section reviews some syntactic notations: free variables versus bound variables and substitution. It lists some of the main equivalences for quantifiers. Finally it describes and illustrates the quantifier rules of the sequent calculus.

6.1 Free vs bound variables

The notion of bound variable occurs widely in mathematics: consider the role of x in $\int f(x)dx$ and the role of k in $\lim_{k=0}^{\infty} a_k$. Similar concepts occur in the λ-calculus. In first-order logic, variables are bound by quantifiers (rather than by λ).

Definition 9 An occurrence of a variable x in a formula is *bound* if it is contained within a subformula of the form $\forall x\ A$ or $\exists x\ A$.

An occurrence of the form $\forall x$ or $\exists x$ is called the *binding occurrence* of x.

An occurrence of a variable is *free* if it is not bound.

A *closed* formula is one that contains no free variables.

A *ground* term, formula or clause is one that contains no variables at all.

In $\forall x\ \exists y\ R(x, y, z)$, the variables x and y are bound while z is free.

In $(\exists x\ P(x)) \wedge Q(x)$, the occurrence of x just after P is bound, while that just after Q is free. Thus x has both free and bound occurrences. Such situations can be avoided by renaming bound variables. Renaming can also ensure that all bound variables in a formula are distinct.

Example 7 Renaming bound variables in a formula preserves its meaning, provided no name clashes are introduced. Consider the following renamings of $\forall x\ \exists y\ R(x, y, z)$:

$\forall u\ \exists y\ R(u, y, z)$	OK
$\forall x\ \exists w\ R(x, w, z)$	OK
$\forall u\ \exists y\ R(x, y, z)$	not done consistently
$\forall y\ \exists y\ R(y, y, z)$	clash with bound variable y
$\forall z\ \exists y\ R(z, y, z)$	clash with free variable z

6.2 Substitution

If A is a formula, t is a term, and x is a variable, then $A[t/x]$ is the formula obtained by substituting t for x throughout A. The substitution only affects the *free* occurrences of x. Pronounce $A[t/x]$ as 'A with t for x.' We also use $u[t/x]$ for substitution in a term u and $C[t/x]$ for substitution in a clause C.

Substitution is only sensible provided all bound variables in A are distinct from all variables in t. This can be achieved by renaming the bound variables in A. For example, if $\forall x\ A$ then $A[t/x]$ is true for all t; the formula holds when we drop the $\forall x$ and replace x by any term. But $\forall x\, \exists y\, x = y$ is true in all models, while $\exists y\, y+1 = y$ is not. We may not replace x by $y+1$, since the free occurrence of y in $y+1$ gets captured by the $\exists y$. First we must rename the bound y, getting say $\forall x\, \exists z\, x = z$; now we may replace x by $y+1$, getting $\exists z\, y+1 = z$. This formula is true in all models, regardless of the meaning of the symbols $+$ and 1.

6.3 Equivalences involving quantifiers

These equivalences are useful for transforming and simplifying quantified formulæ. Later, we shall use them to convert formulæ into *prenex normal form*, where all quantifiers are at the front.

pulling quantifiers through negation
(infinitary de Morgan laws)

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

pulling quantifiers through conjunction and disjunction
(provided x is not free in B)

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

distributive laws

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

implication: $A \to B$ as $\neg A \vee B$

(provided x is not free in B)

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

expansion: $\forall$ and $\exists$ as infinitary conjunction and disjunction

$$\forall x\ A \simeq (\forall x\ A) \wedge A[t/x]$$
$$\exists x\ A \simeq (\exists x\ A) \vee A[t/x]$$

With the help of the associative and commutative laws for $\wedge$ and $\vee$, a quantifier can be pulled out of any conjunct or disjunct.

The distributive laws differ from pulling: they replace two quantifiers by one. (Note that the quantified variables will probably have different names, so one of them will have be renamed.) Depending upon the situation, using distributive laws can be either better or worse than pulling. There are no distributive laws for $\forall$ over $\vee$ and $\exists$ over $\wedge$. If in doubt, do not use distributive laws!

Two substitution laws do not involve quantifiers explicitly, but let us use $x = t$ to replace x by t in a restricted context:

$$(x = t \wedge A) \simeq (x = t \wedge A[t/x])$$
$$(x = t \to A) \simeq (x = t \to A[t/x])$$

Many first-order formulæ have easy proofs using equivalences:

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

The following formula is quite hard to prove using the sequent calculus, but using equivalences it is simple:

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

If you are asked to prove a formula, but no particular formal system (such as the sequent calculus) has been specified, then you may use any convincing argument. Using equivalences as above can shorten the proof considerably. Also, take advantage of symmetries; in proving $A \wedge B \simeq B \wedge A$, it obviously suffices to prove $A \wedge B \models B \wedge A$.

Exercise 15 Verify these equivalences by appealing to the truth definition for first-order logic.

Exercise 16 Explain why the following are not equivalences. Are they implications? In which direction?

$$(\forall x\ A) \vee (\forall x\ B) \stackrel{?}{\simeq} \forall x\ (A \vee B)$$
$$(\exists x\ A) \wedge (\exists x\ B) \stackrel{?}{\simeq} \exists x\ (A \wedge B)$$

6.4 Sequent rules for the universal quantifier

Here are the sequent 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 $(\forall r)$ holds *provided* x is not free in the conclusion! This ensures that x is really arbitrary; if x is free in Γ or Δ then the sequent is assuming properties of x. To understand the proviso, contrast the proof of the theorem $\forall x\ [P(x) \rightarrow P(x)]$ with an attempted proof of the invalid formula $P(x) \rightarrow \forall x\ P(x)$.

Rule $(\forall l)$ lets us create many instances of $\forall x\ A$. The exercises below include some examples that require more than one copy of the quantified formula. Since we regard sequents as consisting of sets, we may regard them as containing unlimited quantities of each of their elements. But except for the two rules $(\forall l)$ and $(\exists r)$ (see below), we only need one copy of each formula.

Example 8 In this elementary proof, rule $(\forall l)$ is applied to instantiate the bound variable x with the term $f(y)$. The application of $(\forall r)$ is permitted because y is not free in the conclusion (which, in fact, is closed).

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

Example 9 This proof concerns part of the law for pulling universal quantifiers out of conjunctions. Rule $(\forall l)$ just discards the quantifier, since it instantiates the bound variable x with the free variable x.

$$\dfrac{\dfrac{\dfrac{\overline{A, B \Rightarrow A}}{A \wedge B \Rightarrow A}\ (\wedge l)}{\forall x\ (A \wedge B) \Rightarrow A}\ (\forall l)}{\forall x\ (A \wedge B) \Rightarrow \forall x\ A}\ (\forall r)$$

Example 10 The sequent $\forall x\ (A \to B) \Rightarrow A \to \forall x\ B$ is valid provided x is not free in A. That condition is required for the application of $(\forall r)$ below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \Rightarrow A, B} \quad \overline{A, B \Rightarrow B}}{A,\ A \to B \Rightarrow B}\ (\to l)}{A,\ \forall x\ (A \to B) \Rightarrow B}\ (\forall l)}{A,\ \forall x\ (A \to B) \Rightarrow \forall x\ B}\ (\forall r)}{\forall x\ (A \to B) \Rightarrow A \to \forall x\ B}\ (\to r)$$

What if the condition fails to hold? Let A and B both be the formula $x = 0$. Then $\forall x\ (x = 0 \to x = 0)$ is valid, but $x = 0 \to \forall x\ (x = 0)$ is not valid (it fails under any valuation that sets x to 0).

Exercise 17 Prove $\neg\forall y\ [(Q(a) \vee Q(b)) \wedge \neg Q(y)]$ using equivalences, and then formally using the sequent calculus.

Exercise 18 Prove the following using the sequent calculus:

$$\forall x\ [P(x) \to P(f(x))] \Rightarrow \forall x\ [P(x) \to P(f(f(x)))]$$
$$(\forall x\ A) \wedge (\forall x\ B) \Rightarrow \forall x\ (A \wedge B)$$
$$\forall x\ (A \wedge B) \Rightarrow (\forall x\ A) \wedge (\forall x\ B)$$

6.5 Sequent rules for the existential quantifier

Here are the sequent rules for $\exists$:

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

Rule $(\exists l)$ holds *provided* x is not free in the conclusion! These rules are strictly dual to the $\forall$-rules; any example involving $\forall$ can easily be transformed into one involving $\exists$ and having a proof of precisely the same form. For example, the sequent $\forall x\ P(x) \Rightarrow \forall y\ P(f(y))$ can be transformed into $\exists y\ P(f(y)) \Rightarrow \exists x\ P(x)$.

Example 11 Here is half of the $\exists$ distributive law. Rule $(\exists r)$ just discards the quantifier, instantiating the bound variable x with the free variable x. In the general case, it can instantiate the bound variable with any term.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \Rightarrow A, B}}{A \Rightarrow A \vee B}\ (\vee r)}{A \Rightarrow \exists x\ (A \vee B)}\ (\exists r)}{\exists x\ A \Rightarrow \exists x\ (A \vee B)}\ (\exists l) \qquad \dfrac{\text{similar}}{\exists x\ B \Rightarrow \exists x\ (A \vee B)}\ (\exists l)}{\exists x\ A \vee \exists x\ B \Rightarrow \exists x\ (A \vee B)}\ (\vee l)$$

Example 12 The sequent $\exists x\ A \wedge \exists x\ B \Rightarrow \exists x\ (A \wedge B)$ is not valid, because the value of x that makes A true could differ from the value of x that makes B true. This comes out clearly in the proof attempt, where we are not allows to apply $(\exists l)$ twice with the same variable name, x. As soon as we are forced to rename the second variable to y, it becomes obvious that the two values could differ. Turning to the right side of the sequent, no application of $(\exists r)$ can lead to a proof. We have nothing to instantiate x with:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A,\ B[y/x] \Rightarrow A \wedge B}{A,\ B[y/x] \Rightarrow \exists x\ (A \wedge B)}\ (\exists r)}{A,\ \exists x\ B \Rightarrow \exists x\ (A \wedge B)}\ (\exists l)}{\exists x\ A,\ \exists x\ B \Rightarrow \exists x\ (A \wedge B)}\ (\exists l)}{\exists x\ A \wedge \exists x\ B \Rightarrow \exists x\ (A \wedge B)}\ (\wedge l)$$

Exercise 19 Prove the following using the sequent calculus:

$$\begin{aligned} P(a) \vee \exists x\ P(f(x)) &\Rightarrow \exists y\ P(y) \\ \exists x\ (A \vee B) &\Rightarrow (\exists x\ A) \vee (\exists x\ B) \\ &\Rightarrow \exists z\ (P(z) \rightarrow\ P(a) \wedge P(b)) \end{aligned}$$

7 Clause Methods for Propositional Logic

This section discusses two proof methods in the context of propositional logic: the Davis-Putnam procedure and resolution.

The Davis-Putnam procedure dates from 1960, and its application to first-order logic has been regarded as obsolete for decades. However, propositional logic has grown in importance, and Davis-Putnam has been rediscovered as an efficient decision procedure. It has been applied to solve some open questions in combinatorial mathematics, such as the existence of certain Latin squares. Its main rival is OBDDs, which have been applied mainly to hardware design.

Resolution is a powerful proof method for first-order logic. We first consider *ground* resolution, which works for propositional logic. Though of little practical use, ground resolution introduces some of the main concepts. The resolution method is not natural for hand proofs, but it is easy to automate: it has only one inference rule and no axioms.

