EXERCISES 9

INFERENCE IN FIRST-ORDER LOGIC

9.1 Propositional vs. First-Order Inference

Exercise 9.UNIP

Prove that Universal Instantiation is sound and that Existential Instantiation produces an inferentially equivalent knowledge base.

We want to show that any sentence of the form $\forall v \;\; \alpha$ entails any universal instantiation of the sentence. The sentence $\forall v \;\; \alpha$ asserts that α is true in all possible extended interpretations. For any model specifying the referent of ground term g, the truth of $\textsc{Subst}(\{v/g\}, \alpha)$ must be identical to the truth value of some extended interpretation in which v is assigned to an object, and in all such interpretations α is true.

EI states: for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)} .$$

If the knowledge base with the original existentially quantified sentence is KB and the result of EI is KB', then we need to prove that KB is satisfiable iff KB' is satisfiable. Forward: if KB is satisfiable, it has a model M for which an extended interpretation assigning v to some object o renders α true. Hence, we can construct a model M' that satisfies KB' by assigning k to refer to o; since k does not appear elsewhere, the truth values of all other sentences are unaffected. Backward: if KB' is satisfiable, it has a model M' with an assignment for k to some object o. Hence, we can construct a model M that satisfies KB with an extended interpretation assigning v to o; since k does not appear elsewhere, removing it from the model leaves the truth values of all other sentences are unaffected.

Exercise 9.EXIN

From $\mathit{Likes}(\mathit{Jerry}, \mathit{IceCream})$ it seems reasonable to infer $\exists x \;\; \mathit{Likes}(x, \mathit{IceCream})$. Write down a general inference rule, **Existential Introduction**, that sanctions this inference. State carefully the conditions that must be satisfied by the variables and terms involved.

For any sentence α containing a ground term g and for any variable v not occuring in α, we have

$$\frac{\alpha}{\exists v \ \ \textsc{Subst}^*(\{g/v\}, \alpha)}$$

where $\textsc{Subst}^*$ is a function that substitutes any or all of the occurrences of g with v. Notice that substituting just one occurrence and applying the rule multiple times is not the same, because it results in a weaker conclusion. For example, $P(a, a)$ should entail $\exists x \ \ P(x, x)$ rather than the weaker $\exists x, y \ \ P(x, y)$.

Exercise 9.KBOS

Suppose a knowledge base contains just one sentence, $\exists x \ \ AsHighAs(x, Everest)$. Which of the following are legitimate results of applying Existential Instantiation?

a. $AsHighAs(Everest, Everest)$.

b. $AsHighAs(Kilimanjaro, Everest)$.

c. $AsHighAs(Kilimanjaro, Everest) \land AsHighAs(BenNevis, Everest)$ (after two applications).

Both b and c are sound conclusions; a is unsound because it introduces the previously-used symbol *Everest*. Note that c does not imply that there are two mountains as high as Everest, because nowhere is it stated that *BenNevis* is different from *Kilimanjaro* (or *Everest*, for that matter).

9.2 Unification and First-Order Inference

Exercise 9.MGUX

For each pair of atomic sentences, give the most general unifier if it exists:

a. $P(A, B, B)$, $P(x, y, z)$.

b. $Q(y, G(A, B))$, $Q(G(x, x), y)$.

c. $Older(Father(y), y)$, $Older(Father(x), John)$.

d. $Knows(Father(y), y)$, $Knows(x, x)$.

This is an easy exercise to check that the student understands unification.

a. $\{x/A, y/B, z/B\}$ (or some permutation of this).

b. No unifier (x cannot bind to both A and B).

c. $\{y/John, x/John\}$.

d. No unifier (because the occurs-check prevents unification of y with $Father(y)$).

Exercise 9.SUBL

Consider the subsumption lattices shown in Figure 9.2 (page 304).

a. Construct the lattice for the sentence $Employs(Mother(John), Father(Richard))$.

b. Construct the lattice for the sentence $Employs(IBM, y)$ ("Everyone works for IBM"). Remember to include every kind of query that unifies with the sentence.

c. Assume that STORE indexes each sentence under every node in its subsumption lattice. Explain how FETCH should work when some of these sentences contain variables; use as examples the sentences in (a) and (b) and the query $Employs(x, Father(x))$.

a. For the sentence $Employs(Mother(John), Father(Richard))$, the page isn't wide enough to draw the diagram as in Figure 9.2, so we will draw it with indentation denoting children nodes:

```
[1] Employs(x, y)
  [2] Employs(x, Father(z))
    [3] Employs(x, Father(Richard))
      [4] Employs(Mother(w), Father(Richard))
        [5] Employs(Mother(John), Father(Richard))
    [6] Employs(Mother(w), Father(z))
      [4] ...
      [7] Employs(Mother(John), Father(z))
        [5] ...
  [8] Employs(Mother(w), y)
    [9] Employs(Mother(John), y)
      [10] Employs(Mother(John), Father(z)
        [5] ...
    [6] ...
```

b. For the sentence $Employs(IBM, y)$, the lattice contains $Employs(x, y)$ and $Employs(y, y)$.

c.

Exercise 9.FOLH

Write down logical representations for the following sentences, suitable for use with Generalized Modus Ponens:

a. Horses, cows, and pigs are mammals.

b. An offspring of a horse is a horse.

c. Bluebeard is a horse.

d. Bluebeard is Charlie's parent.

e. Offspring and parent are inverse relations.

f. Every mammal has a parent.

