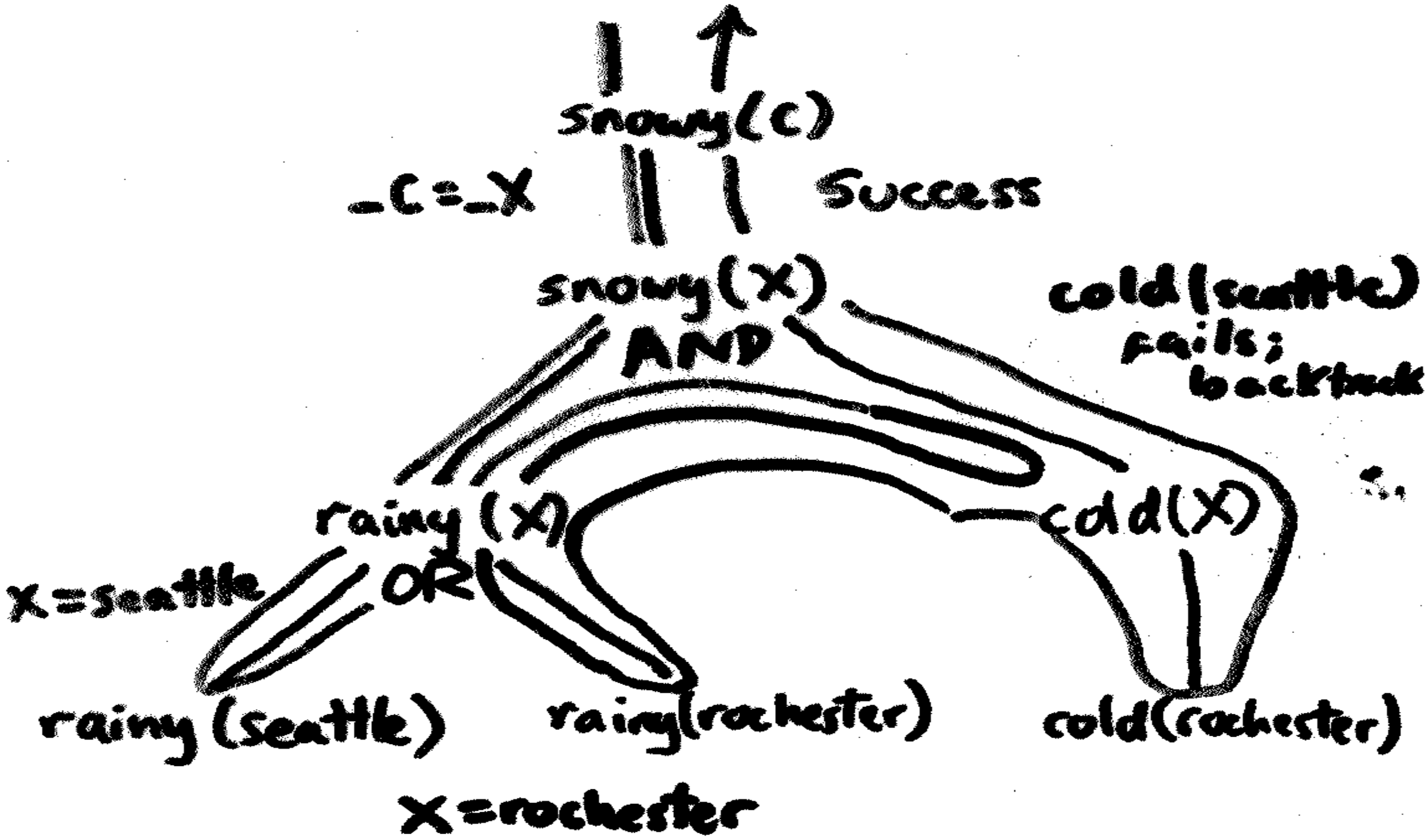
BACKTRACKING

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

BACKTRACKING

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



IMPERATIVE CONTROL FLOW

Programmer has explicit control on backtracking process.