Both methods require the original formula to be negated, then converted into CNF. Recall that CNF is a conjunction of disjunction of literals. A disjunction of literals is called a *clause*, and written as a set of literals. Converting the negated formula to CNF yields a set of such clauses. Both methods seek a contradiction in

the set of clauses; if the clauses are unsatisfiable, then so is the negated formula, and therefore the original formula is valid.

To *refute* a set of clauses is to prove that they are inconsistent. The proof is called a *refutation*.

7.1 Clausal notation

Definition 10 A *clause* is a disjunction of literals

$$\neg K_1 \vee \cdots \vee \neg K_m \vee L_1 \vee \cdots \vee L_n,$$

written as a set

$$\{\neg K_1, \ldots, \neg K_m, L_1, \ldots, L_n\}.$$

Since $\vee$ is commutative, associative, and idempotent, the order of literals in a clause does not matter. The above clause is logically equivalent to the implication

$$(K_1 \wedge \cdots \wedge K_m) \rightarrow (L_1 \vee \cdots \vee L_n)$$

Kowalski notation abbreviates this to

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

and when $n = 1$ we have the familiar Prolog clauses, also known as *definite clauses*.

7.2 The Davis-Putnam Method

The Davis-Putnam method is based upon some obvious identities:

$$\begin{aligned} \mathbf{t} \wedge A &\simeq A \\ A \wedge (A \vee B) &\simeq A \\ A \wedge (\neg A \vee B) &\simeq A \wedge B \\ A &\simeq (A \wedge B) \vee (A \wedge \neg B) \end{aligned}$$

Here is an outline of the algorithm:

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*. A literal L is *pure* if there is no clause containing $\neg L$.

4. If the empty clause is generated, then we have a refutation. Conversely, if all clauses are deleted, then the original clause set is satisfiable.

5. Perform a *case split* on some literal L, and recursively apply the algorithm to the L and $\neg L$ subcases. The clause set is satisfiable if and only if one of the subcases is satisfiable.

This is a decision procedure. It must terminate because each case split eliminates a propositional symbol. Zhang and Stickel (1994) have proposed some efficient algorithms for the Davis-Putnam procedure.

Tautological clauses are deleted because they are always true, and thus cannot participate in a contradiction. A pure literal can always be assumed to be true; deleting the clauses containing it can be regarded as a degenerate case split, in which there is only one case.

Example 13 The Davis-Putnam method can show that a formula is not a theorem. Consider the formula $P \vee Q \rightarrow Q \vee R$. After negating this and converting to CNF, we obtain the three clauses $\{P, Q\}$, $\{\neg Q\}$ and $\{\neg R\}$. The Davis-Putnam method terminates rapidly:

$\{P, Q\}$	$\{\neg Q\}$	$\{\neg R\}$	initial clauses
$\{P\}$		$\{\neg R\}$	unit $\neg Q$
		$\{\neg R\}$	unit P (also pure)
			unit $\neg R$ (also pure)

The clauses are satisfiable by $P \mapsto \mathbf{t},\ Q \mapsto \mathbf{f},\ R \mapsto \mathbf{f}$. This interpretation falsifies $P \vee Q \rightarrow Q \vee R$.

Example 14 Here is an example of a case split. Consider the clause set

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

There are no unit clauses or pure literals, so we arbitrarily select P for case splitting:

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

Exercise 20 Apply the Davis-Putnam procedure to the clause set

$$\{P, Q\} \qquad \{\neg P, Q\} \qquad \{P, \neg Q\} \qquad \{\neg P, \neg Q\}.$$

7.3 Introduction to resolution

Resolution is essentially the following rule of inference:

$$\frac{B \vee A \quad \neg B \vee C}{A \vee C}$$

To convince yourself that this rule is sound, note that B must be either false or true.

- if B is false, then $B \vee A$ is equivalent to A, so we get $A \vee C$
- if B is true, then $\neg B \vee C$ is equivalent to C, so we get $A \vee C$

You might also understand this rule via transitivity of $\rightarrow$ (with $D = \neg A$):

$$\frac{D \rightarrow B \quad B \rightarrow C}{D \rightarrow C}$$

A special case of resolution is when A and C are empty:

$$\frac{B \quad \neg B}{\mathbf{f}}$$

This detects contradictions.

Resolution works with disjunctions. The aim is to prove a contradiction, refuting a formula. Here is the method for proving a formula A:

1. Translate $\neg A$ into CNF as $A_1 \wedge \cdots \wedge A_m$.
2. Break this into a set of clauses: $A_1, \ldots, A_m$.
3. Repeatedly apply the resolution rule to the clauses, producing new clauses. These are all consequences of $\neg A$.
4. If a contradiction is reached, we have refuted $\neg A$.

In set notation the resolution rule is

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

Resolution takes two clauses and creates a new one. A collection of clauses is maintained; the two clauses are chosen from the collection according to some

strategy, and the new clause is added to it. If $m = 0$ or $n = 0$ then the new clause will be smaller than one of the parent clauses; if $m = n = 0$ then the new clause will be empty. A clause is true (in some interpretation) just when one of the literals is true; thus the empty clause indicates contradiction. It is written $\Box$. If the empty clause is generated, resolution terminates successfully.

7.4 Examples of ground resolution

Let us try to prove

$$P \wedge Q \rightarrow Q \wedge P$$

Convert its negation to CNF:

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

We can combine steps 1 (eliminate $\rightarrow$) and 2 (push negations in) using the law $\neg(A \rightarrow B) \simeq A \wedge \neg B$:

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

Step 3, push disjunctions in, has nothing to do. The clauses are

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

We resolve $\{P\}$ and $\{\neg Q, \neg P\}$ as follows:

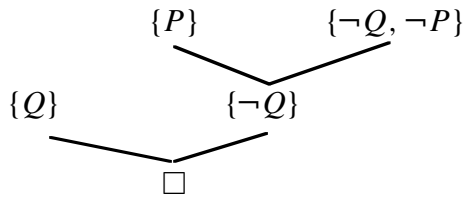
$$\frac{\{P\} \quad \{\neg P, \neg Q\}}{\{\neg Q\}}$$

The resolvent is $\{\neg Q\}$. Resolving $\{Q\}$ and with this new clause gives

$$\frac{\{Q\} \quad \{\neg Q\}}{\{\}}$$

The resolvent is the empty clause, properly written as $\Box$. We have proved $P \wedge Q \rightarrow Q \wedge P$ by assuming its negation and deriving a contradiction.

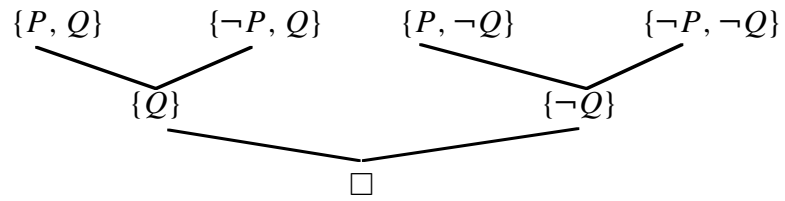
It is nicer to draw a tree like this:



Another example is $(P \leftrightarrow Q) \leftrightarrow (Q \leftrightarrow P)$. The steps of the conversion to clauses is left as an exercise; remember to negate the formula first! The final clauses are

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

A tree for the resolution proof is



Note that the tree contains $\{Q\}$ and $\{\neg Q\}$ rather than $\{Q, Q\}$ and $\{\neg Q, \neg Q\}$. If we forget to suppress repeated literals, we can get stuck. Resolving $\{Q, Q\}$ and $\{\neg Q, \neg Q\}$ (keeping repetitions) gives $\{Q, \neg Q\}$, a tautology. Tautologies are useless. Resolving this one with the other clauses leads nowhere. Try it.

These examples might suggest that a proof has to involve all the clauses, each used exactly once. In general, a clause may be used repeatedly. Problems often contain irrelevant clauses that are not used at all.

Exercise 21 Prove $(A \to B \vee C) \to [(A \to B) \vee (A \to C)]$ using resolution.

7.5 A proof using a set of assumptions

In this example we assume

$$H \to M \vee N \qquad M \to K \wedge P \qquad N \to L \wedge P$$

and prove $H \to P$. It turns out that we can generate clauses separately from the assumptions (taken *positively*) and the conclusion (*negated*).

If we call the assumptions $A_1, \ldots, A_k$ and the conclusion B, then the desired theorem is

$$(A_1 \wedge \cdots \wedge A_k) \to B$$

Try negating this and converting to CNF. Using the law $\neg(A \to B) \simeq A \wedge \neg B$, the negation converts in one step to

$$A_1 \wedge \cdots \wedge A_k \wedge \neg B$$

Since the entire formula is a conjunction, we can separately convert $A_1, \ldots, A_k$, and $\neg B$ to clause form and pool the clauses together.

Assumption $H \to M \vee N$ is essentially in clause form already:

$$\{\neg H, M, N\}$$

Assumption $M \to K \wedge P$ becomes two clauses:

$$\{\neg M, K\} \qquad \{\neg M, P\}$$

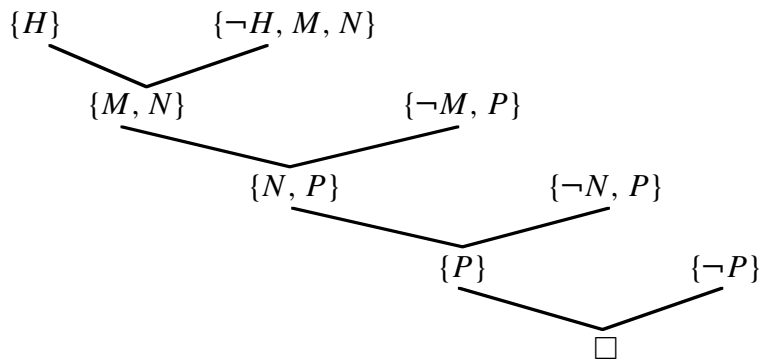
Assumption $N \to L \wedge P$ also becomes two clauses:

$$\{\neg N, L\} \qquad \{\neg N, P\}$$

The negated conclusion, $\neg(H \to P)$, becomes two clauses:

$$\{H\} \qquad \{\neg P\}$$

A tree for the resolution proof is



The clauses were not tried at random. Here are some points of proof strategy.

Ignoring irrelevance. Clauses $\{\neg M, K\}$ and $\{\neg N, L\}$ lead nowhere, so they were not tried. Resolving with one of these would make a clause containing K or L. There is no way of getting rid of either literal, for no clause contains $\neg K$ or $\neg L$. So this is not a way to obtain the empty clause.

Working from the goal. In each resolution step, at least one clause involves the negated conclusion (possibly via earlier resolution steps). We do not blindly derive facts from the assumptions — for, provided the assumptions are consistent, any contradiction will have to involve the negated conclusion. This strategy is called *set of support*.

Linear resolution. The proof has a linear structure: each resolvent becomes the parent clause for the next resolution step. Furthermore, the other parent clause is always one of the original set of clauses. This simple structure is very efficient because only the last resolvent needs to be saved. It is similar to the execution strategy of Prolog.

Exercise 22 Explain in more detail the conversion of this example into clauses.

Exercise 23 Prove Peirce's law, $((P \to Q) \to P) \to P$, using resolution.

Exercise 24 Prove $(Q \to R) \wedge (R \to P \wedge Q) \wedge (P \to Q \vee R) \to (P \leftrightarrow Q)$ using the Davis-Putnam method, and again by resolution. (This one is fairly long.)