We use a very simple ontology to make the examples easier:

a. $Horse(x) \Rightarrow Mammal(x)$
$Cow(x) \Rightarrow Mammal(x)$
$Pig(x) \Rightarrow Mammal(x)$.

b. $\mathit{Offspring}(x, y) \land Horse(y) \Rightarrow Horse(x)$.

c. $Horse(Bluebeard)$.

d. $Parent(Bluebeard, Charlie)$.

e. $\mathit{Offspring}(x, y) \Rightarrow Parent(y, x)$
$Parent(x, y) \Rightarrow \mathit{Offspring}(y, x)$.
(Note we couldn't do $\mathit{Offspring}(x, y) \Leftrightarrow Parent(y, x)$ because that is not in the form expected by Generalized Modus Ponens.)

f. $Mammal(x) \Rightarrow Parent(G(x), x)$ (here G is a Skolem function).

Exercise 9.CENS

Suppose we put into a logical knowledge base a segment of the U.S. census data listing the age, city of residence, date of birth, and mother of every person, using social security numbers as identifying constants for each person. Thus, George's age is given by $Age(\text{443-65-1282}, 56)$. Which of the following indexing schemes S1–S5 enable an efficient solution for which of the queries Q1–Q4 (assuming normal backward chaining)?

- **S1**: an index for each atom in each position.
- **S2**: an index for each first argument.
- **S3**: an index for each predicate atom.
- **S4**: an index for each *combination* of predicate and first argument.
- **S5**: an index for each *combination* of predicate and second argument and an index for each first argument.

- **Q1**: $Age(\text{443-44-4321}, x)$
- **Q2**: $ResidesIn(x, Houston)$
- **Q3**: $Mother(x, y)$
- **Q4**: $Age(x, 34) \land ResidesIn(x, TinyTownUSA)$

We will give the average-case time complexity for each query/scheme combination in the following table. (An entry of the form "1; n" means that it is $O(1)$ to find the first solution to the query, but $O(n)$ to find them all.) We make the following assumptions: hash tables give $O(1)$ access; there are n people in the data base; there are $O(n)$ people of any specified age; every person has one mother; there are H people in Houston and T people in Tiny Town; T is much less than n; in Q4, the second conjunct is evaluated first.

	Q1	Q2	Q3	Q4
S1	1	$1;\ H$	$1;\ n$	$T;\ T$
S2	1	$n;\ n$	$1;\ n$	$n;\ n$
S3	n	$n;\ n$	$1;\ n$	$n^2;\ n^2$
S4	1	$n;\ n$	$1;\ n$	$n;\ n$
S5	1	$1;\ H$	$1;\ n$	$T;\ T$

Anything that is $O(1)$ can be considered "efficient," as perhaps can anything $O(T)$. Note that S1 and S5 dominate the other schemes for this set of queries. Also note that indexing on predicates plays no role in this table (except in combination with an argument), because there are only 3 predicates (which is $O(1)$). It would make a difference in terms of the constant factor.

9.3 Forward Chaining

Exercise 9.CCSP

Explain how to write any given 3-SAT problem of arbitrary size using a single first-order definite clause and no more than 30 atomic sentences.

Consider a 3-SAT problem of the form

$$(x_{1,1} \lor x_{2,1} \lor \neg x_{3,1}) \land (\neg x_{1,2} \lor x_{2,2} \lor x_{3,2}) \lor \ldots$$

We want to rewrite this as a single definite clause of the form

$$A \land B \land C \land \ldots \;\Rightarrow\; Z,$$

along with a few ground clauses. We can do that with the definite clause

$$OneOf(x_{1,1}, x_{2,1}, Not(x_{3,1})) \land OneOf(Not(x_{1,2}), x_{2,2}, x_{3,2}) \land \ldots \;\Rightarrow\; Solved\,.$$

The key is that any solution to the definite clause has to assign the same value to each occurrence of any given variable, even if the variable is negated in some of the SAT clauses but not others. We also need to define $OneOf$. This can be done concisely as follows:

$$\begin{array}{l} OneOf(True, x, y) \\ OneOf(x, True, y) \\ OneOf(x, y, True) \\ OneOf(Not(False), x, y) \\ OneOf(x, Not(False), y) \\ OneOf(x, y, Not(False)) \end{array}$$

9.4 Backward Chaining

Exercise 9.HRKB

This question considers Horn KBs, such as the following:

$$P(F(x)) \Rightarrow P(x)$$
$$Q(x) \Rightarrow P(F(x))$$
$$P(A)$$
$$Q(B)$$

Let FC be a breadth-first forward-chaining algorithm that repeatedly adds all consequences of currently satisfied rules; let BC be a depth-first left-to-right backward-chaining algorithm that tries clauses in the order given in the KB. Which of the following are true?

a. FC will infer the literal $Q(A)$.

b. FC will infer the literal $P(B)$.

c. If FC has failed to infer a given literal, then it is not entailed by the KB.

d. BC will return *true* given the query $P(B)$.

e. If BC does not return *true* given a query literal, then it is not entailed by the KB.

a. False; $Q(A)$ is not entailed.

b. True; via $P(F(B))$.

c. True; breadth-first FC is complete for Horn KBs.

d. False; infinite loop applying the first rule repeatedly.

e. False; $P(b)$ is an example.

