Practice Prolog

Assume we have a database with the following facts and rules:

parent(mona, homer).

\text{test1}(X, Y) :- \ X =\ Y.
\text{test2}(X, Y) :- \ X =\ Y.

For which of the following queries will Prolog return \textit{false}?

(a) parent(Who, homer).
(b) parent(Who, marge).
(c) test1(Y, 2).
(d) test2(Y, 2).
(e) parent(maggie, marge) = parent(Parent, Child).

\textbf{Firstname Lastname}

\textit{(Your response)}
Practice Prolog

Assume we have a database with the following facts and rules:

\begin{verbatim}
parent(mona, homer).
test1(X, Y) :- X == Y.
test2(X, Y) :- X = Y.
\end{verbatim}

For which of the following queries will Prolog return \textit{false}?

(a) parent(Who, homer).
    \hspace{1cm} Who \rightarrow \text{mona} \\
(b) parent(Who, marge).
(c) test1(Y, 2).
(d) test2(Y, 2).
    \hspace{1cm} Y \rightarrow 2 \\
(e) parent(maggie, marge) = parent(Parent, Child).
    \hspace{1cm} Parent \rightarrow \text{maggie}; Child \rightarrow \text{marge}
## Equality

Do these things give us the same value (possibly after using the current set of bindings, \( \Theta \))?  

<table>
<thead>
<tr>
<th>( \text{term}_1 == \text{term}_2 ), given ( \Theta )</th>
<th>( \text{atom}_2 )</th>
<th>( \text{var}_2 )</th>
<th>( \text{pred}<em>2(t</em>{2-1}, \ldots, t_{2-n}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{atom}_1 )</td>
<td>Are they literally the same?</td>
<td>( \Theta[\text{var}_2] == \text{atom}_1 )</td>
<td>false</td>
</tr>
<tr>
<td>( \text{term}_1 ) ( \text{var}_1 )</td>
<td>( \Theta[\text{var}_1] == \text{atom}_2 )</td>
<td>( \Theta[\text{var}_1] == \Theta[\text{var}_2] )</td>
<td>( \Theta[\text{var}<em>1] == \text{pred}<em>2(t</em>{2-1}, \ldots, t</em>{2-n}) )</td>
</tr>
<tr>
<td>( \text{pred}<em>1(t</em>{1-1}, \ldots, t_{1-n}) )</td>
<td>false</td>
<td>( \Theta[\text{var}<em>2] == \text{pred}<em>1(t</em>{1-1}, \ldots, t</em>{1-n}) )</td>
<td>( \text{pred}<em>1 == \text{pred}<em>2, n == m, t</em>{1-i} == t</em>{2-i} ).</td>
</tr>
</tbody>
</table>
## Unification

Can we extend an empty set of bindings, $\Theta$, to make the statement true?

<table>
<thead>
<tr>
<th></th>
<th>$term_1 = term_2$</th>
<th>$term_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$atom_1$</td>
<td>$atom_1 = atom_2$</td>
<td>$true$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$false$</td>
</tr>
<tr>
<td>$var_1$</td>
<td>$true$</td>
<td>$true$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$true$</td>
</tr>
<tr>
<td>$pred_1(t_{1-1}, \ldots, t_{1-n})$</td>
<td>$false$</td>
<td>$true$</td>
</tr>
<tr>
<td></td>
<td>$pred_1 = pred_2,$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$n = m,$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$t_{1-i} = t_{2-i}.$</td>
<td></td>
</tr>
</tbody>
</table>
## Unification

Can we extend an empty set of bindings, $\emptyset$, to make the statement true?

<table>
<thead>
<tr>
<th>$term_1 = term_2$</th>
<th>$term_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$atom_1$</td>
<td>$atom_2$ = $atom_2$</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$term_1$ var$_1$</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td>{var$_1$ $\mapsto$ atom$_2$}</td>
</tr>
<tr>
<td>pred$<em>1$(t$</em>{1-1}$,...,t$_{1-n}$)</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Practice Prolog

Assume we have a database with the following facts and rules:

\[
\begin{align*}
  \text{parent}(\text{mona}, \text{homer}). \\
  \text{test1}(X, Y) :& - X == Y. \\
  \text{test2}(X, Y) :& - X = Y.
\end{align*}
\]

For which of the following queries will Prolog return false?

(a) parent(Who, homer).
(b) parent(Who, marge).
(c) test1(Y, 2).
(d) test2(Y, 2).
(e) parent(maggie, marge) = parent(Parent, Child).

(Your response)
Practice Prolog

Assume we have a database with the following facts and rules:

parent(mona, homer).

\text{test1}(X, Y) :\neg X == Y.

\text{test2}(X, Y) :\neg X = Y.

For which of the following queries will Prolog return \textit{false}?