7.6 Deletion of redundant clauses

During resolution, the number of clauses builds up dramatically; it is important to delete all redundant clauses.

Each new clause is a consequence of the existing clauses. A contradiction can only be derived if the original set of clauses is inconsistent. A clause can be deleted if it does not affect the consistency of the set. Any tautology should be deleted, since it is true in all interpretations.

Here is a subtler case. Consider the clauses

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

Resolving the first two yields $\{P, R\}$. Since each clause is a disjunction, any interpretation that satisfies $\{P, R\}$ also satisfies $\{P, Q, R\}$. Thus $\{P, Q, R\}$ cannot cause inconsistency, and should be deleted.

Put another way, $P \vee R$ implies $P \vee Q \vee R$. Anything that could be derived from $P \vee Q \vee R$ could also be derived from $P \vee R$. This sort of deletion is called *subsumption*; clause $\{P, R\}$ *subsumes* $\{P, Q, R\}$.

Exercise 25 Prove $(P \wedge Q \to R) \wedge (P \vee Q \vee R) \to ((P \leftrightarrow Q) \to R)$ by resolution. Show the steps of converting the formula into clauses.

Exercise 26 Using linear resolution, prove that $(P \wedge Q) \to (R \wedge S)$ follows from $P \to R$ and $R \wedge P \to S$.

Exercise 27 Convert these axioms to clauses, showing all steps. Then prove *Winterstorm* $\rightarrow$ *Miserable* by resolution:

$$\begin{array}{cc} \mathit{Rain} \wedge (\mathit{Windy} \vee \neg \mathit{Umbrella}) \rightarrow \mathit{Wet} & \mathit{Winterstorm} \rightarrow \mathit{Storm} \wedge \mathit{Cold} \\ \mathit{Wet} \wedge \mathit{Cold} \rightarrow \mathit{Miserable} & \mathit{Storm} \rightarrow \mathit{Rain} \wedge \mathit{Windy} \end{array}$$

8 Skolem Functions and Herbrand's Theorem

Propositional logic is the basis of many proof methods for first-order logic. Eliminating the quantifiers from a first-order formula reduces it nearly to propositional logic. This section describes how to do so.

8.1 Prenex normal form

The simplest method of eliminating quantifiers from formula involves first moving them to the front.

Definition 11 A formula is in *prenex normal form* if and only if it has the form

$$\underbrace{Q_1 x_1 \; Q_2 x_2 \cdots Q_n x_n}_{\text{prefix}} \; \underbrace{(A)}_{\text{matrix}},$$

where A is quantifier-free, each Q_i is either $\forall$ or $\exists$, and $n \geq 0$. The string of quantifiers is called the *prefix* and A is called the *matrix*.

Using the equivalences described above, any formula can be put into prenex normal form.

Examples of translation.

The affected subformulæ will be underlined.

Example 15 Start with

$$\neg(\underline{\exists x \; P(x)}) \wedge (\exists y \; Q(y) \vee \forall z \; P(z))$$

Pull out the $\exists x$:

$$\forall x \; \neg P(x) \wedge (\underline{\exists y \; Q(y)} \vee \forall z \; P(z))$$

Pull out the $\exists y$:

$$\forall x \; \neg P(x) \wedge (\underline{\exists y \; (Q(y) \vee \forall z \; P(z))})$$

Pull out the $\exists y$ again:

$$\exists y\, (\forall x\, \neg P(x) \wedge (Q(y) \vee \underline{\forall z\, P(z)}))$$

Pull out the $\forall z$:

$$\exists y\, (\forall x\, \neg P(x) \wedge \underline{\forall z\, (Q(y) \vee P(z))})$$

Pull out the $\forall z$ again:

$$\exists y\, \forall z\, (\underline{\forall x\, \neg P(x)} \wedge (Q(y) \vee P(z)))$$

Pull out the $\forall x$:

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

Example 16 Start with

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

Remove the implication:

$$\neg \forall x\ P(x) \vee \exists y\, \forall z\, R(y, z)$$

Pull out the $\forall x$:

$$\exists x\, \neg P(x) \vee \exists y\, \forall z\, R(y, z)$$

Distribute $\exists$ over $\vee$, renaming y to x:[5]

$$\exists x\, (\neg P(x) \vee \forall z\, R(x, z))$$

Finally, pull out the $\forall z$:

$$\exists x\, \forall z\, (\neg P(x) \vee R(x, z))$$

8.2 Removing quantifiers: Skolem form

Now that the quantifiers are at the front, let's eliminate them! We replace every existentially bound variable by a Skolem constant or function. This transformation does not preserve the meaning of a formula; it does preserve *inconsistency*, which is the critical property, since resolution works by detecting contradictions.

[5] Or simply pull out the quantifiers separately. Using the distributive law is marginally better here because it will result in only one Skolem constant instead of two; see the following section.

How to Skolemize a formula

Suppose the formula is in prenex normal form.[6] Starting from the left, if the formula contains an existential quantifier, then it must have the form

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

where A is a prenex formula, $k \geq 0$, and $\exists y$ is the leftmost existential quantifier. Choose a k-place function symbol not present in A (that is, a *new* function symbol). Delete the $\exists y$ and replace all other occurrences of y by $f(x_1, x_2, \ldots, x_k)$. The result is another prenex formula:

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

If $k = 0$ above then the prenex formula is simply $\exists y \, A$, and other occurrences of y are replaced by a new constant symbol c. The resulting formula is $A[c/y]$.

The remaining existential quantifiers, if any, are in A. Repeatedly eliminate all of them, as above. The new symbols are called *Skolem functions* (or Skolem constants).

After Skolemization the formula is just $\forall x_1 \, \forall x_2 \, \cdots \forall x_k \, A$ where A is quantifier-free. Since the free variables in a formula are taken to be universally quantified, we can drop these quantifiers, leaving simply A. We are almost back to the propositional case, except the formula typically contains terms. We shall have to handle constants, function symbols, and variables.

Examples of Skolemization

The affected expressions are underlined.

Example 17 Start with

$$\underline{\exists x} \, \forall y \, \exists z \, R(\underline{x}, y, z)$$

Eliminate the $\exists x$ using the Skolem constant a:

$$\forall y \, \underline{\exists z} \, R(a, y, \underline{z})$$

Eliminate the $\exists z$ using the 1-place Skolem function f:

$$\forall y \, R(a, y, f(y))$$

Finally, drop the $\forall y$ and convert the remaining formula to a clause:

$$\{R(a, y, f(y))\}$$

[6] This makes things easier to follow. However, some proof methods merely require the formula to be in negation normal form. The basic idea is the same: remove the outermost existential quantifier, replacing its bound variable by a Skolem term. Pushing quantifiers in as far as possible, instead of pulling them out, yields a better set of clauses.

Example 18 Start with

$$\underline{\exists u}\,\forall v\,\exists w\,\exists x\,\forall y\,\exists z\,((P(h(\underline{u}, v)) \vee Q(w)) \wedge R(x, h(y, z)))$$

Eliminate the $\exists u$ using the Skolem constant c:

$$\forall v\,\underline{\exists w}\,\exists x\,\forall y\,\exists z\,((P(h(c, v)) \vee Q(\underline{w})) \wedge R(x, h(y, z)))$$

Eliminate the $\exists w$ using the 1-place Skolem function f:

$$\forall v\,\underline{\exists x}\,\forall y\,\exists z\,((P(h(c, v)) \vee Q(f(v))) \wedge R(\underline{x}, h(y, z)))$$

Eliminate the $\exists x$ using the 1-place Skolem function g:

$$\forall v\,\forall y\,\underline{\exists z}\,((P(h(c, v)) \vee Q(f(v))) \wedge R(g(v), h(y, \underline{z})))$$

Eliminate the $\exists z$ using the 2-place Skolem function j (note that function h is already used!):

$$\forall v\,\forall y\,((P(h(c, v)) \vee Q(f(v))) \wedge R(g(v), h(y, j(v, y))))$$

Finally drop the universal quantifiers, getting a set of clauses:

$$\{P(h(c, v)), Q(f(v))\} \qquad \{R(g(v), h(y, j(v, y)))\}$$

Correctness of Skolemization

Skolemization does *not* preserve meaning. The version presented above does not even preserve validity! For example,

$$\exists x\,(P(a) \rightarrow P(x))$$

is valid. (Why? In any model, the required value of x exists — it is just the value of a in that model.)

Replacing the $\exists x$ by the Skolem constant b gives

$$P(a) \rightarrow P(b)$$

This has a different meaning since it refers to a constant b not previously mentioned. And it is not valid! For example, it is false in the interpretation where $P(x)$ means 'x equals 0' and a denotes 0 and b denotes 1.

Our version of Skolemization does preserve *consistency* — and therefore inconsistency. Consider one Skolemization step.

- The formula $\forall x\, \exists y\, A$ is consistent iff it holds in some interpretation $\mathcal{I} = (D, I)$
- iff for all $x \in D$ there is some $y \in D$ such that A holds
- iff there is some function on D, say $\hat{f} \in D \to D$, such that for all $x \in D$, if $y = \hat{f}(x)$ then A holds
- iff there is an interpretation $\mathcal{I}'$ extending $\mathcal{I}$ so that the symbol f denotes the function $\hat{f}$, and $A[f(x)/y]$ holds for all $x \in D$.
- iff the formula $\forall x\, A[f(x)/y]$ is consistent.

Note that $\mathcal{I}$ above does not interpret f because Skolem functions have to be new. Thus $\mathcal{I}$ may be extended to $\mathcal{I}'$ by giving an interpretation for f.

This argument easily generalizes to the case $\forall x_1\, \forall x_2\, \cdots \forall x_k\, \exists y\, A$. Thus, if a formula is consistent then so is the Skolemized version. If it is inconsistent then so is the Skolemized version. That is what we need: resolution works by proving that a formula is inconsistent.

There is a dual version of Skolemization that preserves validity rather than consistency. It replaces universal quantifiers, rather than existential ones, by Skolem functions.

Exercise 28 Describe this dual version of Skolemization and demonstrate that it preserves validity. What might it be used for?

8.3 Herbrand interpretations

A Herbrand interpretation basically consists of all terms that can be written using just the constant and function symbols in a set of clauses S (or quantifier-free formula). Why define Herbrand interpretations? A mathematical reason: for consistency of S we need only consider Herbrand interpretations. A programming reason: the data processed by a Prolog program S is simply its Herbrand universe.

To define the Herbrand universe for the set of clauses S we start with sets of the constant and function symbols in S, *including Skolem functions*.

Definition 12 Let $\mathcal{C}$ be the set of all constants in S. If there are none, let $\mathcal{C} = \{a\}$ for some constant symbol a of the first-order language.[7] For $n > 0$ let $\mathcal{F}_n$ be the set of all n-place function symbols in S and let $\mathcal{P}_n$ be the set of all n-place predicate symbols in S.

[7] This step ensures that the resulting domain will be non-empty.