Exercise 9.BCAR

Suppose you are given the following axioms:

1. $0 \leq 3$.
2. $7 \leq 9$.
3. $\forall x \quad x \leq x$.
4. $\forall x \quad x \leq x + 0$.
5. $\forall x \quad x + 0 \leq x$.
6. $\forall x, y \quad x + y \leq y + x$.
7. $\forall w, x, y, z \quad w \leq y \land x \leq z \;\Rightarrow\; w + x \leq y + z$.
8. $\forall x, y, z \quad x \leq y \land y \leq z \;\Rightarrow\; x \leq z$

a. Give a backward-chaining proof of the sentence $7 \leq 3 + 9$. (Be sure, of course, to use only the axioms given here, not anything else you may know about arithmetic.) Show only the steps that leads to success, not the irrelevant steps.

b. Give a forward-chaining proof of the sentence $7 \leq 3 + 9$. Again, show only the steps that lead to success.

This is quite tricky but students should be able to manage if they check each step carefully.

a. (Note: At each resolution, we rename the variables in the rule).

Goal G0: $7 \le 3+9$	Resolve with (8) $\{x1/7, z1/3+9\}$.
Goal G1: $7 \le y1$	Resolve with (4) $\{x2/7, y1/7+0\}$. Succeeds.
Goal G2: $7+0 \le 3+9$.	Resolve with (8) $\{x3/7+0, z3/3+9\}$
Goal G3: $7+0 \le y3$	Resolve with (6) $\{x4/7, y4/0, y3/0+7\}$ Succeeds.
Goal G4: $0+7 \le 3+9$	Resolve with (7) $\{w5/0, x5/7, y5/3, z5/9\}$.
Goal G5: $0 \le 3$.	Resolve with (1). Succeeds.
Goal G6: $7 \le 9$.	Resolve with (2). Succeeds.
G4 succeeds	
G2 succeeds.	
G0 succeeds.	

b. From (1),(2), (7) $\{w/0, x/7, y/3, z/9\}$ infer
(9) $0+7 \le 3+9$.
From (9), (6), (8) $\{x1/0, y1/7, x2/0+7, y2/7+0, z2/3+9\}$ infer
(10) $7+0 \le 3+9$.
($x1, y1$ are renamed variables in (6). $x2, y2, z2$ are renamed variables in (8).)
From (4), (10), (8) $\{x3/7, x4/7, y4/7+0, z4/3+9\}$ infer
(11) $7 \le 3+9$.
($x3$ is a renamed variable in (4). $x4, y4, z4$ are renamed variables in (8).)

Exercise 9.STDF
One might suppose that we can avoid the problem of variable conflict in unification during backward chaining by standardizing apart all of the sentences in the knowledge base once and for all. Show that, for some sentences, this approach cannot work. (*Hint*: Consider a sentence in which one part unifies with another.)

This would work if there were no recursive rules in the knowledge base. But suppose the knowledge base contains the sentences:

$$Member(x, [x|r])$$
$$Member(x, r) \;\Rightarrow\; Member(x, [y|r])$$

Now take the query $Member(3, [1, 2, 3])$, with a backward chaining system. We unify the query with the consequent of the implication to get the substitution $\theta = \{x/3, y/1, r/[2, 3]\}$. We then substitute this in to the left-hand side to get $Member(3, [2, 3])$ and try to back chain on that with the substitution θ. When we then try to apply the implication again, we get a failure because y cannot be both 1 and 2. In other words, the failure to standardize apart causes failure in some cases where recursive rules would result in a solution if we did standardize apart.

Exercise 9.HORB

In this exercise, use the sentences you wrote in Exercise 9.FOLH to answer a question by using a backward-chaining algorithm.

a. Draw the proof tree generated by an exhaustive backward-chaining algorithm for the query $\exists\, h \;\; Horse(h)$, where clauses are matched in the order given.

b. What do you notice about this domain?

c. How many solutions for h actually follow from your sentences?

d. Can you think of a way to find all of them? (*Hint*: See Smith *et al.* (1986).)

This questions deals with the subject of looping in backward-chaining proofs. A loop is bound to occur whenever a subgoal arises that is a substitution instance of one of the goals on the stack. Not all loops can be caught this way, of course, otherwise we would have a way to solve the halting problem.

a. The proof tree is shown in Figure S9.1. The branch with $\mathit{Offspring}(\mathit{Bluebeard}, y)$ and $Parent(y, Bluebeard)$ repeats indefinitely, so the rest of the proof is never reached.

b. We get an infinite loop because of rule **b**, $\mathit{Offspring}(x, y) \land Horse(y) \;\Rightarrow\; Horse(x)$. The specific loop appearing in the figure arises because of the ordering of the clauses—it would be better to order $Horse(Bluebeard)$ before the rule from **b**. However, a loop will occur no matter which way the rules are ordered if the theorem-prover is asked for all solutions.

c. One should be able to prove that both Bluebeard and Charlie are horses.

d. Smith *et al.* 1986 recommend the following method. Whenever a "looping" goal occurs (one that is a substitution instance of a supergoal higher up the stack), suspend the attempt to prove that subgoal. Continue with all other branches of the proof for the supergoal, gathering up the solutions. Then use those solutions (suitably instantiated if necessary) as solutions for the suspended subgoal, continuing that branch of the proof to find additional solutions if any. In the proof shown in the figure, the $\mathit{Offspring}(\mathit{Bluebeard}, y)$ is a repeated goal and would be suspended. Since no other way to prove it exists, that branch will terminate with failure. In this case, Smith's method is sufficient to allow the theorem-prover to find both solutions.

Exercise 9.BCCR

Trace the execution of the backward-chaining algorithm in Figure 9.6 (page 311) when it is applied to solve the crime problem (page 305). Show the sequence of values taken on by the $goals$ variable, and arrange them into a tree.

