1 Cut!

Recall the cut primitive, !.

This primitive is always true, but has the side-effect of committing all decisions that led to evaluating it, preventing the backtracking mechanism from going back to before the cut.

Consider the following definition of minimum/3.

minimum(X, Y, Z) ;‐ X =< Y, Z = X.
minimum(X, Y, Z) :‐ X > Y.

If the X =< Y check succeeds, we might still evaluate the second clause if Z = X fails. This is unnecessary work because X > Y is the exact opposite of X =< Y. We can solve this inefficiency by throwing in a cut to get the following:

minimum(X, Y, Z) ;‐ X =< Y, !, Z = X.
minimum(X, Y, Z) :‐ X > Y.
Now, when we try \( X =< Y \) and succeed, Prolog will no longer attempt to try the other definition even if \( Z = X \) fails.

However, cut is defined in terms of how it affects the evaluation of the program, and not in terms of the logical foundation Prolog originates from. Unsurprisingly, cuts can change the logical meaning of a Prolog program in unexpected ways. For example, we are able to define negation as the following:

\[
\neg(X) \leftarrow X, !, \text{fail}.
\]
\[
\neg(X).
\]

After seeing the second definition, one might reasonably assume that \( \neg(X) \) should be true for all \( X \). One might think that regardless of whether the first definition succeeds or not, the second definition should always accept. However, by inserting the cut, the first definition prevents the backtracking mechanism from trying the second definition if the evaluation mechanism fails (which it will, because we have the fail right after it). Unlike the previous green cut, where the cut had no effect on the logical meaning of the program, the use of cut here is fundamental to the behavior of \( \neg/1 \) and is therefore called a red cut.

**Task 1** (2 points). Prove or disprove that the following cut is a green cut.

\[
\text{foo}(A) \leftarrow \text{fail}, !, \text{fail}.
\]
\[
\text{foo}(A).
\]

**Solution 1:** This is a green cut. The side effects of a cut only happen when execution arrives to the cut. The fail right before it prevents this from happening.

**Task 2** (2 points). Prove or disprove that the following cut is a green cut.

\[
\text{member}(X, [X|Xs]) \leftarrow \text{member}(X, Xs), !.
\]
\[
\text{member}(X, [\_|Xs]) \leftarrow \text{member}(X, Xs), !.
\]

**Solution 2:** This was a bit sneaky. In modes of execution where the arguments are grounded, this behaves like a green cut, but this is actually a red cut in the general case. Consider running the following prolog code:

\[
\text{member}(X, [0, 1, 2]), X = 2.
\]

This will be false. Without the cut, it will be true. Perhaps, this might help suggest why modes are useful things to specify.

**Task 3** (2 points). Prove or disprove that the following cut is a green cut.
fancy(1).
fancy(2).
fancy_member(X, [X|Xs]) :- fancy(X), !.
fancy_member(X, [_|Xs]) :- fancy_member(X, Xs).

Solution 3: By the same style of argument, this is also a red cut in general. We can only find one X such that fancy(X) is true in any given list. Try the following example:

fancy_member(X, [0, 1, 0, 2]), X = 2.

2 Smooth Operator Operating Correctly

Task 4 (4 points). Using the rules for explicit backtracking from class, derive a proof of \( d \neq \top \neq \bot \)

\[
a. \\
b :: a \\
c :: a, fail; a. \\
d :: c, b, a.
\]

Solution 4:

\[
\begin{array}{c}
\dfrac{T / \top / \bot}{a / \top / \bot} \\
\dfrac{T / a \wedge \top / \bot}{a / a \wedge \top / \bot} \\
\dfrac{b / a \wedge \top / \bot}{T / b \wedge a \wedge \top / \bot} \\
\dfrac{T / b \wedge a \wedge \top / \bot}{a / b \wedge a \wedge \top / \bot} \\
\dfrac{\bot / b \wedge a \wedge \top / (a \wedge b \wedge a \wedge \top) \vee \bot}{T / b \wedge a \wedge \top / (a \wedge b \wedge a \wedge \top) \vee \bot} \\
\dfrac{(a \wedge \bot)/ b \wedge a \wedge \top / (a \wedge b \wedge a \wedge \top) \vee \bot}{(a \wedge \bot) \vee a / b \wedge a \wedge \top / \bot} \\
\dfrac{c / b \wedge a \wedge \top / \bot}{c \wedge b \wedge a / \top / \bot} \\
\dfrac{c \wedge b \wedge a / \top / \bot}{d / \top / \bot}
\end{array}
\]

3 Bringing them to terms with each other

Using the rules provided in class, find the most general unifier of the following terms or explain why it is not possible to find one.
Task 5 (2 points).  \( f(b, c, a) \) and \( f(c, a, b) \)

Solution 5: Not unifiable. \( a, b, \) and \( c \) are nullary functions and cannot be substituted.

Task 6 (2 points).  \( f(x, x, g(x)) \) and \( f(y, h(z, a), w) \)

Solution 6: \( (y/x)(h(z, a)/y)(g((z, a))/w) \)

Task 7 (2 points).  \( f(x, x, g(x)) \) and \( f(y, h(z, a), z) \)

Solution 7: Not unifiable. Notice that \( z \) needs to be substituted with \( h(z, a) \), but \( z \) occurs inside of \( h(z, a) \).

Task 8 (2 points).  \( f(y, f(x, y)) \) and \( f(a, f(y, x)) \)

Solution 8: \( (a/y)(a/x) \)

Task 9 (2 points).  \( f(x, f(h(y), y, z), h(y)) \) and \( f(x, f(z, y, z), h(x)) \)

Solution 9: \( (h(y)/z)(x/y) \)