1. Consider two atoms \( A_1 = P(t_1, \ldots, t_n) \) and \( A_2 = P(s_1, \ldots, s_n) \), where the \( t_i \) and \( s_i \) are terms. Let \( A = \text{LGG}(A_1, A_2) \), where \( \text{LGG} \) is the procedure described in class for computing least-general generalizations under \( \theta \)-subsumption. Prove that \( A \) \( \theta \)-subsumes both \( A_1 \) and \( A_2 \). That is, show that \( A \) is indeed a generalization of the inputs.

**Proof:** Based on the definition of \( \theta \)-subsumption we need to show that there always exists two substitutions \( \theta_1 \) and \( \theta_2 \) such that \( A \theta_1 = A_1 \) and \( A \theta_2 = A_2 \). Recall that the \( \text{LGG} \) algorithm assigns special names to the variables that it introduces into \( A \). In particular, each variable is given a name of the form \( \text{V}_{t,t'} \) where \( t \) and \( t' \) are sub-terms of \( A_1 \) and \( A_2 \) respectively. Let \( V \) be the set of all such variables introduced by the \( \text{LGG} \) process. We claim that the following substitutions have the desired property:

\[
\theta_1 = \{ \text{V}_{t,t'}/t | \text{V}_{t,t'} \in V \}
\]

\[
\theta_2 = \{ \text{V}_{t,t'}/t' | \text{V}_{t,t'} \in V \}
\]

That is, \( \theta_1 \) and \( \theta_2 \) simply substitutes any occurrence of \( \text{V}_{t,t'} \) with \( t \) and \( t' \) respectively. We now prove that \( \theta_1 \) and \( \theta_2 \) have the desired property where we will let \( A = P(r_1, \ldots, r_n) \). In particular, we prove by induction on the structure of terms that for all \( 1 \leq i \leq n \), \( r_i \theta_1 = t_i \), noting that the same argument easily extends to hold for \( \theta_2 \) with respect to the \( s_i \).

The first base case of the \( \text{LGG} \) is when \( t_i = s_i \), for which \( r_i \) is just equal to \( t_i \). In this case, the equivalence holds for any substitution. Another base case is when \( t_i \neq s_i \) and do not involve the same outermost function symbol. In this case, \( r_i = \text{V}_{t_i} \), and it is easy to see that \( r_i \theta_1 = t_i \). The inductive case occurs when \( t_i = f(g_1, \ldots, g_m) \) and \( s_i = f(h_1, \ldots, h_m) \) for some function symbol \( f \). In this case, we have that \( r_i = f(\text{LGG}(g_1, h_1), \ldots, \text{LGG}(g_m, h_m)) \). By the inductive hypothesis we have that \( \text{LGG}(g_i, h_i) \theta_1 = g_i \), which implies that \( r_i \theta_1 = t_i \). This shows that \( A \theta_1 = A_1 \). The exact same argument holds for \( \theta_2 \) and \( A_2 \), which completes the proof.

2. Recall that for two distinct constants \( c \) and \( c' \) we defined \( \text{LGG}(c, c') = \text{V}_{c,c'} \), where \( \text{V}_{c,c'} \) is a variable that is indexed by the pair \( (c, c') \). Consider a new version of \( \text{LGG} \) which we will call \( \text{LGG}' \). \( \text{LGG}' \) is identical to \( \text{LGG} \) except that for pairs of distinct constants \( c \) and \( c' \) we have \( \text{LGG}'(c, c') = V \), where \( V \) is a new variable that has not been introduced anywhere else in the \( \text{LGG}' \) execution. Show that in general \( \text{LGG}' \) does not compute a least-general generalization.
under \(\theta\)-subsumption. In particular show two clauses (possibly just two atoms) such that 
\(\text{LGG}'\) produces a clause that is strictly more general under \(\theta\)-subsumption than the clause produced by \(\text{LGG}\).

**Solution:** Consider the two ground atoms \(A_1 = P(a, a)\) and \(A_2 = P(b, b)\), where \(a\) and \(b\) are constants. We have the following:

\[
\text{LGG}(A_1, A_2) = P(V_{a,b}, V_{a,b}) = A
\]
\[
\text{LGG}'(A_1, A_2) = P(V, W) = A'
\]

where \(V_{a,b}, V, \) and \(W\) are all distinct variables. We can see that \(A'\) \(\theta\)-subsumes \(A\) since for \(\theta = \{V/V_{a,b}, W/V_{a,b}\}\) we have \(A'\theta = A\). Furthermore we see that there is no substitution \(\theta\) such that \(A\theta = A'\) since \(V_{a,b}\) would need to be mapped to different terms. Essentially \(A\) encodes the constraint that in both \(A_1\) and \(A_2\) the first and second arguments were equal, which is not encoded in \(A'\). This shows that under \(\theta\)-subsumption \(A'\) is strictly more general than \(A\), showing that the procedure \(\text{LGG}'\) does not always compute least-general generalizations.

3. Consider the two clauses:

\[
c_1 = P(x) \rightarrow P(f(x))
\]
\[
c_2 = P(x) \rightarrow P(f(f(x)))
\]

Prove that \(c_1 \models c_2\), but that \(c_1\) does not \(\theta\)-subsume \(c_2\). Note that to show that \(c_1 \models c_2\) it is sufficient to show that \(c_2\) can be derived/proved from a knowledge base that contains just \(c_1\).