Here is a goal tree:

```
goals = [Criminal(West)]
  goals = [American(West), Weapon(y), Sells(West, y, z), Hostile(z)]
  goals = [Weapon(y), Sells(West, y, z), Hostile(z)]
```

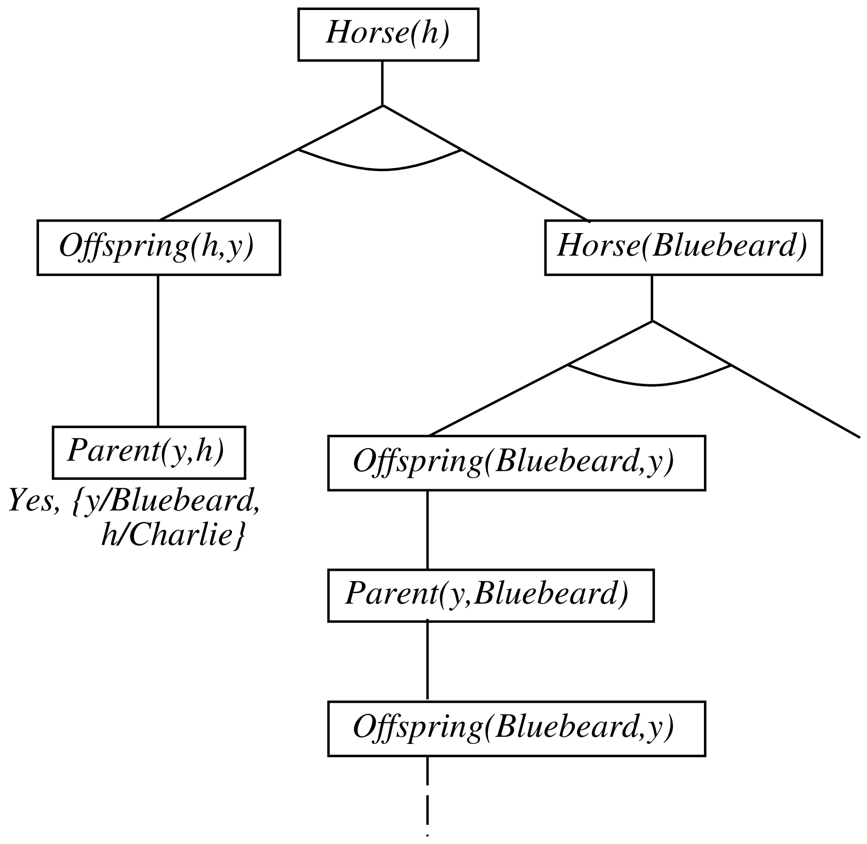

Figure S9.1 Partial proof tree for finding horses.

```
    goals = [Missle(y), Sells(West, y, z), Hostile(z)]
    goals = [Sells(West, M1, z), Hostile(z)]
      goals = [Missle(M1), Owns(Nono, M1), Hostile(Nono)]
      goals = [Owns(Nono, M1), Hostile(Nono)]
    goals = [Hostile(Nono)]
goals = []
```

Exercise 9.PRLC

The following Prolog code defines a predicate P. (Remember that uppercase terms are variables, not constants, in Prolog.)

```
P(X,[X|Y]).
P(X,[Y|Z]) :- P(X,Z).
```

a. Show proof trees and solutions for the queries `P(A,[2,1,3])` and `P(2,[1,A,3])`.

b. What standard list operation does `P` represent?

a. In the following, an indented line is a step deeper in the proof tree, while two lines at the same indentation represent two alternative ways to prove the goal that is unindented above it. `P1` and `P2` refer to the first and second clauses of the definition respectively. We show each goal as it is generated and the result of unifying it with the head of each clause.

```
P(A, [2,1,3])                   goal
  P(2, [2|[1,3]])               unify with head of P1
    => solution, with A = 2
  P(A, [2|[1,3]])               unify with head of P2
    P(A, [1,3])                 subgoal
      P(1, [1,3])               unify with head of P1
        => solution, with A = 1
      P(A, [1|[3]])             unify with head of P2
        P(A, [3])               subgoal
          P(3, [3|[]])          unify with head of P1
            => solution, with A = 3
          P(A, [3|[]])          unify with head of P2
            P(A, [])            subgoal (fails)

P(2, [1,A,3])                   goal
  P(2, [1|[A,3]])               unify with head of P2
    P(2, [A,3])                 subgoal
      P(2, [2,3])               unify with head of P1
        => solution, with A = 2
      P(2, [A|[3]])             unify with head of P2
        P(2, [3])               subgoal
          P(2, [3|[]])          unify with head of P2
            P(2, [])            subgoal
```

b. `P` could better be called `Member`; it succeeds when the first argument is an element of the list that is the second argument.

Exercise 9.PRLS

This exercise looks at sorting in Prolog.

a. Write Prolog clauses that define the predicate `sorted(L)`, which is true if and only if list `L` is sorted in ascending order.

b. Write a Prolog definition for the predicate `perm(L,M)`, which is true if and only if `L` is a permutation of `M`.

c. Define `sort(L,M)` (`M` is a sorted version of `L`) using `perm` and `sorted`.

d. Run `sort` on longer and longer lists until you lose patience. What is the time complexity of your program?

e. Write a faster sorting algorithm, such as insertion sort or quicksort, in Prolog.