The *Herbrand universe* is the set $H = \bigcup_{i \geq 0} H_i$, where

$$\begin{aligned} H_0 &= \mathcal{C} \\ H_{i+1} &= H_i \cup \{f(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in H_i \text{ and } f \in \mathcal{F}_n\} \end{aligned}$$

By this construction, H turns out to satisfy the recursive equation

$$H = \{f(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in H \text{ and } f \in \mathcal{F}_n\}$$

The *Herbrand base* (or atom set) consists of all possible applications of predicate symbols in S to terms of the Herbrand universe for S:

$$HB = \{P(t_1, \ldots, t_n) \mid t_1, \ldots, t_n \in H \text{ and } P \in \mathcal{P}_n\}$$

Note that the elements of H are ground terms — this is a syntactic interpretation. An interpretation (H, I) is a *Herbrand interpretation* provided $I[t] = t$ for all ground terms t. The interpretation of the predicates is given by a subset of the Herbrand base; this specifies which predicate applications are true.

Example 19 Suppose we have the set (consisting of two clauses)

$$S = \{\{P(a)\}, \quad \{Q(g(y, z)), \neg P(f(x))\}\}$$

Then

$$\begin{aligned} \mathcal{C} &= \{a\} \\ \mathcal{F}_1 &= \{f\} \\ \mathcal{F}_2 &= \{g\} \\ \mathcal{F}_n &= \emptyset \qquad (n > 2) \\ H &= \{a, f(a), g(a, a), f(f(a)), g(f(a), a), g(a, f(a)), g(f(a), f(a)), \ldots\} \\ HB &= \{P(a), Q(a), P(f(a)), Q(f(a)), P(g(a, a)), Q(g(a, a)), \\ &\quad P(f(f(a))), Q(f(f(a))), P(g(f(a), a)), Q(g(f(a), a)), \\ &\quad P(g(a, f(a))), Q(g(a, f(a))), P(g(f(a), f(a))), Q(g(f(a), f(a))), \ldots\} \end{aligned}$$

Every interpretation $\mathcal{I}$ over an arbitrary universe can be mimicked by some Herbrand interpretation: just take

$$\{P(t_1, \ldots, t_n) \in HB \mid P(t_1, \ldots, t_n) \text{ holds in } \mathcal{I}\}$$

This is a subset of HB. Each subset of HB specifies a Herbrand interpretation by listing the values (in H) for which each predicate holds. To mimic the interpretation $\mathcal{I}$ we take exactly the set of ground atomic formulæ that hold in $\mathcal{I}$; this is a Herbrand interpretation.

Thus, we have informally proved the following two results (Chang and Lee, page 55):

Lemma 13 *Let S be a set of clauses. If an interpretation satisfies S, then an* Herbrand *interpretation satisfies S.*

Theorem 14 *A set S of clauses is unsatisfiable if and only no Herbrand interpretation satisfies S.*

Equality may behave strangely in Herbrand interpretations. Given an interpretation $\mathcal{I}$, the denotation of $=$ is the set of all pairs of ground terms (t_1, t_2) such that $t_1 = t_2$ according to $\mathcal{I}$. In a context of the natural numbers, the denotation of $=$ could include pairs like $(1 + 1, 2)$ — the two components need not be identical, contrary to the normal situation with equality.

8.4 The Skolem-Gödel-Herbrand Theorem

Finally we have the Skolem-Gödel-Herbrand Theorem. A version of the Completeness Theorem, it tells us that unsatisfiability can always be detected by a *finite* process. It does not tell us how to detect satisfiability, for there is no general method.[8]

Definition 15 An *instance* of a clause C is a clause that results by replacing variables in C by terms. A *ground instance* of a clause C is an instance of C that contains no variables. (It can be obtained by replacing all variables in C by elements of a Herbrand universe, which are ground terms.)

Since the variables in a clause are taken to be universally quantified, every instance of C is a logical consequence of C.

Example 20 This clause is valid in the obvious integer model:

$$C = \{\neg\text{even}(x), \neg\text{even}(y), \text{even}(x + y)\}$$

Replacing x by $y + y$ in C results in the instance

$$C' = \{\neg\text{even}(y + y), \neg\text{even}(y), \text{even}((y + y) + y)\}$$

Replacing y by 2 in C' results in the ground instance

$$C'' = \{\neg\text{even}(2 + 2), \neg\text{even}(2), \text{even}((2 + 2) + 2)\}$$

[8] It is often confused with Herbrand's Theorem, a stronger result.

Example 21 Consider the clause

$$C = \{Q(g(y, x)), \neg P(f(x))\}$$

Replacing x by $f(z)$ in C results in the instance

$$C' = \{Q(g(y, f(z))), \neg P(f(f(z)))\}$$

Replacing y by $j(a)$ and z by b in C' results in the instance

$$C'' = \{Q(g(j(a), f(b))), \neg P(f(f(b)))\}$$

Assuming that a and b are constants, C'' is a ground instance of C.

Theorem 16 *A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.*

The proof is rather involved; see Chang and Lee, pages 56–61, for details. The ($\Longrightarrow$) direction is the interesting one. It uses a non-constructive argument to show that if there is no finite unsatisfiable set S', then there must be a model of S.

The ($\Longleftarrow$) direction simply says that if S' is unsatisfiable then so is S. This is straightforward since every clause in S' is a logical consequence of some clause in S. Thus if S' is inconsistent, the inconsistency is already present in S.

Question: how do we discover *which* ground instances? Answer: by *unification*.

Exercise 29 Consider a first-order language with 0 and 1 as constant symbols, with $-$ as a 1-place function symbol and $+$ as a 2-place function symbol, and with $<$ as a 2-place predicate symbol.

(a) Describe the Herbrand Universe for this language.

(b) The language can be interpreted by taking the integers for the universe and giving 0 1, $-$, $+$, and $<$ their usual meanings over the integers. What do those symbols denote in the corresponding Herbrand model?

9 Unification

Unification is the operation of finding a common instance of two terms. Though the concept is simple, it involves a complicated theory. Proving the unification algorithm's correctness (especially termination) is difficult.

To introduce the idea of unification, consider a few examples. The terms $f(x, b)$ and $f(a, y)$ have the common instance $f(a, b)$, replacing x by a and y

by b. The terms $f(x, x)$ and $f(a, b)$ have no common instance, assuming that a and b are distinct constants. The terms $f(x, x)$ and $f(y, g(y))$ have no common instance, since there is no way that x can have the form y and $g(y)$ at the same time — unless we admit the infinite term $g(g(g(\cdots)))$.

Only variables may be replaced by other terms. Constants are not affected (they remain constant!). If a term has the form $f(t, u)$ then instances of that term must have the form $f(t', u')$, where t' is an instance of t and u' is an instance of u.

9.1 Substitutions

We have already seen substitutions informally. It is now time for a more detailed treatment.

Definition 17 A *substitution* is a finite set of replacements

$$[t_1/x_1, \ldots, t_k/x_k]$$

where $x_1, \ldots, x_k$ are distinct variables such that $t_i \neq x_i$ for all $i = 1, \ldots, k$. We use Greek letters ϕ, θ, σ to stand for substitutions.

The finite set $\{x_1, \ldots, x_k\}$ is called the *domain* of the substitution. The domain of a substitution θ is written $\mathrm{dom}(\theta)$.

A substitution $\theta = [t_1/x_1, \ldots, t_k/x_k]$ defines a function from the variables $\{x_1, \ldots, x_k\}$ to terms. Postfix notation is usual for applying a substitution; thus, for example, $x_i\theta = t_i$. Substitutions may be applied to terms, not just to variables. Substitution on terms is defined recursively as follows:

$$\begin{aligned} f(t_1, \ldots, t_n)\theta &= f(t_1\theta, \ldots, t_n\theta) \\ x\theta &= x \qquad \text{for all } x \notin \mathrm{dom}(\theta) \end{aligned}$$

Here f is an n-place function symbol. The operation substitutes in the arguments of functions, and leaves unchanged any variables outside of the domain of θ.

Substitution may be extended to literals and clauses as follows:

$$\begin{aligned} P(t_1, \ldots, t_n)\theta &= P(t_1\theta, \ldots, t_n\theta) \\ \{L_1, \ldots, L_m\}\theta &= \{L_1\theta, \ldots, L_m\theta\} \end{aligned}$$

Here P is an n-place predicate symbol (or its negation), while $L_1, \ldots, L_m$ are the literals in a clause.

Example 22 The substitution $\theta = [h(y)/x, b/y]$ says to replace x by $h(y)$ and y by b. The replacements occur simultaneously; it does *not* have the effect of replacing x by $h(b)$. Its domain is $\mathrm{dom}(\theta) = \{x, y\}$. Applying this substitution gives

$$f(x, g(u), y)\theta = f(h(y), g(u), b)$$
$$R(h(x), z)\theta = R(h(h(y)), z)$$
$$\{P(x), \neg Q(y))\}\theta = \{P(h(y)), \neg Q(b))\}$$

9.2 Composition of substitutions

If ϕ and θ are substitutions then so is their *composition* $\phi \circ \theta$, which satisfies

$$t(\phi \circ \theta) = (t\phi)\theta \qquad \text{for all terms } t$$

Can we write $\phi \circ \theta$ as a set of replacements? It has to satisfy the above for all relevant variables:

$$x(\phi \circ \theta) = (x\phi)\theta \qquad \text{for all } x \in \mathrm{dom}(\phi) \cup \mathrm{dom}(\theta)$$

Thus it must be the set consisting of the replacements

$$(x\phi)\theta \ / \ x \qquad \text{for all } x \in \mathrm{dom}(\phi) \cup \mathrm{dom}(\theta)$$

Equality of substitutions ϕ and θ is defined as follows: $\phi = \theta$ if $x\phi = x\theta$ for all variables x. Under these definitions composition enjoys an associative law. It also has an identity element, namely [], the empty substitution.

$$(\phi \circ \theta) \circ \sigma = \phi \circ (\theta \circ \sigma)$$
$$\phi \circ [] = \phi$$
$$[] \circ \phi = \phi$$

Example 23 Let $\phi = [j(x)/u, 0/y]$ and $\theta = [h(z)/x, g(3)/y]$. Then $\mathrm{dom}(\phi) = \{u, y\}$ and $\mathrm{dom}(\theta) = \{x, y\}$, so $\mathrm{dom}(\phi) \cup \mathrm{dom}(\theta) = \{u, x, y\}$. Thus

$$\phi \circ \theta = [j(h(z))/u, h(z)/x, 0/y]$$

Notice that $y(\phi \circ \theta) = (y\phi)\theta = 0\theta = 0$; the replacement $g(3)/y$ has disappeared.

Exercise 30 Verify that $\circ$ is associative and has [] for an identity.

9.3 Unifiers

Definition 18 A substitution θ is a *unifier* of terms t_1 and t_2 if $t_1\theta = t_2\theta$. More generally, θ is a unifier of terms $t_1, t_2, \ldots, t_m$ if $t_1\theta = t_2\theta = \cdots = t_m\theta$. The term $t_1\theta$ is called the *common instance* of the unified terms. A unifier of two or more literals is defined similarly.

Two terms can only be unified if they have similar structure apart from variables. The terms $f(x)$ and $h(y, z)$ are clearly non-unifiable since no substitution can do anything about the differing function symbols. It is easy to see that θ unifies $f(t_1, \ldots, t_n)$ and $f(u_1, \ldots, u_n)$ if and only if θ unifies t_i and u_i for all $i = 1, \ldots, n$.

