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

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

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

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

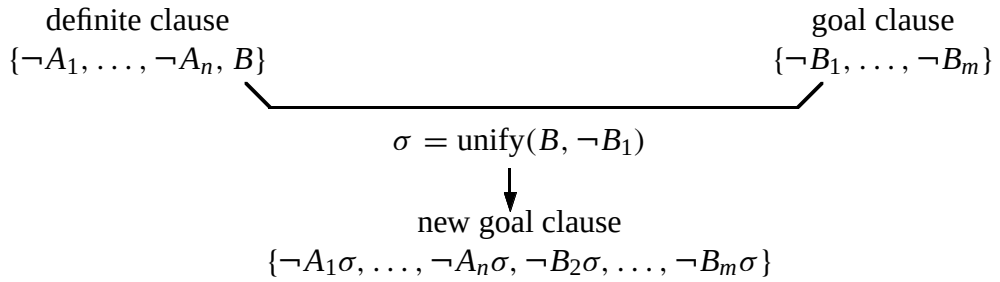
A Prolog database consists of definite clauses. Observe that definite clauses cannot express negative assertions, since they must contain a positive literal. From a mathematical point of view, they have little expressive power; every set of definite clauses is consistent! Even so, definite clauses are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

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

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

{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.

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

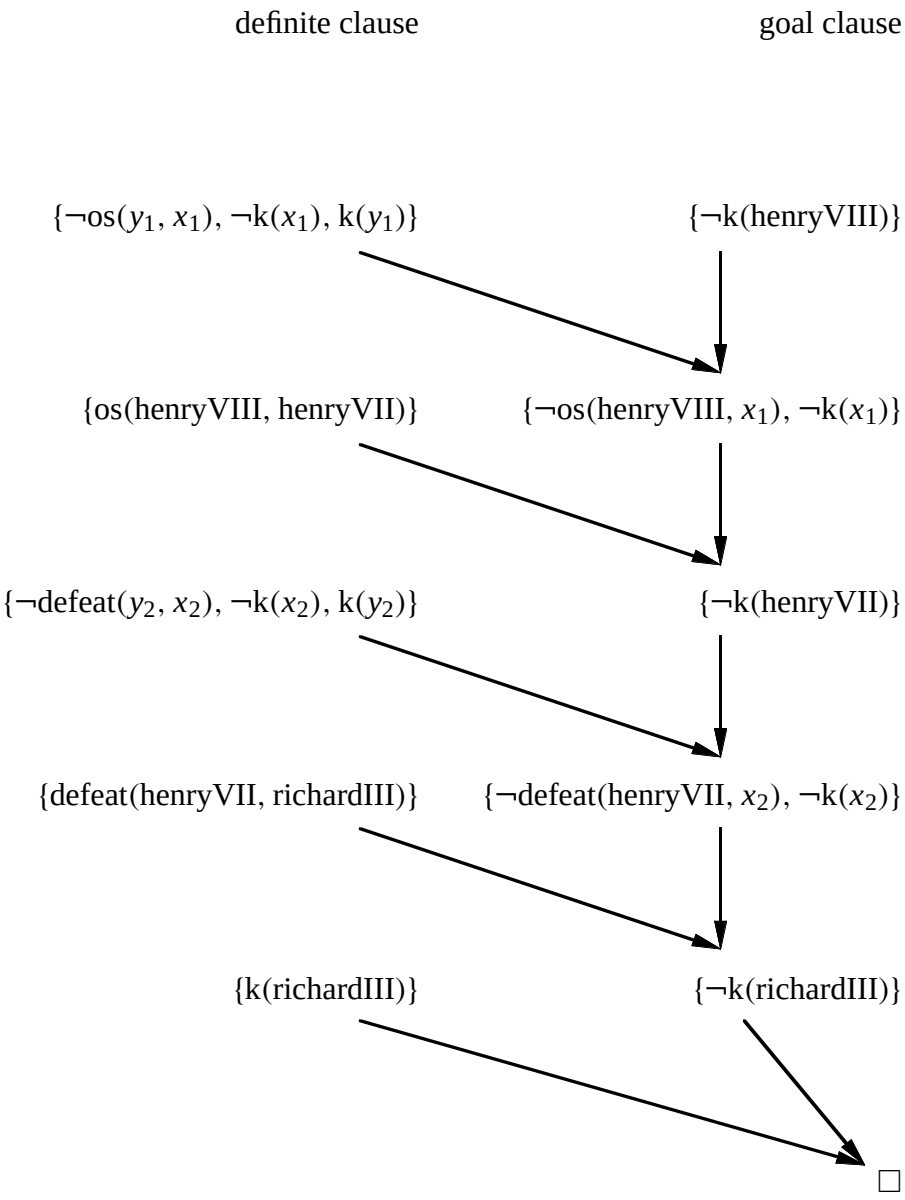


Figure 2: Execution of a Prolog program (os = oldestson, k = king)

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 $\Box A$.