The different versions of `sort` illustrate the distinction between logical and procedural semantics in Prolog.

a.
```
sorted([]).
sorted([X]).
sorted([X,Y|L]) :- X<Y, sorted([Y|L]).
```

b.
```
perm([],[]).
perm([X|L],M) :-
  delete(X,M,M1),
  perm(L,M1).

delete(X,[X|L],L).       %% deleting an X from [X|L] yields L
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

member(X,[X|L]).
member(X,[_|L]) :- member(X,L).
```

c. `sort(L,M) :- perm(L,M), sorted(M).`
This is about as close to an executable formal specification of sorting as you can get—it says the absolute minimum about what sort means: in order for `M` to be a sort of `L`, it must have the same elements as `L`, and they must be in order.

d. Unfortunately, this doesn't fare as well as a program as it does as a specification. It is a generate-and-test sort: `perm` generates candidate permutations one at a time, and `sorted` tests them. In the worst case (when there is only one sorted permutation, and it is the last one generated), this will take $O(n!)$ generations. Since each `perm` is $O(n^2)$ and each `sorted` is $O(n)$, the whole `sort` is $O(n!n^2)$ in the worst case.

e. Here's a simple insertion sort, which is $O(n^2)$:

```
isort([],[]).
isort([X|L],M) :- isort(L,M1), insert(X,M1,M).

insert(X,[],[X]).
insert(X,[Y|L],[X,Y|L]) :- X=<Y.
insert(X,[Y|L],[Y|M]) :- Y<X, insert(X,L,M).
```

Exercise 9.DFSM

This exercise looks at the recursive application of rewrite rules, using logic programming. A rewrite rule (or **demodulator** in OTTER terminology) is an equation with a specified direction. For example, the rewrite rule $x + 0 \rightarrow x$ suggests replacing any expression that matches $x + 0$ with the expression x. Rewrite rules are a key component of equational reasoning systems. Use the predicate `rewrite(X,Y)` to represent rewrite rules. For example, the earlier rewrite rule is written as `rewrite(X+0,X)`. Some terms are *primitive* and cannot be further simplified; thus, we write `primitive(0)` to say that 0 is a primitive term.

a. Write a definition of a predicate `simplify(X,Y)`, that is true when `Y` is a simplified version of `X`—that is, when no further rewrite rules apply to any subexpression of `Y`.

b. Write a collection of rules for the simplification of expressions involving arithmetic operators, and apply your simplification algorithm to some sample expressions.

c. Write a collection of rewrite rules for symbolic differentiation, and use them along with your simplification rules to differentiate and simplify expressions involving arithmetic expressions, including exponentiation.

This exercise illustrates the power of pattern-matching, which is built into Prolog.

a. The code for simplification looks straightforward, but students may have trouble finding the middle way between undersimplifying and looping infinitely.

```
simplify(X,X) :- primitive(X).
simplify(X,Y) :- evaluable(X), Y is X.
simplify(Op(X)) :- simplify(X,X1), simplify_exp(Op(X1)).
simplify(Op(X,Y)) :- simplify(X,X1), simplify(Y,Y1), simplify_exp(Op(X1,Y1)).

simplify_exp(X,Y) :- rewrite(X,X1), simplify(X1,Y).
simplify_exp(X,X).

primitive(X) :- atom(X).
```

b. Here are a few representative rewrite rules drawn from the extensive list in Norvig (1992).

```
Rewrite(X+0,X).
Rewrite(0+X,X).
Rewrite(X+X,2*X).
Rewrite(X*X,X^2).
Rewrite(X^0,1).
Rewrite(0^X,0).
Rewrite(X*N,N*X) :- number(N).
Rewrite(ln(e^X),X).
Rewrite(X^Y*X^Z,X^(Y+Z)).
Rewrite(sin(X)^2+cos(X)^2,1).
```

c. Here are the rules for differentiation, using `d(Y,X)` to represent the derivative of expression `Y` with respect to variable `X`.

```
Rewrite(d(X,X),1).
Rewrite(d(U,X),0) :- atom(U), U /= X.
Rewrite(d(U+V,X),d(U,X)+d(V,X)).
Rewrite(d(U-V,X),d(U,X)-d(V,X)).
Rewrite(d(U*V,X),V*d(U,X)+U*d(V,X)).
Rewrite(d(U/V,X),(V*d(U,X)-U*d(V,X))/(V^2)).
Rewrite(d(U^N,X),N*U^(N-1)*d(U,X)) :- number(N).
Rewrite(d(log(U),X),d(U,X)/U).
Rewrite(d(sin(U),X),cos(U)*d(U,X)).
Rewrite(d(cos(U),X),-sin(U)*d(U,X)).
Rewrite(d(e^U,X),d(U,X)*e^U).
```

Exercise 9.PRSE

This exercise considers the implementation of search algorithms in Prolog. Suppose that `successor(X,Y)` is true when state `Y` is a successor of state `X`; and that `goal(X)` is true when `X` is a goal state. Write a definition for `solve(X,P)`, which means that `P` is a path (list of states) beginning with `X`, ending in a goal state, and consisting of a sequence of legal steps as defined by `successor`. You will find that depth-first search is the easiest way to do this. How easy would it be to add heuristic search control?

Once you understand how Prolog works, the answer is easy:

```
solve(X,[X]) :- goal(X).
solve(X,[X|P]) :- successor(X,Y), solve(Y,P).
```

We could render this in English as “Given a start state, if it is a goal state, then the path consisting of just the start state is a solution. Otherwise, find some successor state such that there is a path from the successor to the goal; then a solution is the start state followed by that path."

Notice that `solve` can not only be used to find a path `P` that is a solution, it can also be used to verify that a given path is a solution.

If you want to add heuristics (or even breadth-first search), you need an explicit queue. The algorithms become quite similar to the versions written in Lisp or Python or Java or in pseudo-code in the book.