Example 24 The substitution $[3/x, g(3)/y]$ unifies the terms $g(g(x))$ and $g(y)$. The common instance is $g(g(3))$. These terms have many other unifiers, including the following:

unifying substitution	common instance
$[f(u)/x, g(f(u))/y]$	$g(g(f(u)))$
$[z/x, g(z)/y]$	$g(g(z))$
$[g(x)/y]$	$g(g(x))$

Note that $g(g(3))$ and $g(g(f(u)))$ are instances of $g(g(x))$. Thus $g(g(x))$ is more general than $g(g(3))$ and $g(g(f(u)))$; it admits many other instances. Certainly $g(g(3))$ seems to be arbitrary — neither of the original terms mentions 3! A separate point worth noting is that $g(g(x))$ is equivalent to $g(g(z))$, apart from the name of the variable. Let us formalize these intuitions.

9.4 Most general unifiers

Definition 19 The substitution θ is *more general* than ϕ if $\phi = \theta \circ \sigma$ for some substitution σ.

Example 25 Recall the unifiers of $g(g(x))$ and $g(y)$. The unifier $[g(x)/y]$ is more general than the others listed, for

$$\begin{aligned}
[3/x, g(3)/y] &= [g(x)/y] \circ [3/x] \\
[f(u)/x, g(f(u))/y] &= [g(x)/y] \circ [f(u)/x] \\
[z/x, g(z)/y] &= [g(x)/y] \circ [z/x] \\
[g(x)/y] &= [g(x)/y] \circ []
\end{aligned}$$

The last line above illustrates that every substitution θ is more general than itself because $\theta = \theta \circ []$; 'more general' is a reflexive relation.

If two substitutions θ and ϕ are each more general than the other then they differ at most by renaming of variables, and can be regarded as equivalent. For instance, $[y/x, f(y)/w]$ and $[x/y, f(x)/w]$ are equivalent:

$$[y/x, f(y)/w] = [x/y, f(x)/w] \circ [y/x]$$
$$[x/y, f(x)/w] = [y/x, f(y)/w] \circ [x/y]$$

What does all this mean in practice? Suppose we would like to apply either θ or ϕ, where $\phi = \theta \circ \sigma$. If we apply θ then we can still get the effect of ϕ by applying σ later. Furthermore, there is an algorithm to find a most general unifier of two terms; by composition, this one unifier can generate all the unifiers of the terms.

Definition 20 A substitution θ is a *most general unifier* (MGU) of terms $t_1, \ldots, t_m$ if

- θ unifies $t_1, \ldots, t_m$, *and*
- θ is more general than every other unifier of $t_1, \ldots, t_m$.

A most general unifier of two or more *literals* is defined similarly.

Thus if θ is an MGU of terms t_1 and t_2 and $t_1\phi = t_2\phi$ then $\phi = \theta \circ \sigma$ for some substitution σ.

9.5 A simple unification algorithm

In many books, the unification algorithm is presented as operating on the concrete syntax of terms, scanning along character strings. But terms are really tree structures and are so represented in a computer. Unification should be presented as operating on trees. In fact, we need consider only binary trees, since these can represent n-ary branching trees. Unification is easily implemented in Lisp, where the basic data structure (the S-expression) is a binary tree with labelled leaves.

Our trees have three kinds of nodes:

- A *variable* $x, y, \ldots$ — can be modified by substitution
- A *constant* $a, b, \ldots$ — handles function symbols also
- A *pair* (t, u) — where t and u are terms

Unification of two terms considers nine cases, most of which are trivial. It is impossible to unify a constant with a pair; in this case the algorithm fails. When trying to unify two constants a and b, if $a = b$ then the most general unifier is []; if $a \neq b$ then unification is impossible. The interesting cases are variable-anything and pair-pair.

Unification with a variable

Consider unifying a variable x with a term t, where $x \neq t$. If x does not occur in t then the substitution $[t/x]$ has no effect on t, so it does the job trivially:

$$x[t/x] = t = t[t/x]$$

It is not hard to show that $[t/x]$ is a *most general* unifier.

If x *does* occur in t then no unifier exists, for if $x\theta = t\theta$ then the term $x\theta$ would be a subterm of itself, which is impossible.

Example 26 The terms x and $f(x)$ are not unifiable. If $x\theta = u$ then $f(x)\theta = f(u)$. Thus $x\theta = f(x)\theta$ would imply $u = f(u)$. We could, perhaps, introduce the infinite term

$$u = f(f(f(f(f(\cdots)))))$$

as a unifier, but this would require a rigorous definition of the syntax and semantics of infinite terms.

Unification of two pairs

Unifying the pairs (t_1, t_2) with (u_1, u_2) requires two recursive calls of the unification algorithm. If θ_1 unifies t_1 with u_1 and θ_2 unifies $t_2\theta_1$ with $u_2\theta_1$ then $\theta_1 \circ \theta_2$ unifies (t_1, t_2) with (u_1, u_2):

$$\begin{aligned}
(t_1, t_2)(\theta_1 \circ \theta_2) &= (t_1, t_2)\theta_1\theta_2 \\
&= (t_1\theta_1\theta_2, t_2\theta_1\theta_2) \\
&= (u_1\theta_1\theta_2, t_2\theta_1\theta_2) && \text{since } t_1\theta_1 = u_1\theta_1 \\
&= (u_1\theta_1\theta_2, u_2\theta_1\theta_2) && \text{since } (t_2\theta_1)\theta_2 = (u_2\theta_1)\theta_2 \\
&= (u_1, u_2)\theta_1\theta_2 \\
&= (u_1, u_2)(\theta_1 \circ \theta_2)
\end{aligned}$$

It is possible to prove that if θ_1 and θ_2 are *most general* unifiers then so is $\theta_1 \circ \theta_2$. If either recursive call fails then the pairs are not unifiable.

Note that the substitution θ_1 is applied to t_2 and u_2 before the second recursive call. Will this terminate, even if $t_2\theta_1$ and $u_2\theta_1$ are much bigger than t_2 and u_2? One can show that either θ_1 does not affect t_2 and u_2, or else θ_1 reduces the number of variables in the pair of terms. This is enough to show termination.

As given above, the algorithm works from left to right. An equally good alternative is to begin by unifying t_2 and u_2.

Examples of unification

These examples are given for terms rather than binary trees. The translation to binary trees is left as an exercise.

In most of these examples, the two terms have no variables in common. Most uses of unification (including resolution, see below) rename variables in one of the terms to ensure this. However, such renaming is *not* part of unification itself.

Example 27 Unify $f(x, b)$ with $f(a, y)$. Steps:

> Unify x and a getting $[a/x]$.
>
> Unify b and y getting $[b/y]$.
>
> **Result** is $[a/x] \circ [b/y]$, which is $[a/x, b/y]$.

Strictly speaking we also have to unify f with f, but this just gives [], the null substitution.

Example 28 Unify $f(x, x)$ with $f(a, b)$. Steps:

> Unify x and a getting $[a/x]$.
>
> Try to unify $x[a/x]$ and $b[a/x]$.
>
> These are a and b, distinct constants. **Fail.**

Example 29 Unify $f(x, g(y))$ with $f(y, x)$. Steps:

> Unify x and y getting $[y/x]$.
>
> Try to unify $g(y)[y/x]$ and $x[y/x]$. These are $g(y)$ and y, violating the occurs check. **Fail.**
>
> If we had renamed the variables in one of the terms beforehand, unification would have succeeded. In the next example, the two terms have no variables in common, but unification fails anyway.

Example 30 Unify $f(x, x)$ with $f(y, g(y))$. Steps:

Unify x and y getting $[y/x]$.

Try to unify $x[y/x]$ and $g(y)[y/x]$.

These are y and $g(y)$, where y occurs in $g(y)$. **Fail.**

Example 31 Unify $j(w, a, h(w))$ with $j(f(x, y), x, z)$. Steps:

Unify w and $f(x, y)$ getting $[f(x, y)/w]$.

Unify a and x (the substitution has no effect) getting $[a/x]$.

Unify $(h(w)[f(x, y)/w])[a/x]$ and $(z[f(x, y)/w])[a/x]$.

These are $h(f(x, y))[a/x]$ and $z[a/x]$.

These are $h(f(a, y))$ and z; unifier is $[h(f(a, y))/z]$.

Result is $[f(x, y)/w] \circ [a/x] \circ [h(f(a, y))/z]$. Performing the compositions, this simplifies to $[f(a, y)/w, a/x, h(f(a, y))/z]$.

Example 32 Unify $j(w, a, h(w))$ with $j(f(x, y), x, y)$. This is the previous example but with a y in place of a z.

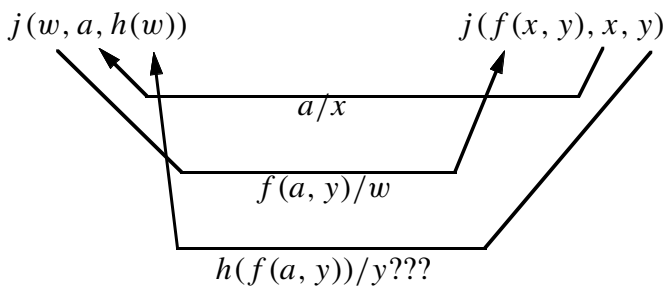
Unify w and $f(x, y)$ getting $[f(x, y)/w]$.

Unify a and x getting $[a/x]$.

Unify $(h(w)[f(x, y)/w])[a/x]$ and $(y[f(x, y)/w])[a/x]$.

These are $h(f(a, y))$ and y, but y occurs in $h(f(a, y))$. **Fail.**

Diagrams can be helpful. The lines indicate variable replacements:



Implementation remarks

To unify terms $t_1, t_2, \ldots, t_m$ for $m > 2$, compute a unifier θ of t_1 and t_2, then recursively compute a unifier σ of the terms $t_2\theta, \ldots, t_m\theta$. The overall unifier is then $\theta \circ \sigma$. If any unification fails then the set is not unifiable.

A real implementation does not need to compose substitutions. Most represent variables by pointers and effect the substitution $[t/x]$ by updating pointer x to t. The compositions are cumulative, so this works. However, if unification fails at some point, the pointer assignments must be undone!

To avoid pointers you can store the updates as a list of pairs, called an *environment*. For example, the environment $a/x, f(x)/y$ represents the substitution $[a/x, f(a)/y]$. The algorithm sketched here can take exponential time in unusual cases. Faster algorithms exist, although they are more complex and are seldom adopted.

Prolog systems, for the sake of efficiency, omit the occurs check. This can result in circular data structures and looping. It is unsound for theorem proving.

9.6 Examples of theorem proving

These two examples are fundamental. They illustrate how the occurs check enforces correct quantifier reasoning.

Example 33 Consider a proof of

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

Produce clauses separately for the antecedent and for the negation of the consequent; this is more efficient than producing clauses for the negation of the entire formula.

- The antecedent is $\exists y\, \forall x\, R(x, y)$; replacing y by the Skolem constant a yields the clause $\{R(x, a)\}$.
- In $\neg(\forall x\, \exists y\, R(x, y))$, pushing in the negation produces $\exists x\, \forall y\, \neg R(x, y)$. Replacing x by the Skolem constant b yields the clause $\{\neg R(b, y)\}$.

Unifying $R(x, a)$ with $R(b, y)$ detects the contradiction $R(b, a) \wedge \neg R(b, a)$.

Example 34 In a similar vein, let us try to prove

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

- Here the antecedent is $\forall x\, \exists y\, R(x, y)$; replacing y by the Skolem function f yields the clause $\{R(x, f(x))\}$.
- The negation of the consequent is $\neg(\exists y\, \forall x\, R(x, y))$, which becomes $\forall y\, \exists x\, \neg R(x, y)$. Replacing x by the Skolem function g yields the clause $\{\neg R(g(y), y)\}$.