(a) parent(Who, homr). \hspace{1cm} \text{Who} \rightarrow \text{mona}
(b) parent(Who, marge).
(c) test1(Y, 2).
(d) test2(Y, 2). \hspace{1cm} Y \rightarrow 2
(e) parent(maggie, marge) = parent(Parent, Child). \hspace{1cm} \text{Parent} \rightarrow \text{maggie}; \text{Child} \rightarrow \text{marge}
How Prolog works

depth-first search, using unification

Given a goal...
e.g., siblings(bart, Who)

...for each rule with the goal’s name...
e.g., siblings(X, Y)

...unify the goal with rule’s head.
e.g., siblings(bart, Who) = siblings(X, Y).

If the rule has a body with subgoals...
e.g., parent(P, X), parent(P, Y), X \( \Leftarrow\) Y.

...for each subgoal...
e.g., parent(P, X) with \( \{X \leftarrow \text{bart}\}\)

...solve subgoal to extend the current set of bindings.
e.g., \( \{P \leftarrow \text{homer}, X \leftarrow \text{bart}\}\)

If we ever get false, backtrack and try the next option.
e.g., \( \{P \leftarrow \text{homer}, X \leftarrow \text{bart}, Y \leftarrow \text{bart}\}\) but \( X \Leftarrow Y \). Therefore, backtrack and eventually try \( Y \leftarrow \text{lisa}\)
siblings(X, Y) :- parent(P, X), parent(P, Y), X \( \leq \) Y.

?- siblings(bart, lisa).
true ;
true.

?- siblings(bart, bart).
false.

?- siblings(bart, Who).
Who = lisa ;
Who = maggie ;
Who = lisa ;
Who = maggie.
How Prolog works

depth-first search through possible bindings

Given a goal...
  e.g., siblings(bart, Who)

...for each rule with the goal’s name...
  e.g., siblings(X, Y)

...unify the goal with rule’s head.
  e.g., siblings(bart, Who) = siblings(X, Y).

If the rule has a body with subgoals...
  e.g., parent(P, X), parent(P, Y), X \models Y.

...for each subgoal...
  e.g., parent(P, X) with \{X\rightarrow bart\}

...solve subgoal to extend the current set of bindings.
  e.g., \{P\rightarrow homer, X\rightarrow bart\}

If we ever get \textit{false}, backtrack and try the next option.
  e.g., \{P\rightarrow homer, X\rightarrow bart, Y\rightarrow bart\} but X \models Y. Therefore, backtrack and eventually try \{Y \rightarrow lisa\}

Prolog gives us \underline{all true statements}.
siblings(X, Y) :- parent(P, X), parent(P, Y), X <= Y.

?- siblings(bart, lisa).
true ;
true.

?- siblings(bart, bart).
false.

?- siblings(bart, Who).
Who = lisa ;
Who = maggie ;
Who = lisa ;
Who = maggie.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \=\leq Y.

?- siblings(bart, lisa), !.
true ;

?- siblings(bart, bart).
false.

?- siblings(bart, Who), !.
Who = lisa.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \<\=\ Y.

?- siblings(bart, lisa), !. true ;

?- siblings(bart, bart). false.

?- siblings(bart, Who), !. Who = maggie.
siblings(X, Y) :- parent(P, X), parent(P, Y), X \leq Y.

?- setof(Who, siblings(bart, Who), Answer).
Answer = [lisa, maggie].

?- setof([X, Y], siblings(X, Y), Answer).
Answer = [[bart, lisa], [bart, maggie], [lisa, bart], [lisa, maggie], [maggie, bart], [maggie, lisa]].
Lists in Prolog

Construction

[]  empty

[F|R]  cons!

[E₁, E₂, E₃]  literal
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length}(L, N) :- L = [], N = 0.
\]
Lists in Prolog

Selection, i.e., pattern matching

\[ \text{length}(L, N) :- L = [], N = 0. \]

is the same as

\[ \text{length}([], 0). \]

pattern matching on the left-hand side!
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length}([], 0).
\]

\[
\text{length}([F|R], N) :\text{length}(R, M), N \text{ is } M+1.
\]

singleton variable :(

math, in Prolog

pattern matching on the left-hand side!
Lists in Prolog

Selection, i.e., pattern matching

\[
\text{length}([], 0).
\]

"don't care"

\[
\text{length}([_|R], N) :- \text{length}(R, M), N \text{ is } M+1.
\]

math, in Prolog

pattern matching on the left-hand side!
You-try! member

```prolog
member( , ).
member( , ) :- ...

?- member(3, [1,2,3,4,5]).
true

?- member(6, [1,2,3,4,5]).
false.

?- member(E, [1,2,3,4,5]).
E = 1 ;
E = 2 ;
E = 3 ;
E = 4 ;
E = 5.
```
You-try! member

member is built in to SWI-Prolog

```
member(E, [E|_]).

member(E, [_|R]) :- member(E, R).
```

?- member(3, [1,2,3,4,5]).
true;
false.

?- member(6, [1,2,3,4,5]).
false.

?- member(E, [1,2,3,4,5]).
E = 1 ;
E = 2 ;
E = 3 ;
E = 4 ;
E = 5.