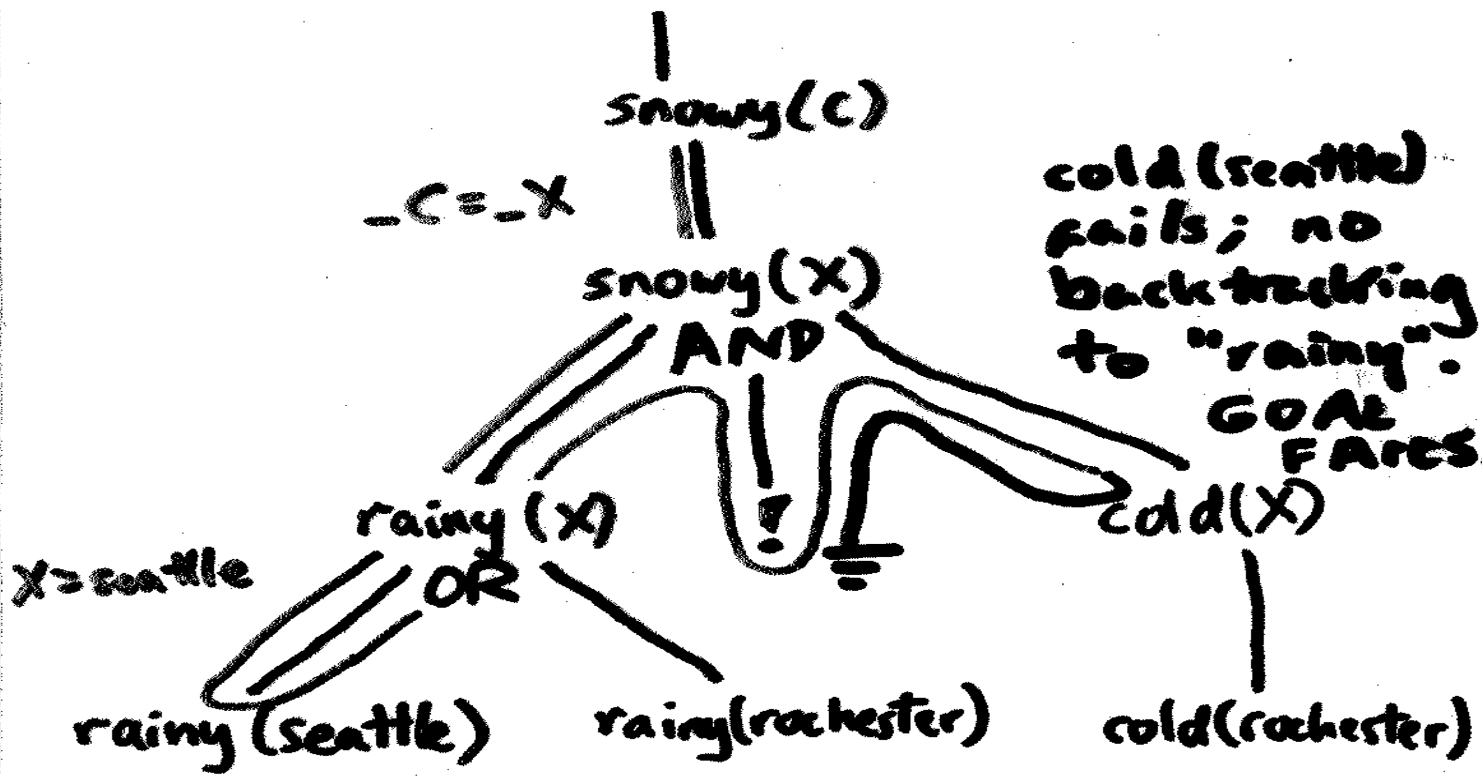
CUT (!)

- As a goal it succeeds, but with side-effect:

  Commits interpreter to choices made since unifying parent goal with left-hand side of current rule.