Observe that $R(x, f(x))$ and $R(g(y), y)$ are not unifiable because of the occurs check. And so it should be, because the original formula is not a theorem!

Exercise 31 For each of the following pairs of terms, give a most general unifier or explain why none exists. Do not rename variables prior to performing the unification.

$f(g(x), z)$	$f(y, h(y))$
$j(x, y, z)$	$j(f(y, y), f(z, z), f(a, a))$
$j(x, z, x)$	$j(y, f(y), z)$
$j(f(x), y, a)$	$j(y, z, z)$
$j(g(x), a, y)$	$j(z, x, f(z, z))$

10 Applications of Unification

By means of unification, we can extend resolution to first-order logic. As a special case we obtain Prolog. Other theorem provers are also based on unification. Other applications include polymorphic type checking for the language ML.

10.1 Binary resolution

We now define the binary resolution rule with unification:

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

As before, the first clause contains B and other literals, while the second clause contains $\neg D$ and other literals. The substitution σ is a unifier of B and D (almost always a *most general* unifier). This substitution is applied to all remaining literals, producing the conclusion.

The variables in one clause are renamed before resolution to prevent clashes with the variables in the other clause. Renaming is sound because the scope of each variable is its clause. Resolution is sound because it takes an instance of each clause — the instances are valid, because the clauses are universally valid —

and then applies the propositional resolution rule, which is sound. For example, the two clauses

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

yield the empty clause in a single resolution step. This works by renaming variables — say, x to y in the second clause — and unifying $P(x)$ with $P(g(y))$. Forgetting to rename variables is fatal, because $P(x)$ cannot be unified with $P(g(x))$.

10.2 Factoring

In the general case, the resolution rule must perform *factoring*. This uses additional unifications to identify literals in the same clause. Factoring can make the clause $\{P(x, b), P(a, y)\}$ behave like the clause $\{P(a, b)\}$, since $P(a, b)$ is the result of unifying $P(x, b)$ with $P(a, y)$.

The factoring unifications are done at the same time as the unification of the complementary literals in the two clauses. The binary resolution rule with factoring is

$$\frac{\{B_1, \ldots, B_k, A_1, \ldots, A_m\} \quad \{\neg D_1, \ldots, \neg D_l, C_1, \ldots, C_n\}}{\{A_1, \ldots, A_m, C_1, \ldots, C_n\}\sigma}$$

where σ is the most general substitution such that

$$B_1\sigma = \cdots = B_k\sigma = D_1\sigma = \cdots = D_l\sigma.$$

Resolution with factoring is *refutation complete*: it will find a contradiction if there is one. Showing this is difficult.

The search space is huge: resolution with factoring can be applied in many different ways, every time. Modern resolution systems use highly complex heuristics to limit the search. Typically they only perform resolutions that can lead (perhaps after several steps) to very short clauses, and they discard the intermediate clauses produced along the way. Dozens of flags and parameters influence their operation.

Example 35 Let us prove $\forall x\, \exists y\, \neg(P(y, x) \leftrightarrow \neg P(y, y))$.

Negate and expand the $\leftrightarrow$, getting

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

Its negation normal form is

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

Skolemization yields

$$(\neg P(y, a) \vee \neg P(y, y)) \wedge (P(y, y) \vee P(y, a))$$

The clauses are

$$\{\neg P(y, a), \neg P(y, y)\} \qquad \{P(y, y), P(y, a)\}$$

Note that $\neg P(a, a)$ is an instance of the first clause and that $P(a, a)$ is an instance of the second, contradiction. This is a one-step proof! But it involves both resolution and factoring, since the 2-literal clauses must collapse to singleton clauses.

Example 36 Let us prove $\exists x\,[P \rightarrow Q(x)] \wedge \exists x\,[Q(x) \rightarrow P] \rightarrow \exists x\,[P \leftrightarrow Q(x)]$. The clauses are

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

A short resolution proof follows. The complementary literals are underlined:

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

Exercise 32 Show the steps of converting $\exists x\,[P \rightarrow Q(x)] \wedge \exists x\,[Q(x) \rightarrow P] \rightarrow \exists x\,[P \leftrightarrow Q(x)]$ into clauses. Then show two resolution proofs different from the one shown above.

Exercise 33 Is the clause $\{P(x, b), P(a, y)\}$ logically equivalent to the unit clause $\{P(a, b)\}$? Is the clause $\{P(y, y), P(y, a)\}$ logically equivalent to $\{P(y, a)\}$? Explain both answers.

10.3 Prolog clauses

Prolog clauses, also called Horn clauses, have at most one positive literal. A *definite* clause is one of the form

$$\{\neg A_1, \ldots, \neg A_m, B\}$$

It is logically equivalent to $(A_1 \wedge \cdots \wedge A_m) \rightarrow B$. Prolog's notation is

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

If $m = 0$ then the clause is simply written as B and is sometimes called a *fact*.

A *negative* or *goal* clause is one of the form

$$\{\neg A_1, \ldots, \neg A_m\}$$

Prolog permits just one of these; it represents the list of unsolved goals. Prolog's notation is

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

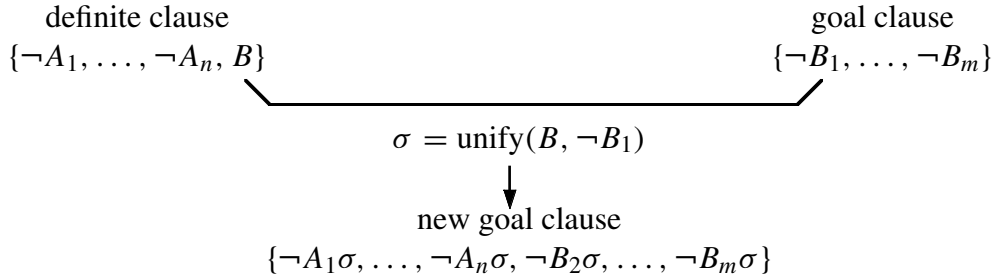
A Prolog database consists of definite clauses. Observe that definite clauses cannot express negative assertions, since they must contain a positive literal. From a mathematical point of view, they have little expressive power; every set of definite clauses is consistent! Even so, definite clauses are a natural notation for many problems.

Exercise 34 Show that every set of definite clauses is consistent. (Hint: first consider propositional logic, then extend your argument to first order logic.)

10.4 Prolog computations

A Prolog computation takes a database of definite clauses together with one goal clause. It repeatedly resolves the goal clause with some definite clause to produce a new goal clause. If resolution produces the empty goal clause, then execution succeeds.

Here is a diagram of a Prolog computation step:



This is a *linear* resolution (§7). Two program clauses are never resolved with each other. The result of each resolution step becomes the next goal clause; the previous goal clause is discarded after use.

Prolog resolution is very efficient, compared with general resolution, because it involves less search and storage. General resolution must consider all possible pairs of clauses; it adds their resolvents to the existing set of clauses; it spends

a great deal of effort getting rid of subsumed (redundant) clauses and probably useless clauses. Prolog always resolves some program clause with the goal clause. Because goal clauses do not accumulate, Prolog requires little storage. Prolog never uses factoring, with all its complexity.

Prolog has a fixed, deterministic execution strategy. The program is is regarded as a list of clauses, not a set; the clauses are tried strictly in order. With a clause, the literals are also regarded as a list. The literals in the goal clause are proved strictly from left to right. The goal clause's first literal is replaced by the literals from the unifying program clause, preserving their order.

Prolog's search strategy is depth-first. To illustrate what this means, suppose that the goal clause is simply $\leftarrow P$ and that the program clauses are $P \leftarrow P$ and $P \leftarrow$. Prolog will resolve $P \leftarrow P$ with $\leftarrow P$ to obtain a new goal clause, which happens to be identical to the original one. Prolog never notices the repeated goal clause, so it repeats the same useless resolution over and over again. Depth-first search means that at every 'choice point,' such as between using $P \leftarrow P$ and $P \leftarrow$, Prolog will explore every avenue arising from its first choice before considering the second choice. Obviously, the second choice would prove the goal trivially, but Prolog never notices this.

10.5 Example of Prolog execution

Here are axioms about the English succession: how y can become King after x.

$$\forall x\, \forall y\, (\text{oldestson}(y, x) \wedge \text{king}(x) \rightarrow \text{king}(y))$$

$$\forall x\, \forall y\, (\text{defeat}(y, x) \wedge \text{king}(x) \rightarrow \text{king}(y))$$

$$\text{king(richardIII)}$$

$$\text{defeat(henryVII, richardIII)}$$

$$\text{oldestson(henryVIII, henryVII)}$$

The goal is to prove king(henryVIII).

These axioms correspond to the following definite clauses:

$$\{\neg\text{oldestson}(y, x), \neg\text{king}(x), \text{king}(y)\}$$

$$\{\neg\text{defeat}(y, x), \neg\text{king}(x), \text{king}(y)\}$$

$$\{\text{king(richardIII)}\}$$

$$\{\text{defeat(henryVII, richardIII)}\}$$

$$\{\text{oldestson(henryVIII, henryVII)}\}$$

The goal clause is

$$\{\neg\text{king(henryVIII)}\}$$

Figure 2 shows the execution. The subscripts in the clauses are to rename the variables.

Note how crude this formalization is. It says nothing about the passage of time, about births and deaths, about not having two kings at once. Henry VIII was the second son of Henry VII; the first son, Arthur, died before his father. Logic is clumsy for talking about situations in the real world.

The Frame Problem in Artificial Intelligence reveals another limitation of logic. Consider writing an axiom system to describe a robot's possible actions. We might include an axiom to state that if the robot lifts an object at time t, then it will be holding the object at time $t + 1$. But we also need to assert that the positions of everything else remain the same as before. Then we must consider the possibility that the object is a table and has other things on top of it ...

Prolog is a powerful and useful language, but it is not necessarily logic. Most Prolog programs rely on special predicates that affect execution but have no logical meaning. There is a huge gap between the theory and practice of logic programming.