9.5 Resolution

Exercise 9.SKOL

These questions concern concern issues with substitution and Skolemization.

a. Given the premise $\forall x\ \exists y\ \ P(x, y)$, it is not valid to conclude that $\exists q\ \ P(q, q)$. Give an example of a predicate P where the first is true but the second is false.

b. Suppose that an inference engine is incorrectly written with the occurs check omitted, so that it allows a literal like $P(x, F(x))$ to be unified with $P(q, q)$. (As mentioned, most standard implementations of Prolog actually do allow this.) Show that such an inference engine will allow the conclusion $\exists y\ \ P(q, q)$ to be inferred from the premise $\forall x\ \exists y\ \ P(x, y)$.

c. Suppose that a procedure that converts first-order logic to clausal form incorrectly Skolemizes $\forall x\ \exists y\ \ P(x, y)$ to $P(x, Sk0)$—that is, it replaces y by a Skolem constant rather than by a Skolem function of x. Show that an inference engine that uses such a procedure will likewise allow $\exists q\ \ P(q, q)$ to be inferred from the premise $\forall x\ \exists y\ \ P(x, y)$.

d. A common error among students is to suppose that, in unification, one is allowed to substitute a term for a Skolem constant instead of for a variable. For instance, they will say that the formulas $P(Sk1)$ and $P(A)$ can be unified under the substitution $\{Sk1/A\}$. Give an example where this leads to an invalid inference.

a. Let $P(x, y)$ be the relation "x is less than y" over the integers. Then $\forall x\ \exists y\ \ P(x, y)$ is true but $\exists x\ \ P(x, x)$ is false.

b. Converting the premise to clausal form gives $P(x, Sk0(x))$ and converting the negated goal to clausal form gives $\neg P(q, q)$. If the two formulas can be unified, then these resolve to the null clause.

c. If the premise is represented as $P(x, Sk0)$ and the negated goal has been correctly converted to the clause $\neg P(q, q)$ then these can be resolved to the null clause under the substitution $\{q/Sk0, x/Sk0\}$.

d. Suppose you are given the premise $\exists x\ \ Cat(x)$ and you wish to prove $Cat(Socrates)$. Converting the premise to clausal form gives the clause $Cat(Sk1)$. If this unifies with $Cat(Socrates)$ then you can resolve this with the negated goal $\neg Cat(Socrates)$ to give the null clause.

Exercise 9.BRSI

A popular children's riddle is "Brothers and sisters have I none, but that man's father is my father's son." Use the rules of the family domain (Section 8.3.2 on page 284) to show who that man is. You may apply any of the inference methods described in this chapter. Why do you think that this riddle is difficult?

Surprisingly, the hard part to represent is "who is that man." We want to ask "what relationship does that man have to some known person," but if we represent relations with predicates (e.g., $Parent(x, y)$) then we cannot make the relationship be a variable in first-order logic. So instead we need to reify relationships. We will use $Rel(r, x, y)$ to say that the family relationship r holds between people x and y. Let Me denote me and MrX denote "that man." We will also need the Skolem constants FM for the father of Me and FX for the father of MrX. The facts of the case (put into implicative normal form) are:

(1) $Rel(Sibling, Me, x) \;\Rightarrow\; False$
(2) $Male(MrX)$
(3) $Rel(Father, FX, MrX)$
(4) $Rel(Father, FM, Me)$
(5) $Rel(Son, FX, FM)$

We want to be able to show that Me is the only son of my father, and therefore that Me is father of MrX, who is male, and therefore that "that man" is my son. The relevant definitions from the family domain are:

(6) $Rel(Parent, x, y) \land Male(x) \;\Leftrightarrow\; Rel(Father, x, y)$
(7) $Rel(Son, x, y) \;\Leftrightarrow\; Rel(Parent, y, x) \land Male(x)$
(8) $Rel(Sibling, x, y) \;\Leftrightarrow\; x \neq y \land \exists p \;\; Rel(Parent, p, x) \land Rel(Parent, p, y)$
(9) $Rel(Father, x_1, y) \land Rel(Father, x_2, y) \;\Rightarrow\; x_1 = x_2$

and the query we want is:

(Q) $Rel(r, MrX, y)$

We want to be able to get back the answer $\{r/Son, y/Me\}$. Translating 1-9 and Q into INF (and negating Q and including the definition of $\neq$) we get:

$(6a)$ $Rel(Parent,x,y) \wedge Male(x) \;\Rightarrow\; Rel(Father,x,y)$
$(6b)$ $Rel(Father,x,y) \;\Rightarrow\; Male(x)$
$(6c)$ $Rel(Father,x,y) \;\Rightarrow\; Rel(Parent,x,y)$
$(7a)$ $Rel(Son,x,y) \;\Rightarrow\; Rel(Parent,y,x)$
$(7b)$ $Rel(Son,x,y) \;\Rightarrow\; Male(x))$
$(7c)$ $Rel(Parent,y,x) \wedge Male(x) \;\Rightarrow\; Rel(Son,x,y)$
$(8a)$ $Rel(Sibling,x,y) \;\Rightarrow\; x \neq y$
$(8b)$ $Rel(Sibling,x,y) \;\Rightarrow\; Rel(Parent,P(x,y),x)$
$(8c)$ $Rel(Sibling,x,y) \;\Rightarrow\; Rel(Parent,P(x,y),y)$
$(8d)$ $Rel(Parent,P(x,y),x) \wedge Rel(Parent,P(x,y),y) \wedge x \neq y \;\Rightarrow\; Rel(Sibling,x,y)$
(9) $Rel(Father,x_1,y) \wedge Rel(Father,x_2,y) \;\Rightarrow\; x_1 = x_2$
(N) $True \;\Rightarrow\; x = y \vee x \neq y$
(N') $x = y \wedge x \neq y \;\Rightarrow\; False$
(Q') $Rel(r,MrX,y) \;\Rightarrow\; False$