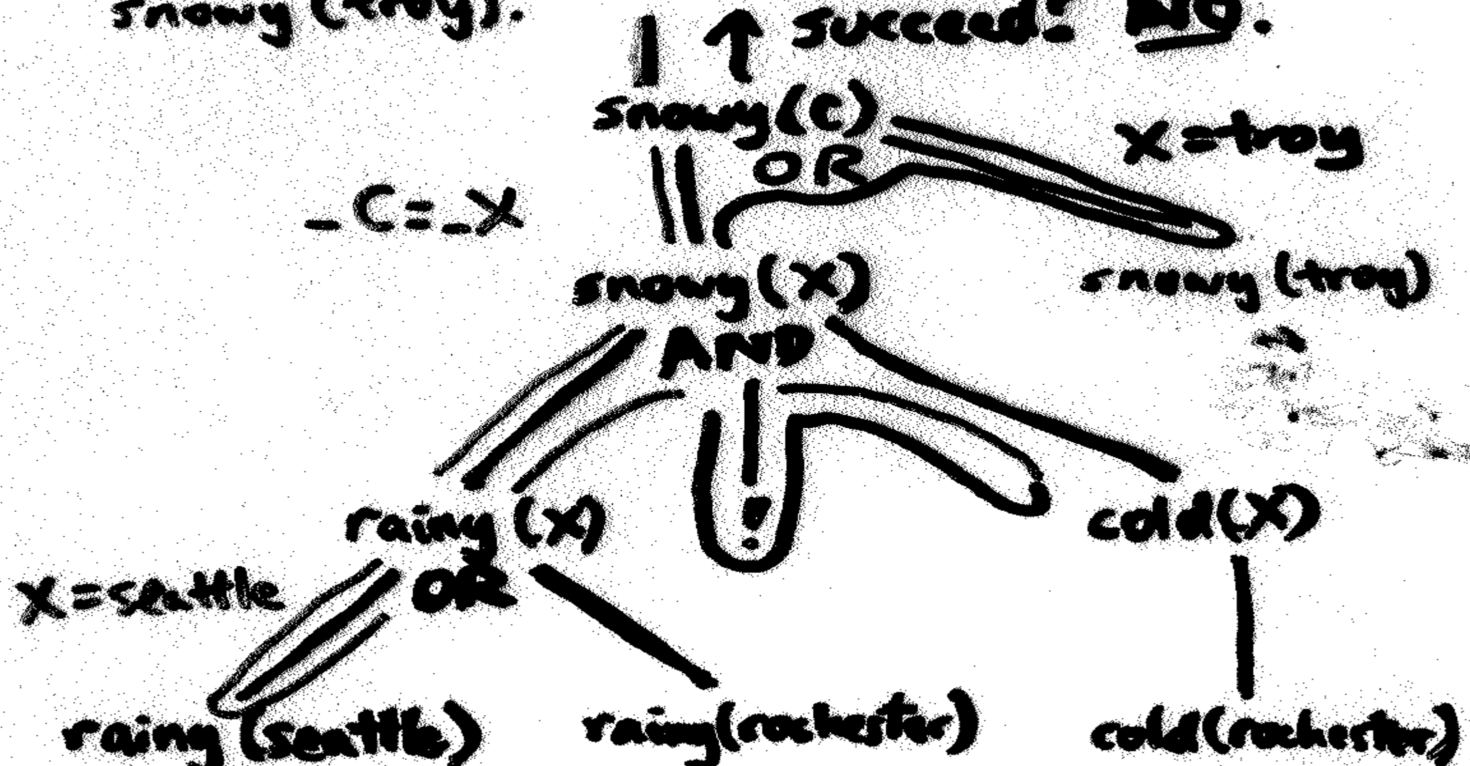
BACKTRACKING

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



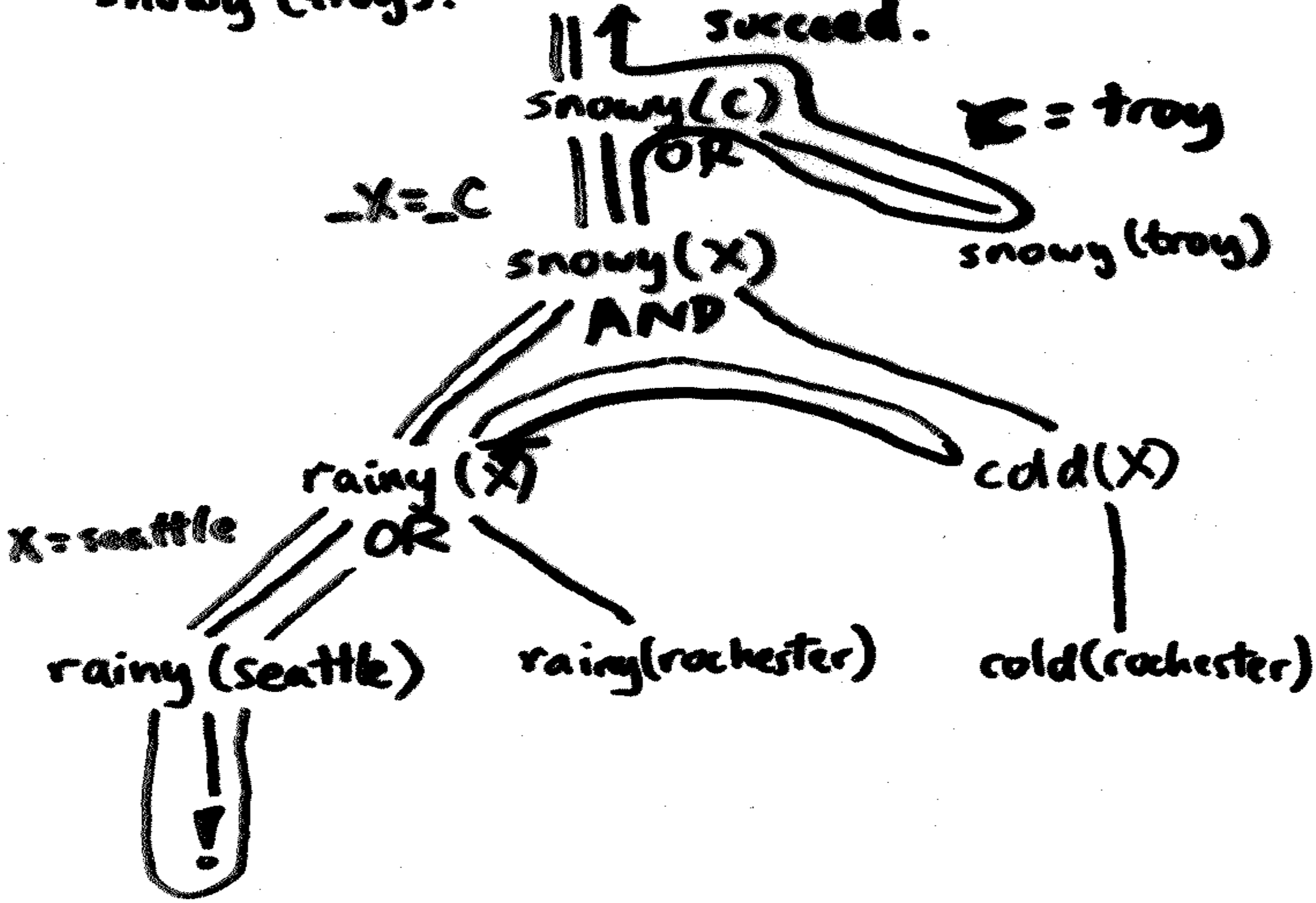
BACKTRACKING

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