Exercise 35 Convert these formulæ into clauses, showing each step: negating the formula, eliminating $\rightarrow$ and $\leftrightarrow$, pushing in negations, moving the quantifiers, Skolemizing, dropping the universal quantifiers, and converting the matrix into CNF.

$$(\forall x\, \exists y\, R(x, y)) \rightarrow (\exists y\, \forall x\, R(x, y))$$
$$(\exists y\, \forall x\, R(x, y)) \rightarrow (\forall x\, \exists y\, R(x, y))$$
$$\exists x\, \forall yz\, ((P(y) \rightarrow Q(z)) \rightarrow (P(x) \rightarrow Q(x)))$$
$$\neg\exists y\, \forall x\, (R(x, y) \leftrightarrow \neg\exists z\, (R(x, z) \wedge R(z, x)))$$

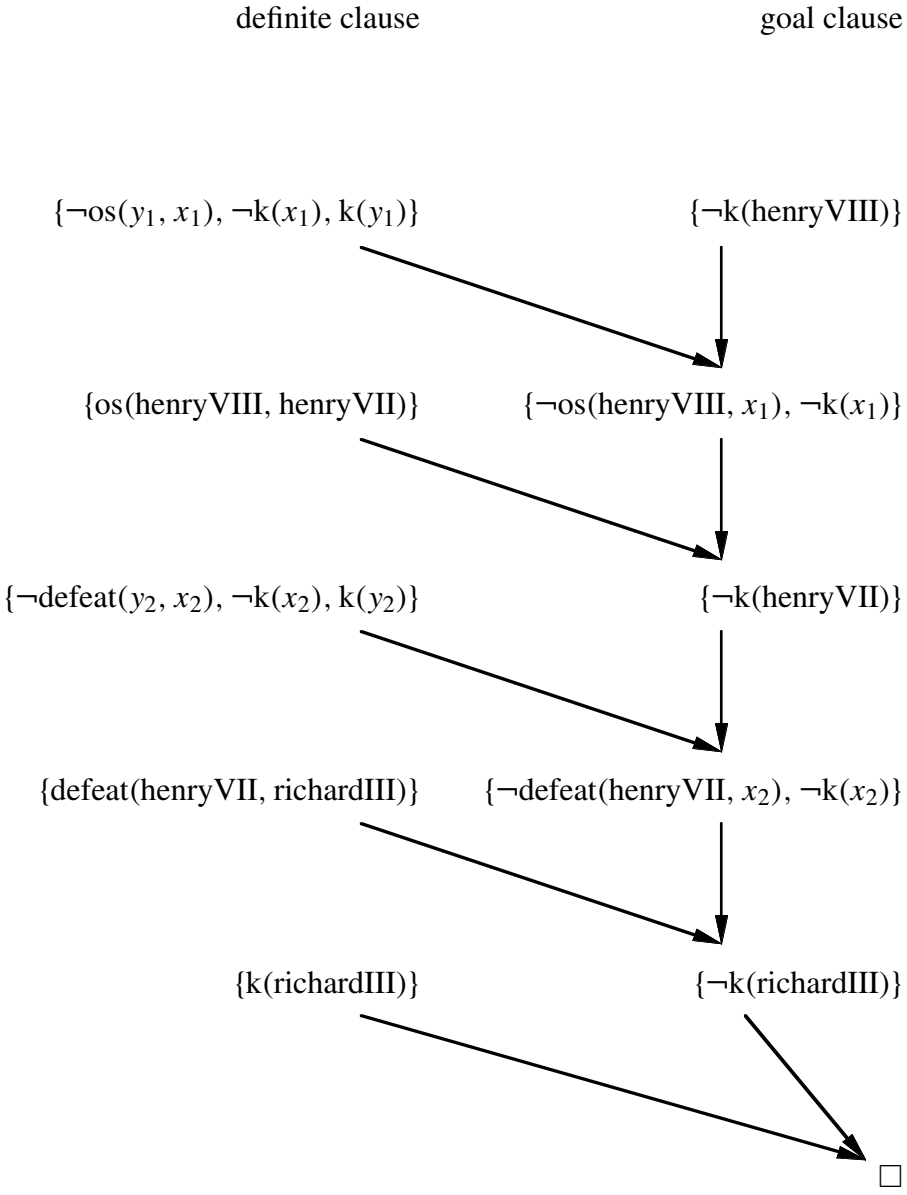


Figure 2: Execution of a Prolog program (os = oldestson, k = king)

Exercise 36 Consider the Prolog program consisting of the definite clauses

$$P(f(x, y)) \leftarrow Q(x), R(y)$$
$$Q(g(z)) \leftarrow R(z)$$
$$R(a) \leftarrow$$

Describe the Prolog computation starting from the goal clause $\leftarrow P(v)$. Keep track of the substitutions affecting v to determine what answer the Prolog system would return.

Exercise 37 Find a refutation from the following set of clauses using resolution with factoring.

$$\{\neg P(x, a), \neg P(x, y), \neg P(y, x)\}$$
$$\{P(x, f(x)), P(x, a)\}$$
$$\{P(f(x), x), P(x, a)\}$$

Exercise 38 Prove the following formulæ by resolution, showing all steps of the conversion into clauses. Remember to negate first!

$$\forall x\ (P \vee Q(x)) \rightarrow (P \vee \forall x\ Q(x))$$
$$\exists xy\ (P(x, y) \rightarrow \forall vw\ P(v, w))$$

11 Modal Logics

There are many forms of modal logic. Each one is based upon two parameters:

- W is the set of *possible worlds* (machine states, future times, ...)
- R is the *accessibility relation* between worlds (state transitions, flow of time, ...)

The pair (W, R) is called a *modal frame*.

The two *modal operators*, or *modalities*, are $\Box$ and $\Diamond$:

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

Here 'necessarily true' means 'true in all worlds accessible from the present one'. The modalities are related by the law $\neg\Diamond A \simeq \Box\neg A$; in words, '$A$ is necessarily false' is equivalent to 'it is not possible that A is true'.

Complex modalities are made up of strings of the modal operators, such as $\Box\Box A$, $\Box\Diamond A$, $\Diamond\Box A$, etc. Typically many of these are equivalent to others; in $S4$, a a standard modal logic, $\Box\Box A$ is equivalent to $\Box A$.

11.1 Semantics of propositional modal logic

Here are some basic definitions, with respect to a particular frame (W, R):

An *interpretation* I maps the propositional letters to subsets of W. For each letter P, the set $I(P)$ consists of those worlds in which P is regarded as true.

If $w \in W$ and A is a modal formula, then $w \Vdash A$ means A is true in world w. This relation is defined as follows:

$$\begin{array}{lll} w \Vdash P & \iff & w \in I(P) \\ w \Vdash \Box A & \iff & v \Vdash A \text{ for all } v \text{ such that } R(w, v) \\ w \Vdash \Diamond A & \iff & v \Vdash A \text{ for some } v \text{ such that } R(w, v) \\ w \Vdash A \vee B & \iff & w \Vdash A \text{ or } w \Vdash B \\ w \Vdash A \wedge B & \iff & w \Vdash A \text{ and } w \Vdash B \\ w \Vdash \neg A & \iff & w \Vdash A \text{ does not hold} \end{array}$$

This definition of truth is more complex than we have seen previously (§2.2), because of the extra parameters W and R. We shall not consider quantifiers at all; they really complicate matters, especially if the universe is allowed to vary from one world to the next.

For a particular frame (W, R), further relations can be defined in terms of $w \Vdash A$:

$$\begin{array}{rl} \models_{W,R,I} A & \text{means } w \Vdash A \text{ for all } w \text{ under interpretation } I \\ \models_{W,R} A & \text{means } w \Vdash A \text{ for all } w \text{ and all } I \end{array}$$

Now $\models A$ means $\models_{W,R} A$ for all frames. We say that A is *universally valid*. In particular, all tautologies of propositional logic are universally valid.

Typically we take additional assumptions on accessibility relation. We may assume, for example, that R is transitive, and consider whether a formula holds under all frames satisfying that assumption. More formulæ become universally valid if we take additional assumptions, as they exclude some modal frames from consideration.

11.2 Hilbert-style proof systems for the modal logics

Start with any proof system for propositional logic. Then add the *distribution* axiom

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

and the *necessitation* rule:

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

There are no axioms or inference rules for $\Diamond$. The modality is viewed simply as an abbreviation:

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

The distribution axiom clearly holds in our semantics. The propositional connectives obey their usual truth tables in each world. If A holds in all worlds, and $A \to B$ holds in all worlds, then B holds in all worlds. Thus if $\Box A$ and $\Box(A \to B)$ hold then so does $\Box B$, and that is the essence of the distribution axiom.

The necessitation rule states that all theorems are necessarily true. In more detail, if A can be proved, then it holds in all worlds; therefore $\Box A$ is also true.

The modal logic that results from adding the distribution axiom and necessitation rule is called K. It is a pure modal logic, from which others are obtained by adding further axioms. Each axiom corresponds to a property that is assumed to hold of all accessibility relations. Here are just a few of the main ones:

T	$\Box A \to A$	(reflexive)
4	$\Box A \to \Box\Box A$	(transitive)
B	$A \to \Box\Diamond A$	(symmetric)

Logic T includes axiom T: reflexivity. Logic $S4$ includes axioms T and 4: reflexivity and transitivity. Logic $S5$ includes axioms T, 4 and B: reflexivity, transitivity and symmetry; these imply that the accessibility relation is an equivalence relation, which is a strong condition.

Other conditions on the accessibility relation concern forms of *confluence*. One such condition might state that if w_1 and w_2 are both accessible from w then there exists some v that is accessible from both w_1 and w_2.

11.3 Sequent Calculus Rules for $S4$

We shall mainly look at $S4$, which is one of the mainstream modal logics. The sequent calculus for $S4$ extends the usual sequent rules for propositional logic with additional ones for $\Box$ and $\Diamond$. Four rules are required because the modalities may occur on either the left or right side of a sequent.

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

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

The $(\Box r)$ rule is analogous to the necessitation rule. But now A may be proved from other formulæ. This introduces complications. Modal logic is notorious for requiring strange conditions in inference rules. The symbols Γ^* and Δ^* stand for sets of formulæ, defined as follows:

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

In effect, applying rule $(\Box r)$ in a backward proof throws away all left-hand formulæ that do not begin with a $\Box$ and all right-hand formulæ that do not begin with a $\Diamond$.

If you consider why the $(\Box r)$ rule actually holds, it is not hard to see why those formulæ must be discarded. If we forgot about the restriction, then we could use $(\Box r)$ to infer $A \Rightarrow \Box A$ from $A \Rightarrow A$, which is ridiculous. The restriction ensures that the proof of A in the premise is independent of any particular world.

The rule $(\Diamond l)$ is an exact dual of $(\Box r)$. The obligation to discard formulæ forces us to plan proofs carefully. If rules are applied in the wrong order, vital information may have to be discarded and the proof will fail.

11.4 Some sample proofs in $S4$

A few examples will illustrate how the $S4$ sequent calculus is used.

The distribution axiom is assumed in the Hilbert-style proof system. Using the sequent calculus, we can prove it:

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

A few applications of $(\to r)$ yield the desired theorem, $\Box(A \to B) \to (\Box A \to \Box B)$.

Working backwards, the first step is to apply rule $(\Box r)$. This rule discards non-$\Box$ formulæ, but there happen not to be any. Had we first applied $(\Box l)$, removing the boxes from the left side, then we should get stuck:

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

Applying $(\Box r)$ before $(\Box l)$ is analogous to applying $(\forall r)$ before $(\forall l)$. The analogy because $\Box A$ has an implicit universal quantifier: for all accessible worlds.

The following two proofs establish the equivalence $\Box\Diamond\Box\Diamond A \simeq \Box\Diamond A$. Strings of modalities, like $\Box\Diamond\Box\Diamond$ and $\Box\Diamond$, are called *operator strings*. So the pair of results establish an operator string equivalence.

Here is the first entailment in the equivalence. As usual we apply $(\Box r)$ before $(\Box l)$. Dually, and analogously to the treatment of the $\exists$ rules, we apply $(\Diamond l)$ before $(\Diamond r)$:

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

The opposite entailment is easy to prove:

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

Logic $S4$ enjoys many operator string equivalences, including $\Box\Box A \simeq \Box A$. And for every operator string equivalence, its dual (obtained by exchanging $\Box$ with $\Diamond$) also holds. In particular, $\Diamond\Diamond A \simeq \Diamond A$ and $\Diamond\Box\Diamond\Box A \simeq \Diamond\Box A$ hold. So we only need to consider operator strings in which the boxes and diamonds alternate, and whose length does not exceed three.

The distinct $S4$ operator strings are therefore $\Box$, $\Diamond$, $\Box\Diamond$, $\Diamond\Box$, $\Box\Diamond\Box$ and $\Diamond\Box\Diamond$.

