Problem A: (Skolem Constants) Consider the formula $\exists x \phi(x)$, where $\phi(x)$ is an arbitrary first-order logic expression involving the variable $x$.

i. Let $c$ be a constant that does not appear anywhere in $\phi(x)$. Prove that $\phi(c)$ is satisfiable iff $\exists x \phi(x)$ is satisfiable. This shows that the simplest case of skolemization of a formula preserves the property of satisfiability.

For the forward direction assume that $\phi(c)$ is satisfiable. This implies that there is a model $M = \langle D, I \rangle$ that satisfies $\phi(c)$. Now construct a new model $M' = \langle D, I' \rangle$ such that $I'$ is identical to $I$ except that it is not defined for $c$. Now consider the the extended model $M'[x/I(c)]$, i.e. the model that is $M'$ extended to map variable $x$ to the same object in $D$ that $c$ was mapped to. Clearly $\phi(x)$ is true in $M'[x/I(c)]$ since we know that $\phi(c)$ is true in $M$. This shows that $\exists x \phi(x)$ is satisfiable.

The reverse direction is almost identical. Assume that $\exists x \phi(x)$ is satisfiable. This means that there is a model $M = \langle D, I \rangle$ such that $M[x/o]$ for $o \in D$ satisfies $\phi(x)$. Now consider the model $M' = \langle D, I' \rangle$ that is identical to $M$, except that $I$ is extended to interpret $c$ (we know that $I$ does not already interpret $c$ since $c$ is is not in $\phi(x)$) such that $I(c) = o$. Clearly $\phi(c)$ is true in $M'$ since $\phi(x)$ is true in $M[x/o]$.

ii. Suppose that $c$ does appear in $\phi(x)$. Give a counter example showing that $\phi(c)$ and $\exists x \phi(x)$ are not necessarily equivalent with respect to satisfiability.

Let $\phi(x) = P(x) \land \neg P(c)$. Now consider the model $M = \langle D, I \rangle$ such that $D = \{o_1, o_2\}$, $I(c) = o_1$, and $I(P) = \{\{o_2\}\}$. We have that $M[x/o_2]$ satisfies $\exists x \phi(x)$. However, $\phi(c) = P(c) \land \neg P(c)$ is not satisfiable.

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

$$\theta_a = \{x/A, y/B, z/B\}$$

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

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

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

$$\theta_c = \{x/John, y/John\}$$

d. $Knows(Father(y), y), Knows(x, x)$. No unifier, because the occurs-check prevents unification of $y$ with $Father(y)$.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse(x)</td>
<td><em>x</em> is a horse</td>
<td>Mammal(x)</td>
<td><em>x</em> is a mammal</td>
</tr>
<tr>
<td>Cow(x)</td>
<td><em>x</em> is a cow</td>
<td>Parent(x,y)</td>
<td><em>x</em> is the parent of <em>y</em></td>
</tr>
<tr>
<td>Pig(x)</td>
<td><em>x</em> is a pig</td>
<td>Offspring(x,y)</td>
<td><em>x</em> is the offspring of <em>y</em></td>
</tr>
</tbody>
</table>

a. Horses, cows, and pigs are mammals.

\[
\forall x \text{ Horse}(x) \Rightarrow \text{ Mammal}(x) \\
\forall x \text{ Cow}(x) \Rightarrow \text{ Mammal}(x) \\
\forall x \text{ Pig}(x) \Rightarrow \text{ Mammal}(x)
\]

b. An offspring of a horse is a horse.

\[
\forall x, y \text{ Horse}(x) \land \text{ Offspring}(x,y) \Rightarrow \text{ Horse}(y)
\]
c. Bluebeard is a horse.

\[
\text{ Horse(Bluebeard)}
\]
d. Bluebeard is Charlie’s parent.

\[
\text{ Parent(Charlie, Bluebeard)}
\]
e. Offspring and parent are inverse relations.

\[
\forall x, y \text{ Parent}(y,x) \Rightarrow \text{ Offspring}(x,y) \\
\forall x, y \text{ Offspring}(x,y) \Rightarrow \text{ Parent}(y,x)
\]
f. Every mammal has a parent.

\[
\forall x \text{ Mammal}(x) \Rightarrow \exists y \text{ Parent}(y,x) \\
\forall x \text{ Mammal}(x) \Rightarrow \text{ Parent}(H_1(x),x)
\]

Here, \(H_1(x)\) is a skolem function.