**Solution:** To show that \(c_1 \models c_2\) it suffices to give a proof of \(c_2\) starting with \(c_1\), i.e. to show that \(c_1 \vdash c_2\). To do this we first note that given \(c_1\) we can use the substitution inference rule for universal quantifiers and infer that

\[
c_3 = P(f(x)) \rightarrow P(f(f(x)))
\]

is true. Now applying the resolution rule to \(c_1\) and \(c_3\) results in \(c_2\), showing that \(c_1 \models c_2\).

Alternatively, you could have done a more mechanical resolution proof by turning the formula \(c_1 \land \neg c_2\) into clausal form and using resolution to derive a contradiction.

We now argue that \(c_1\) does not \(\theta\)-subsume \(c_2\). To see this suppose that \(c_1\) does \(\theta\)-subsume \(c_2\). From this we know \(c_1\theta = c_2\) for some \(\theta\). In particular, we know that \(\theta\) must map \(x\) to itself, since \(P(x)\) is the only negative literal in \(c_1\) and \(c_2\). However, under this substitution it is not the case that \(c_1\theta \subseteq c_2\) due to the inequality of the positive literals under \(\theta\). This gives a contradiction and shows that \(c_1\) does not \(\theta\)-subsume \(c_2\).

4. Consider the following learning problem with background knowledge \(B\), positive examples \(P\), and negative examples \(N\).

\[
P = \{Q(a, b), Q(d, e), Q(g, h)\}
\]
\[
N = \{Q(j, k)\}
\]
\[
B = \{R(a, b), R(b, c), R(d, e), R(e, f), R(g, h), P(h), R(j, k), P(j)\}
\]

Simulate running Golem on this learning problem to learn a rule set that covers all positives but no negatives. For pruning LGGs you can use the pruning technique that removes literals from the body if the removal does not result in covering negative examples.
Solution: We start by computing the saturations of the three positive examples. These give us the following three ground clauses:

\[ c_1 = R(a, b) \land R(b, c) \rightarrow Q(a, b) \]
\[ c_2 = R(d, e) \land R(e, f) \rightarrow Q(d, e) \]
\[ c_3 = R(g, h) \land P(h) \rightarrow Q(g, h) \]

Golem will now consider the LGG of all pairs of these clauses \( c_{i,j} = \text{LGG}(c_i, c_j) \) giving the following:

\[ c_{1,2} = R(V_{a,d}, V_{b,e}) \land R(V_{a,e}, V_{b,f}) \land R(V_{b,d}, V_{c,e}) \land R(V_{b,e}, V_{c,f}) \rightarrow Q(V_{a,d}, V_{b,e}) \]
\[ c_{1,3} = R(V_{a,g}, V_{b,h}) \land R(V_{b,g}, V_{c,h}) \rightarrow Q(V_{a,g}, V_{b,h}) \]
\[ c_{2,3} = R(V_{d,g}, V_{e,h}) \land R(V_{e,g}, V_{f,h}) \rightarrow Q(V_{d,g}, V_{e,h}) \]

We can see that \( B \land c_{1,3} \) and \( B \land c_{2,3} \) both entail the negative example \( Q(j, k) \). This means that Golem would not consider these generalizations as seeds. Rather, we see that \( B \land c_{1,2} \) does not entail the negative example, so Golem will start with \( c_{1,2} \).

The next step of Golem is to prune the clause \( c_{1,2} \) using the negative example. Since there is only one example, this pruning is not really expected to be very reliable (since the negatives are not really representative of the full set of negatives), but we will go through the pruning anyway. The result of pruning can depend on the order that we consider removing literals. Let’s suppose that we remove body literals in the order of left to right. In this case, we can remove the first three literals and still not cover the negative example. However, if we remove the final literal \( R(V_{b,e}, V_{c,f}) \) we are left with an empty body and the negative example would be covered. So the result of pruning is the clause

\[ c' = R(V_{b,e}, V_{c,f}) \rightarrow Q(V_{a,d}, V_{b,e}) \]

Golem will now consider generalizing \( c' \) with the remaining uncovered positive clause \( c_3 \). The result of this will be overly general and cover the negative example. We can see that this is the case since we already know that the generalization of \( c_2 \) and \( c_3 \) is overly general and thus \( c' \) generalized with \( c_3 \) is overly general since \( c' \) is more general than \( c_2 \).

At this point Golem has learned its first clause \( c' \) and remove the first two positive examples corresponding to \( c_1 \) and \( c_2 \). Since \( c_3 \) is the only remaining clause and there is nothing left to generalize with, Golem can’t do anything except output \( c_3 \) or a pruned version of it as the second clause. The pruning of \( c_3 \) would just result in a clause with an empty body \( \rightarrow Q(g, h) \), which obviously does not cover the negative example. So the final theory output by Golem is:

\[ R(V_{b,e}, V_{c,f}) \rightarrow Q(V_{a,d}, V_{b,e}) \]
\[ \rightarrow Q(g, h) \]