Finally, here are two attempted proofs that fail — because their conclusions are not theorems! The modal sequent $A \Rightarrow \Box\Diamond A$ states that if A holds now then it necessarily holds again: from each accessible world, another world is accessible in which A holds. This formula is valid if the accessibility relation is symmetric; then one could simply return to the original world. The formula is therefore a theorem of $S5$ modal logic, but not $S4$.

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

Here, the modal sequent $\Diamond A, \Diamond B \Rightarrow \Diamond(A \wedge B)$ states that if A holds in some accessible world, and B holds in some accessible world, then both A and B hold in some accessible world. It is a fallacy because those two worlds need not coincide.

The $(\Diamond l)$ rule prevents us from removing the diamonds from both $\Diamond A$ and $\Diamond B$; if we choose one we must discard the other:

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

The topmost sequent may give us a hint as to why the conclusion fails. Here we are in a world in which B holds, and we are trying to show $A \wedge B$, but there is no reason why A should hold in that world.

Exercise 39 Why does the dual of an operator string equivalence also hold?

Exercise 40 Prove the sequent $\Diamond(A \vee B) \Rightarrow \Diamond A, \Diamond B$.

Exercise 41 Prove the sequent $\Diamond A \vee \Diamond B \Rightarrow \Diamond(A \vee B)$. Together with the previous exercise, this yields $\Diamond(A \vee B) \simeq \Diamond A \vee \Diamond B$.

Exercise 42 Prove the sequent $\Diamond(A \rightarrow B), \Box A \Rightarrow \Diamond B$.

Exercise 43 Prove the equivalence $\Box(A \wedge B) \simeq \Box A \wedge \Box B$.

Exercise 44 Prove the sequent $\Box\Diamond A \Rightarrow \Box\Diamond\Box\Diamond A$.

12 Tableaux-Based Methods

There is a lot of redundancy among the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$. We could get away using only three of them (two if we allowed exclusive 'or'), but use the full set for readability. There is also a lot of redundancy in the sequent calculus, because it was designed to model human reasoning, not to be as small as possible.

One approach to removing redundancy results in the resolution method. Clause notation replaces the connectives, and there is only one inference rule. A less radical approach still removes much of the redundancy, while preserving much of the natural structure of formulæ. This approach is often adopted by proof theorists because of its logical simplicity; it is also amenable to implementation.

12.1 Simplifying the sequent calculus

The usual formalisation of first-order logic involves seven connectives, or nine in the case of modal logic. For each connective the sequent calculus has a left and a right rule. So, apart from the structural rules (basic sequent and cut) there are 14 rules, or 18 for modal logic.

Suppose we allow only formulæ in negation normal form. This immediately disposes of the connectives $\rightarrow$ and $\leftrightarrow$. Really $\neg$ is discarded also, as it is allowed only on propositional letters. So only four connectives remain, six for modal logic.

The greatest simplicity gain comes in the sequent rules. The only sequent rules that move formulæ from one side to the other (across the $\Rightarrow$ symbol) are the rules for the connectives that we have just discarded. Half of the sequent rules can be discarded too. It makes little difference whether we discard the left-side rules or the right-side rules.

Let us discard the right-side rules. The resulting system allows sequents of the form $A \Rightarrow$. It is a form of refutation system (proof by contradiction), since the formula A has the same meaning as the sequent $\neg A \Rightarrow$. Moreover, a basic sequent has the form of a contradiction.

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

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

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

Rule $(\exists l)$ has the usual proviso: it holds *provided* x is not free in the conclusion!

We can extend the system to $S4$ modal logic by adding just two further rules, one for $\Box$ and one for $\Diamond$:

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

As previously, Γ^* is defined to erase all non-$\Box$ formulæ:

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

We have gone from 14 rules to four, ignoring the structural rules. For modal logic, we have gone from 18 rules to six.

A simple proof will illustrate how the simplified system works. Let us prove $\forall x\ (A \rightarrow B) \Rightarrow A \rightarrow \forall x\ B$, where x is not free in A. We must negate the formula

and convert it to NNF; the resulting sequent is $A \wedge \exists x\, \neg B,\ \forall x\, (\neg A \vee B) \Rightarrow$. Elaborate explanations should not be necessary because this sequent calculus is essentially a subset of the one described in §6.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A,\ \neg B,\ \neg A \Rightarrow} \quad \dfrac{}{A,\ \neg B,\ B \Rightarrow}}{A,\ \neg B,\ \neg A \vee B \Rightarrow}\,(\vee l)}{A,\ \neg B,\ \forall x\, (\neg A \vee B) \Rightarrow}\,(\forall l)}{A,\ \exists x\, \neg B,\ \forall x\, (\neg A \vee B) \Rightarrow}\,(\exists l)}{A \wedge \exists x\, \neg B,\ \forall x\, (\neg A \vee B) \Rightarrow}\,(\wedge l)$$

12.2 Mechanising the technique

Some proof theorists adopt the simplified sequent calculus as their formalisation of first-order logic. It has most of the advantages of the usual sequent calculus, without the redundancy. But can we use it as the basis for a theorem prover? Implementing the calculus (or indeed, implementing the full sequent calculus) requires a treatment of quantifiers. As with the resolution method, we can use unification together with Skolemization.

First, consider how to add unification. The rule $(\forall l)$ to substitutes some term for the bound variable. Since we do not know in advance what the term ought to be, instead substitute a free variable. The variable ought to be fresh, not used elsewhere in the proof:

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

Then allow unification to instantiate variables with terms. This should occur when trying to solve any goal containing two formulæ, $\neg A$ and B. Try to unify A with B, producing a basic sequent. Of course, instantiating a variable updates the entire proof tree.

Rule $(\exists l)$, used in backward proof, must create a fresh variable. That will no longer do, in part because we now allow variables to become instantiated by terms. We have a choice of techniques, but the simplest is to Skolemize the formula. All existential quantifiers disappear, so we can discard rule $(\exists l)$.

Previously (§8.2) we performed Skolemization on formulæ in prenex form: all quantifiers at the front. The outermost existentially-bound variable was replaced by a function, which took as many arguments as there were enclosing universal quantifiers. But there is no need to pull quantifiers to the front. Precisely the same approach works, although now the existential quantifiers are found in subformulæ instead of being lined up in a row.

The Skolem form of $\forall y\, \exists z\, Q(y,z) \wedge \exists x\, P(x)$ is $\forall y\, Q(y, f(y)) \wedge P(a)$. The subformula $\exists x\, P(x)$ goes to $P(a)$ and not to $P(g(y))$ because it is outside the scope of the $\forall y$.

12.3 Sample proofs

To demonstrate the system, let us prove the formula $\exists x\, \forall y\, [P(x) \to P(y)]$. First negate it and convert to NNF, getting $\forall x\, \exists y\, [P(x) \wedge \neg P(y)]$. The Skolemized sequent to be proved is $\forall x\, [P(x) \wedge \neg P(f(x))] \Rightarrow$. Unification completes the proof by creating a basic sequent; there are two distinct ways of doing so:

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

In the first inference from the bottom, the universal formula is retained because it must be used again. In principle, universally quantified formulæ ought always to be retained, as they may be used any number of times. I normally erase them to save space.

Pulling quantifiers to the front is not merely unnecessary; it can be harmful. Skolem functions should have as few arguments as possible, as this leads to shorter proofs. Attaining this requires that quantifiers should have the smallest possible scopes; we ought to push quantifiers in, not pull them out. This is sometimes called *miniscope* form.

For example, the formula $\exists x\, \forall y\, [P(x) \to P(y)]$ is tricky to prove. But putting it in miniscope form makes its proof trivial. Let us do this step by step:

$$\begin{aligned}
\text{Negate; convert to NNF:} &\quad \forall x\, \exists y\, [P(x) \wedge \neg P(y)] \\
\text{Push in the } \exists y: &\quad \forall x\, [P(x) \wedge \exists y\, \neg P(y)] \\
\text{Push in the } \forall x: &\quad \forall x\, P(x) \wedge \exists y\, \neg P(y) \\
\text{Skolemize:} &\quad \forall x\, P(x) \wedge \neg P(a)]
\end{aligned}$$

The formula $\forall x\, P(x) \wedge \neg P(a)]$ is obviously inconsistent. Here is its refutation in the modified sequent calculus:

$$\dfrac{\dfrac{\dfrac{y \mapsto a}{P(y),\ \neg P(a) \Rightarrow}\ basic}{\forall x\, P(x),\ \neg P(a) \Rightarrow}\ (\forall l)}{\forall x\, P(x) \wedge \neg P(a) \Rightarrow}\ (\wedge l)$$

A failed proof is always illuminating. Let us try to prove the invalid formula

$$\forall x\,[P(x) \vee Q(x)] \Rightarrow \forall x\,P(x) \vee \forall x\,Q(x).$$

Negation and conversion to NNF gives $\exists x\,\neg P(x) \wedge \exists x\,\neg Q(x)$, $\forall x\,[P(x) \vee Q(x)]$.

Skolemization gives $\neg P(a) \wedge \neg Q(b)$, $\forall x\,[P(x) \vee Q(x)]$.

The proof fails because a and b are distinct constants. It is impossible to instantiate y to both simultaneously.

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

12.4 Tableaux-based theorem provers

An *analytic tableau* represents a partial proof as a set of *branches* of formulæ. Each formula on a branch is *expanded* until this is no longer possible (and the proof fails) or until the proof succeeds.

Expanding a conjunction $A \wedge B$ on a branch replaces it by the two conjuncts, A and B. Expanding a disjunction $A \vee B$ splits the branch in two, with one branch containing A and the other branch B. Expanding the quantification $\forall x\,A$ extends the branch by a formula of the form $A[t/x]$. If a branch contains both A and $\neg A$ then it is said to be *closed*. When all branches are closed, the proof has succeeded.

A tableau is, in fact, nothing but a compact, graph-based representation of a set of sequents. The branch operations described above correspond to our sequent rules in an obvious way.

Quite a few theorem provers have been based upon the tableau method. The simplest by far is due to Beckert and Posegga (1994) and is called leanTAP. The entire program appears below! Its deductive system is similar to the reduced sequent calculus we have just studied. It relies on some Prolog tricks, and is certainly not pure Prolog code. It demonstrates just how simple a theorem prover can be. leanTAP does not outperform big resolution systems. But it quickly proves some fairly hard theorems.

```
prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,
        prove(A,[B|UnExp],Lits,FreeV,VarLim).
prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,
        prove(A,UnExp,Lits,FreeV,VarLim),
        prove(B,UnExp,Lits,FreeV,VarLim).
```

```
prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,
        \+ 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],_,_) :-
        (Lit = -Neg; -Lit = Neg) ->
        (unify(Neg,L); prove(Lit,[],Lits,_,_)).
prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-
        prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).
```

(Note: you are not expected to memorize this program or understand how it works in any detail.)

Exercise 45 Use the tableau calculus to prove examples given in previous sections.

References

Beckert, B. and Posegga, J. (1994). leanTAP: Lean, tableau-based theorem proving. In A. Bundy, editor, *Automated Deduction — CADE-12 International Conference*, LNAI 814, pages 793–797. Springer.

Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *Computing Surveys*, **24**(3), 293–318.

Zhang, H. and Stickel, M. E. (1994). An efficient algorithm for unit propagation. Technical Report 94-12, Computer Science Dept., University of Iowa.