Backtracking

- Forward chaining goes from axioms forward into goals.
- Backward chaining starts from goals and works backwards to prove them with existing axioms.

Backtracking example

```prolog
rainy(seattle).
rainy(rochester). cold(rochester).
snowy(X) :- rainy(X), cold(X).
```

Cut (!)

- As a goal it succeeds, but with a side effect:
  - Commits interpreter to choices made since unifying parent goal with left-hand side of current rule.

Cut (!) Example

```prolog
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).
```
Cut (!) Example 2

rainy(seattle).
rainy(rochester).
cold(rochester).

snowy(X) :- rainy(X), !, cold(X).
snowy(troy).

Cut (!) Example 3

rainy(seattle) :- !.
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
snowy(troy).

Cut (!) Example 4

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- !, rainy(X), cold(X).

C. Varela  7
C. Varela  8
C. Varela  9
C. Varela 10
C. Varela 11
C. Varela 12

C. Varela  2
Cut (!) Example 5

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X), !.

First-Class Terms

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call(P)</td>
<td>Invoke predicate as a goal.</td>
</tr>
<tr>
<td>assert(P)</td>
<td>Adds predicate to database.</td>
</tr>
<tr>
<td>retract(P)</td>
<td>Removes predicate from database.</td>
</tr>
<tr>
<td>functor(T,F,A)</td>
<td>Succeeds if T is a term with functor F and arity A.</td>
</tr>
</tbody>
</table>

not P is not ¬P

- In Prolog, the database of facts and rules includes a list of things assumed to be true.
- It does not include anything assumed to be false.
- Unless our database contains everything that is true (the closed-world assumption), the goal not P can succeed simply because our current knowledge is insufficient to prove P.

More not vs ¬

- snowy(X), X = seattle.
  
Prolog does not reply: X = seattle.

The meaning of not(snowy(X)) is:

¬∃X [snowy(X)]

rather than:

∃X [¬snowy(X)]

Fail, true, repeat

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail</td>
<td>Fails current goal.</td>
</tr>
<tr>
<td>true</td>
<td>Always succeeds.</td>
</tr>
<tr>
<td>repeat</td>
<td>Always succeeds, provides infinite choice points.</td>
</tr>
</tbody>
</table>

repeat. repeat := repeat.
**not Semantics**

```prolog
not(P) :- call(P), !, fail.
not(_).  
```

Definition of `not` in terms of failure (`fail`) means that variable bindings are lost whenever `not` succeeds, e.g.:

```
?- not(not(snowy(X))).  
X = _G147  
```

---

**Conditionals and Loops**

```prolog
statement := condition, !, then.
statement := else.
```

```
natural(1).
natural(N) :- natural(M), N is M+1.
my_loop(N) :- natural(I), I<=$N,
             write(I), nl,
             I=N,  
             !, fail.
```

Also called `generate-and-test`.

---

**Exercises**

74. What do the following Prolog queries do?

```
?- repeat.  
?- repeat, true.  
?- repeat, fail.  
```

Corroborate your thinking with a Prolog interpreter.

75. Draw the search tree for the query "not(not(snowy(City)))". When are variables bound/unbound in the search/backtracking process?

76. PLP Exercise 11.24 (pg 655).

77. *PLP Exercise 11.34 (pg 656).