Mathematical Logic – Proofs by Resolution (recap)

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

we want to prove that the negation is false

- $P \wedge Q \to R$ processed as a whole
- or process separately P, Q and $\neg R$
- why?

⇒ its negation leads to a contradiction

The method:

- negate the statement (why?)

(1)
- convert to prenex form
  - move quantifiers as prefix
- convert to skolem form
  - remove quatifiers and replace with functions
- convert to clausal form = conj NF = (... ∨ ...) ∧ (... ∨ ...) ...

(2)
- unifications and substitutions

(3)
- resolve by (predicates) resolution

(4)
- resolution for propositions (explanation)
- examples

(5)
- ex 37, example of predicates resolution

- Prolog computation
- example, the English succession
- Prolog execution of above

fact	$B.$	$\{B\}$
definite clause	$B \leftarrow A_1, \dots, A_n.$	$\{\neg A_1, \dots, \neg A_n, B\}$
goal	$\leftarrow A_1, \dots, A_n.$	$\{\neg A_1, \dots, \neg A_n\}$

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

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 16 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 17 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:[6]

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

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.

[6] 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.[7] 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 18 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:

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

[7] 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 19 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.

Example 22 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 23 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 24 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 25 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 26 Recall the unifiers of $g(g(x))$ and $g(y)$. The unifier $[g(x)/y]$ is more general than the others listed, for

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

- 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 36 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 37 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 $\underline{\{P\}}$ with $\underline{\{\neg P\}}$ 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

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

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:

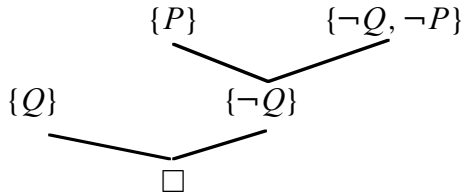
$$\frac{\{P\} \quad \{\neg P, \neg Q\}}{\{\neg Q\}}$$

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.

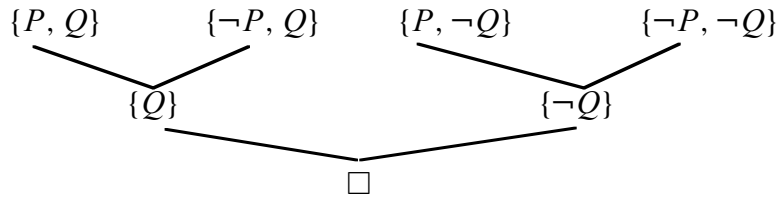
It is nicer to draw a tree like this:



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

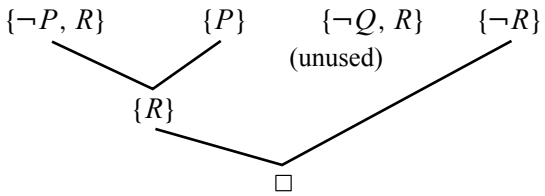
$$\{P, Q\} \qquad \{\neg P, Q\} \qquad \{P, \neg Q\} \qquad \{\neg P, \neg Q\}$$

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 could mislead. Must a proof use each clause exactly once? No! A clause may be used repeatedly, and many problems contain redundant clauses. Here is an example:



Redundant clauses can make the theorem-prover flounder; this is a challenge facing the field.

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

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

Try negating this and converting to CNF. Using the law $\neg(A \rightarrow B) \simeq A \wedge \neg B$, 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 $H \rightarrow M \vee N$ is essentially in clause form already:

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

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

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

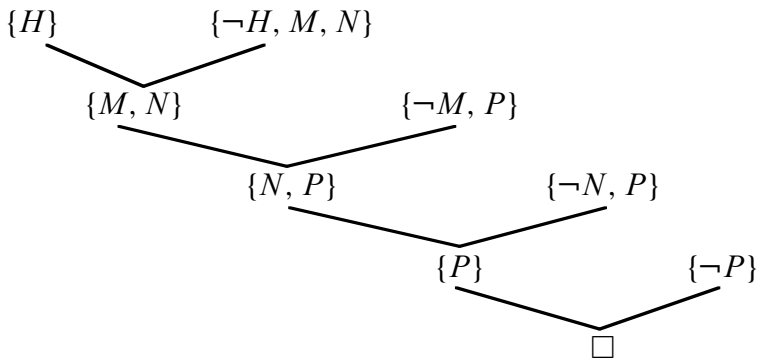
Assumption $N \rightarrow L \wedge P$ also becomes two clauses:

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

The negated conclusion, $\neg(H \rightarrow P)$, 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 resolution.

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.

