

Cut (!) Example 2

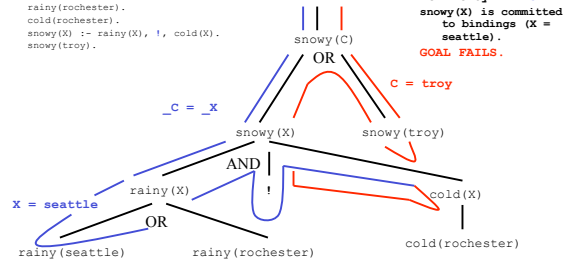
```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).
snowy(troy).
```

C. Varela

7

Cut (!) Example 2

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



C. Varela

8

Cut (!) Example 3

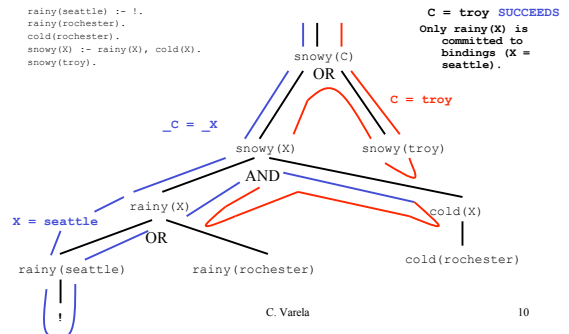
```
rainy(seattle) :- !.
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
snowy(troy).
```

C. Varela

9

Cut (!) Example 3

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



C. Varela

10

Cut (!) Example 4

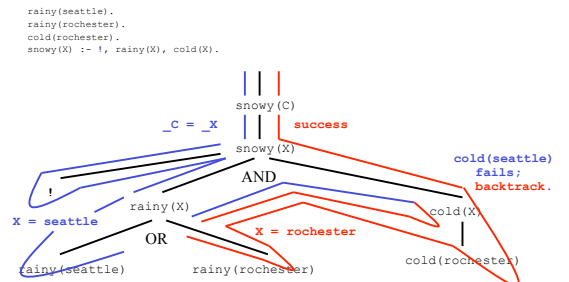
```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- !, rainy(X), cold(X).
```

C. Varela

11

Cut (!) Example 4

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



C. Varela

12

Cut (!) Example 5

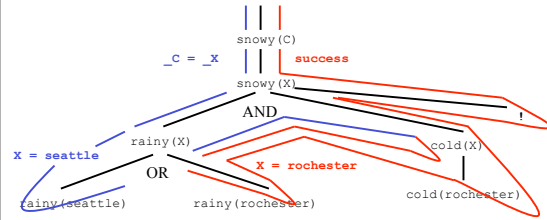
```
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X), !.
```

C. Varela

13

Cut (!) Example 5

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



C. Varela

14

First-Class Terms

call(P)	Invoke predicate as a goal.
assert(P)	Adds predicate to database.
retract(P)	Removes predicate from database.
functor(T, F, A)	Succeeds if T is a <i>term</i> with <i>functor</i> F and <i>arity</i> A.

C. Varela

15

not P is not $\neg P$

- 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.

C. Varela

16

More not vs \neg

```
?- snowy(X).
X = rochester
?- not (snowy(X)).
no
```

Prolog does not reply: **X = seattle.**

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

$\neg \exists X$ [snowy(X)]

rather than:

$\exists X$ [\neg snowy(X)]

C. Varela

17

Fail, true, repeat

fail	Fails current goal.
true	Always succeeds.
repeat	Always succeeds, provides infinite choice points.

```
repeat.
repeat :- repeat.
```

C. Varela

18

not Semantics

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

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
```

C. Varela

19

Conditionals and Loops

```
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*.

C. Varela

20

Prolog lists

- `[a,b,c]` is syntactic sugar for:

```
.(a,.(b,.(c,[])))
```

where `[]` is the empty list, and `.` is a built-in cons-like functor.

- `[a,b,c]` can also be expressed as:

```
[a | [b,c]] ,or  
[a, b | [c]] ,or  
[a,b,c | []]
```

C. Varela

21

Prolog lists append example

```
append([],L,L).  
append([_:_],A,[_:L]) :- append(T,A,L).
```

C. Varela

22

Exercises

8. What do the following Prolog queries do?

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

Corroborate your thinking with a Prolog interpreter.

9. Draw the search tree for the query `not(not(snowy(City)))`. When are variables bound/unbound in the search/backtracking process?
10. PLP Exercise 11.24 (pg 655).
11. *PLP Exercise 11.34 (pg 656).

C. Varela

23