Note that (1) is non-Horn, so we will need resolution to be be sure of getting a solution. It turns out we also need demodulation to deal with equality. The following lists the steps of the proof, with the resolvents of each step in parentheses:

(10) $Rel(Parent,FM,Me)$	$(4,6c)$
(11) $Rel(Parent,FM,FX)$	$(5,7a)$
(12) $Rel(Parent,FM,y) \wedge Me \neq y \;\Rightarrow\; Rel(Sibling,Me,y)$	$(10,8d)$
(13) $Rel(Parent,FM,y) \wedge Me \neq y \;\Rightarrow\; False$	$(12,1)$
(14) $Me \neq FX \;\Rightarrow\; False$	$(13,11)$
(15) $Me = FX$	$(14,N)$
(16) $Rel(Father,Me,MrX)$	$(15,3,demodulation)$
(17) $Rel(Parent,Me,MrX)$	$(16,6c)$
(18) $Rel(Son,MrX,Me)$	$(17,2,7c)$
(19) $False\ \{r/Son,y/Me\}$	$(18,Q')$

Exercise 9.ANCH

Suppose a knowledge base contains just the following first-order Horn clauses:

$$Ancestor(Mother(x),x)$$
$$Ancestor(x,y) \wedge Ancestor(y,z) \;\Rightarrow\; Ancestor(x,z)$$

Consider a forward chaining algorithm that, on the jth iteration, terminates if the KB contains a sentence that unifies with the query, else adds to the KB every atomic sentence that can be inferred from the sentences already in the KB after iteration $j-1$.

a. For each of the following queries, say whether the algorithm will (1) give an answer (if so, write down that answer); or (2) terminate with no answer; or (3) never terminate.

(i) $Ancestor(Mother(y), John)$
(ii) $Ancestor(Mother(Mother(y)), John)$
(iii) $Ancestor(Mother(Mother(Mother(y))), Mother(y))$
(iv) $Ancestor(Mother(John), Mother(Mother(John)))$

b. Can a resolution algorithm prove the sentence $\neg Ancestor(John, John)$ from the original knowledge base? Explain how, or why not.

c. Suppose we add the assertion that $\neg(Mother(x)=x)$ and augment the resolution algorithm with inference rules for equality. Now what is the answer to (b)?

a. Results from forward chaining:

(i) $Ancestor(Mother(y), John)$: Yes, $\{y/John\}$ (immediate).
(ii) $Ancestor(Mother(Mother(y)), John)$: Yes, $\{y/John\}$ (second iteration).
(iii) $Ancestor(Mother(Mother(Mother(y))), Mother(y))$: Yes, $\{\}$ (second iteration).
(iv) $Ancestor(Mother(John), Mother(Mother(John)))$: Does not terminate.

b. Although resolution is complete, it cannot prove this because it does not follow. Nothing in the axioms rules out the possibility of everything being the ancestor of everything else.

c. Same answer.

Exercise 9.SHAV

Let $\mathcal{L}$ be the first-order language with a single predicate $S(p,q)$, meaning "p shaves q." Assume a domain of people.

a. Consider the sentence "There exists a person P who shaves every one who does not shave themselves, and only people that do not shave themselves." Express this in $\mathcal{L}$.

b. Convert the sentence in (a) to clausal form.

c. Construct a resolution proof to show that the clauses in (b) are inherently inconsistent. (Note: you do not need any additional axioms.)

a. $\exists p\ \forall q\ \ S(p,q)\ \Leftrightarrow\ \neg S(q,q)$.

b. There are two clauses, corresponding to the two directions of the implication.
C1: $\neg S(Sk1,q) \vee \neg S(q,q)$.
C2: $S(Sk1,q) \vee S(q,q)$.

c. Applying factoring to C1, using the substitution $q/Sk1$ gives:
C3: $\neg S(Sk1,Sk1)$.
Applying factoring to C2, using the substitution $q/Sk1$ gives:
C4: $S(Sk1,Sk1)$.
Resolving C3 with C4 gives the null clause.

Exercise 9.RESV

How can resolution be used to show that a sentence is valid? Unsatisfiable?

This question tests both the student's understanding of resolution and their ability to think at a high level about relations among sets of sentences. Recall that resolution allows one to show that $KB \models \alpha$ by proving that $KB \land \neg\alpha$ is inconsistent. Suppose that in general the resolution system is called using $\textsc{Ask}(KB, \alpha)$. Now we want to show that a given sentence, say β is valid or unsatisfiable.

A sentence β is valid if it can be shown to be true without additional information. We check this by calling $\textsc{Ask}(KB_0, \beta)$ where KB_0 is the empty knowledge base.

A sentence β that is unsatisfiable is inconsistent by itself. So if we empty the knowledge base again and call $\textsc{Ask}(KB_0, \neg\beta)$ the resolution system will attempt to derive a contradiction starting from $\neg\neg\beta$. If it can do so, then it must be that $\neg\neg\beta$, and hence β, is inconsistent.

Exercise 9.REST

Construct an example of two clauses that can be resolved together in two different ways giving two different outcomes.