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 37 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

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

* prolog is a description of what we know about the problem to be a fact, goal

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

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

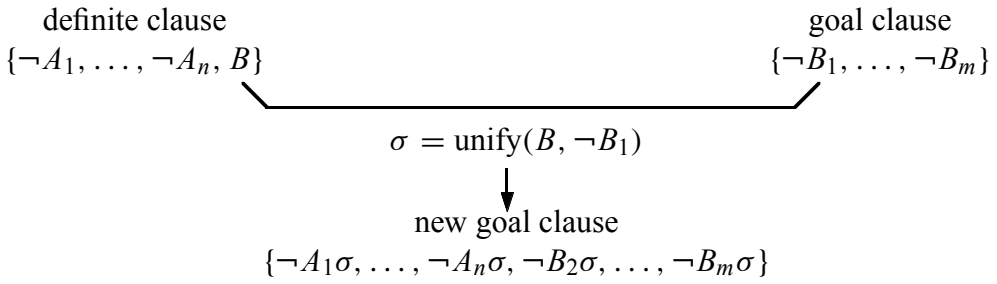
A Prolog database consists of definite clauses. Observe that definite clauses cannot express negative assertions, since they must contain a positive literal. From a mathematical point of view, they have little expressive power; every set of definite clauses is consistent! Even so, definite clauses are 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 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 and does not even remove repeated literals from a clause.

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:

definite clauses

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

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

{king(richardIII)}

{defeat(henryVII, richardIII)}

{oldestson(henryVIII, henryVII)}

The goal clause is

{¬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.

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

goal is always part of the reasoning
Top → down approach

definite clause goal clause

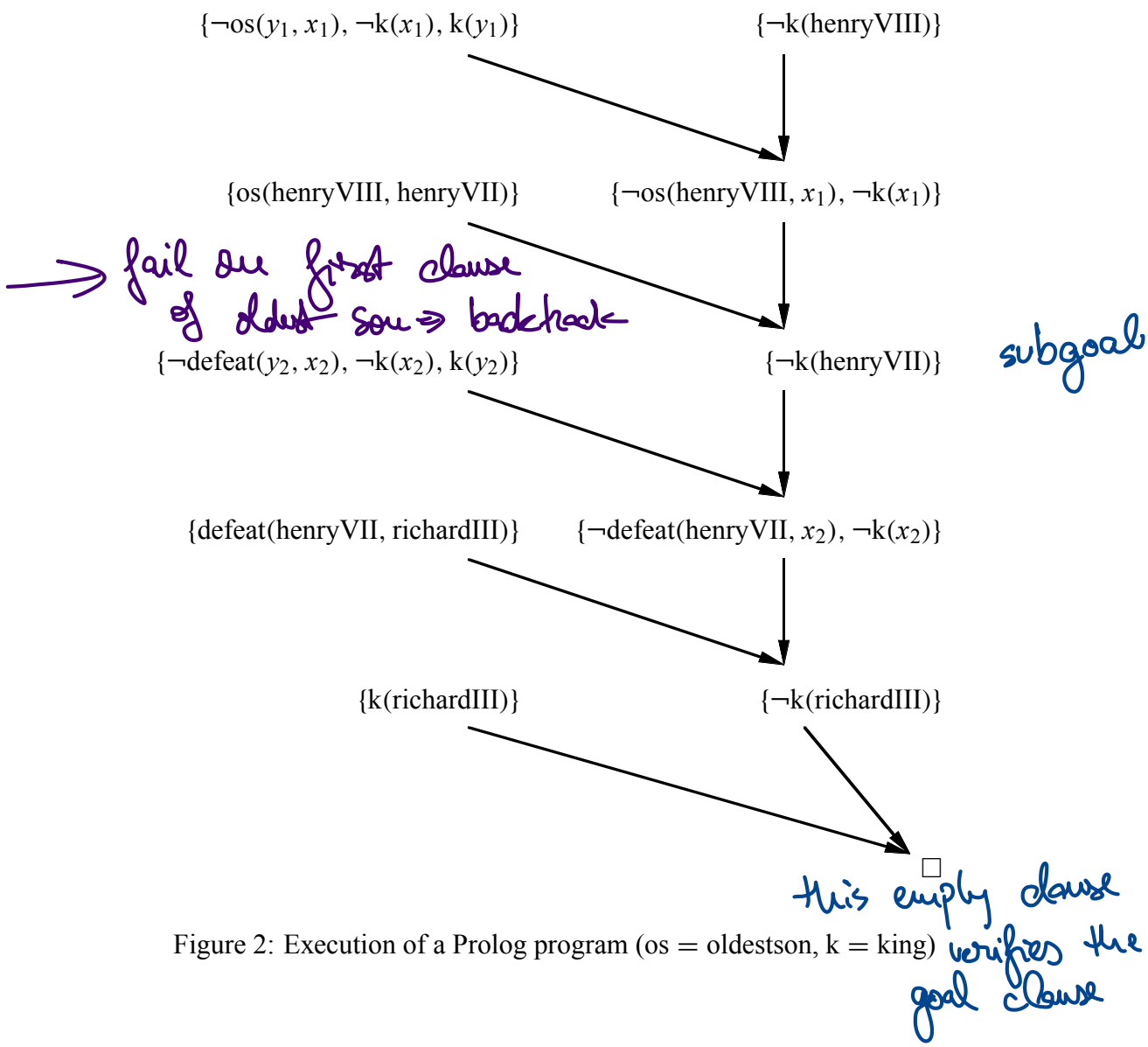


Figure 2: Execution of a Prolog program (os = oldestson, k = king)

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!

$$\forall x\ (P \vee Q(x)) \rightarrow (P \vee \forall x\ Q(x))$$
$$\exists xy\ (P(x, y) \rightarrow \forall zw\ P(z, w))$$

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 standard modal logic, $\Box\Box A$ is equivalent to $\Box A$.