9.10. In this question we will use the sentences you wrote in Exercise 9.9 to answer a question using a backward-chaining algorithm:

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

The proof tree is shown below. The branch with \(\text{Offspring(Bluebeard,y)}\) and \(\text{Parent(y,Bluebeard)}\) repeats indefinitely, so the rest of the proof is never reached.

\[
\begin{align*}
\text{Horse(h)} & \\
& \\
\text{Offspring(h,y)} & \text{ ------------------------ Horse(Bluebeard)} \\
& \\
& \text{Parent(y,h)} \\
& \text{yes, } \{y/\text{Bluebeard}, \text{ Offspring(Bluebeard,y)} \ldots \} \\
& \{h/\text{Charlie}\} \\
& \text{Parent(y,Bluebeard)} \\
& \text{Offspring(Bluebeard,y)} \\
& \ldots.
\end{align*}
\]
The following Prolog code defines a predicate $P$:

\[
\begin{align*}
P(X, [X|Y]). \\
P(X, [X|Z]) :- P(X,Z).
\end{align*}
\]

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

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 total that is unindented above it. The $P1$ and $P2$ annotation on a line mean that the first or second clause of $P$ was used to derive the line.

\[
\begin{align*}
P(A, [1,2,3]) \quad & \text{goal} \\
P(1, [1|2,3]) & \quad P1 \Rightarrow \text{solution, with } A = 1 \\
P(1, [1|2,3]) & \quad P2 \\
P(2, [2,3]) & \quad P1 \Rightarrow \text{solution, with } A = 2 \\
P(2, [2,3]) & \quad P2 \\
P(3, [3]) & \quad P1 \Rightarrow \text{solution, with } A = 3 \\
P(3, [3]) & \quad P2 \\
P(2, [1, A, 3]) \quad & \text{goal} \\
P(2, [1|2,3]) & \quad P1 \\
P(2, [1|2,3]) & \quad P2 \\
P(2, [2|3]) & \quad P1 \Rightarrow \text{solution, with } A = 2 \\
P(2, [2|3]) & \quad P2 \\
P(2, [3]) & \quad P1 \\
P(2, [3]) & \quad P2 \\
\end{align*}
\]

b What standard list operation does $P$ represent.

$P$ could better be called Member; it succeeds when the first argument is an element of the list that is the second argument.
9.18. 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: \( \text{HeadOf}(h,x) \) (meaning “\( h \) is the head of \( x \)”), \( \text{Horse}(x) \), and \( \text{Animal}(x) \).

\[
\forall x \ (\text{Horse}(x) \Rightarrow \text{Animal}(x))
\]
\[
\forall x, h \ [(\text{Horse}(x) \land \text{HeadOf}(h,x)) \Rightarrow \exists y \ (\text{Animal}(y) \land \text{HeadOf}(h,y))]
\]

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

A. \( \neg\text{Horse}(x) \lor \text{Animal}(x) \)
B. \( \text{Horse}(G) \)
C. \( \text{HeadOf}(H,G) \)
D. \( \neg\text{Animal}(y) \lor \neg\text{HeadOf}(H,y) \)

Here A. comes from the first sentence in a. while the others come from the second. \( H \) and \( G \) are Skolem constants.

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

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

9.21. 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?

Hint: You may want to consider somehow using a resolution theorem prover, which is refutation complete, as a black box inside another algorithm. The input to the algorithm should be a knowledge base \( KB \). When executed on \( KB \), the algorithm should output formulas, perhaps running forever. The only requirements we have are 1) the algorithm should only output formulas that are entailed by \( KB \), and 2) for any sentence \( \alpha \) such that \( KB \models \alpha \), the algorithm should output \( \alpha \) in a finite amount of time. This means of course that if \( KB \) has an infinite number of entailed formulas, the algorithm will run forever and never stop outputting formulas.

It is possible to construct an algorithm that will eventually output a formula \( \alpha \) iff \( \alpha \) is entailed by \( KB \).

The basic idea is that any resolution proof of a sentence \( \alpha \) can be encoded as a finite binary string. Furthermore, it is straightforward in concept to write an algorithm that checks whether or not a string corresponds to a legal resolution proof of a formula \( \alpha \). This suggests that we can simply enumerate strings from small to large, and outputting a formula if a string corresponds to a proof. The following pseudo-code does just that:

\[
n = 1
\]
\[
\text{repeat forever}
\]
\[
\text{for each binary string } S \text{ of length } n \text{ do},
\]
\[
\text{check if } S \text{ is a legal resolution proof}
\]
\[
\text{output } \alpha, \text{ where } \alpha \text{ is a formula proven by } S
\]

If we have that \( KB \models \alpha \) then the algorithm will eventually generate a string corresponding to its proof and then output \( \alpha \). If \( \neg(KB \models \alpha) \), then there is no such string and the algorithm will never output \( \alpha \).