There are two ways to do this: one literal in one clause that is complementary to two different literals in the other, such as

$$P(x) \quad \neg P(a) \lor \neg P(b)$$

or two complementary pairs of literals, such as

$$P(x) \lor Q(x) \quad \neg P(a) \lor \neg Q(b) .$$

Note that this does not work in propositional logic: in the first case, the two literals in the second clause would have to be identical; in the second case, the remaining unresolved complementary pair after resolution would render the result a tautology.

Exercise 9.HORA

From "Horses are animals," it follows that "The head of a horse is the head of an animal." Demonstrate that this inference is valid by carrying out the following steps:

a. Translate the premise and the conclusion into the language of first-order logic. Use three predicates: $HeadOf(h, x)$ (meaning "h is the head of x"), $Horse(x)$, and $Animal(x)$.

b. Negate the conclusion, and convert the premise and the negated conclusion into conjunctive normal form.

c. Use resolution to show that the conclusion follows from the premise.

This is a form of inference used to show that Aristotle's syllogisms could not capture all sound inferences.

a. $\forall x \;\; Horse(x) \;\Rightarrow\; Animal(x)$
$\forall x, h \;\; Horse(x) \land HeadOf(h, x) \;\Rightarrow\; \exists y \;\; Animal(y) \land HeadOf(h, y)$

b. $A.\ \neg Horse(x) \lor Animal(x)$
$B.\ Horse(G)$
$C.\ HeadOf(H, G)$
$D.\ \neg Animal(y) \lor \neg HeadOf(H, y)$
(Here A. comes from the first sentence in **a.** while the others come from the second. H and G are Skolem constants.)

c. Resolve D and C to yield $\neg Animal(G)$. Resolve this with A to give $\neg Horse(G)$. Resolve this with B to obtain a contradiction.

Exercise 9.QUOR

Here are two sentences in the language of first-order logic:

(A) $\forall x \;\; \exists y \;\; (x \geq y)$
(B) $\exists y \;\; \forall x \;\; (x \geq y)$

a. Assume that the variables range over all the natural numbers $0, 1, 2, \ldots, \infty$ and that the "$\geq$" predicate means "is greater than or equal to." Under this interpretation, translate (A) and (B) into English.

b. Is (A) true under this interpretation?

c. Is (B) true under this interpretation?

d. Does (A) logically entail (B)?

e. Does (B) logically entail (A)?

f. Using resolution, try to prove that (A) follows from (B). Do this even if you think that (B) does not logically entail (A); continue until the proof breaks down and you cannot proceed (if it does break down). Show the unifying substitution for each resolution step. If the proof fails, explain exactly where, how, and why it breaks down.

g. Now try to prove that (B) follows from (A).

This exercise tests the students' understanding of models and implication.

a. (A) translates to "For every natural number there is some other natural number that is smaller than or equal to it." (B) translates to "There is a particular natural number that is smaller than or equal to any natural number."

b. Yes, (A) is true under this interpretation. You can always pick the number itself for the "some other" number.

c. Yes, (B) is true under this interpretation. You can pick 0 for the "particular natural number."

d. No, (A) does not logically entail (B).

e. Yes, (B) logically entails (A).

f. We want to try to prove via resolution that (A) entails (B). To do this, we set our knowledge base to consist of (A) and the negation of (B), which we will call (-B), and try to derive a contradiction. First we have to convert (A) and (-B) to canonical form. For (-B), this involves moving the $\neg$ in past the two quantifiers. For both sentences, it involves introducing a Skolem function:

(A) $x \geq F_1(x)$
(-B) $\neg F_2(y) \geq y$

Now we can try to resolve these two together, but the occurs check rules out the unification. It looks like the substitution should be $\{x/F_2(y),\ y/F_1(x)\}$, but that is equivalent to $\{x/F_2(y),\ y/F_1(F_2(y))\}$, which fails because y is bound to an expression containing y. So the resolution fails, there are no other resolution steps to try, and therefore (B) does not follow from (A).

g. To prove that (B) entails (A), we start with a knowledge base containing (B) and the negation of (A), which we will call (-A):

(-A) $\neg F_1 \geq y$
(B) $x \geq F_2$

This time the resolution goes through, with the substitution $\{x/F_1,\ y/F_2\}$, thereby yielding $False$, and proving that (B) entails (A).

Exercise 9.RSNC
Resolution can produce nonconstructive proofs for queries with variables, so we had to introduce special mechanisms to extract definite answers. Explain why this issue does not arise with knowledge bases containing only definite clauses.

One way of seeing this is that resolution allows reasoning by cases, by which we can prove C by proving that either A or B is true, without knowing which one. If the query contains a variable, we cannot prove that any *particular* instantiation gives a fact that is entailed. With definite clauses, we always have a single chain of inference, for which we can follow the chain and instantiate variables; the solution is always a single MGU.

Exercise 9.ALLC
We said in this chapter that resolution cannot be used to generate all logical consequences of a set of sentences. Can any algorithm do this?

Not exactly. Part of the definition of algorithm is that it must terminate. Since there can be an infinite number of consequences of a set of sentences, no algorithm can generate them all. Another way to see that the answer is no is to remember that entailment for FOL is semidecidable. If there were an algorithm that generates the set of consequences of a set of sentences S, then when given the task of deciding if B is entailed by S, one could just check if B is in the generated set. But we know that this is not possible, therefore generating the set of sentences is impossible.

If we relax the definition of "algorithm" to allow for programs that *enumerate* the consequences, in the same sense that a program can enumerate the natural numbers by printing them out in order, the answer is yes. For example, we can enumerate them in order of the deepest allowable nesting of terms in the proof.