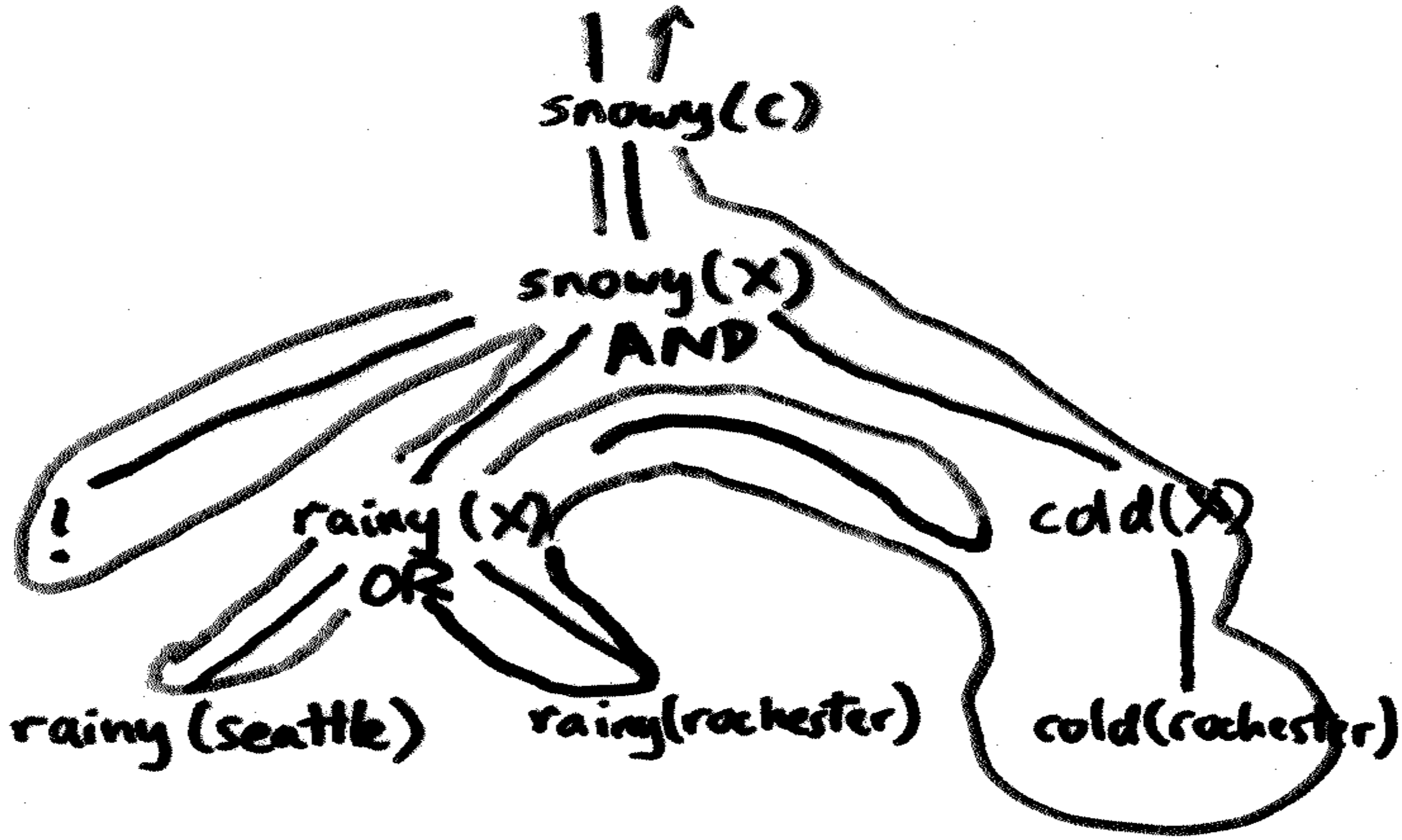
BACKTRACKING

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



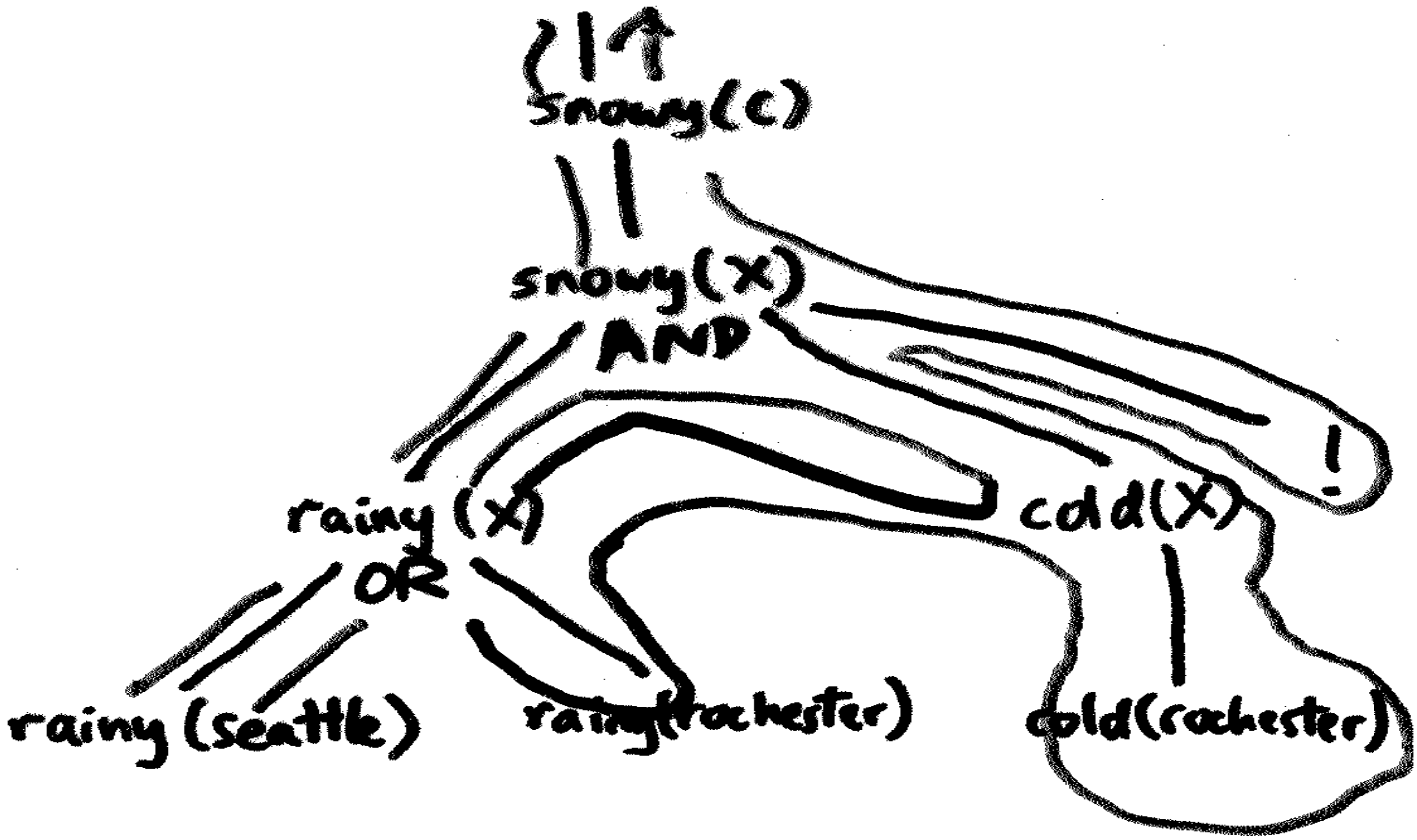
BACKTRACKING

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



BACKTRACKING

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



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 term w/ functor F and arity A.

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.

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
```

MORE NOT VS $\neg$

?- snowy(X).

X = rochester

?- not(snowy(X)).

no

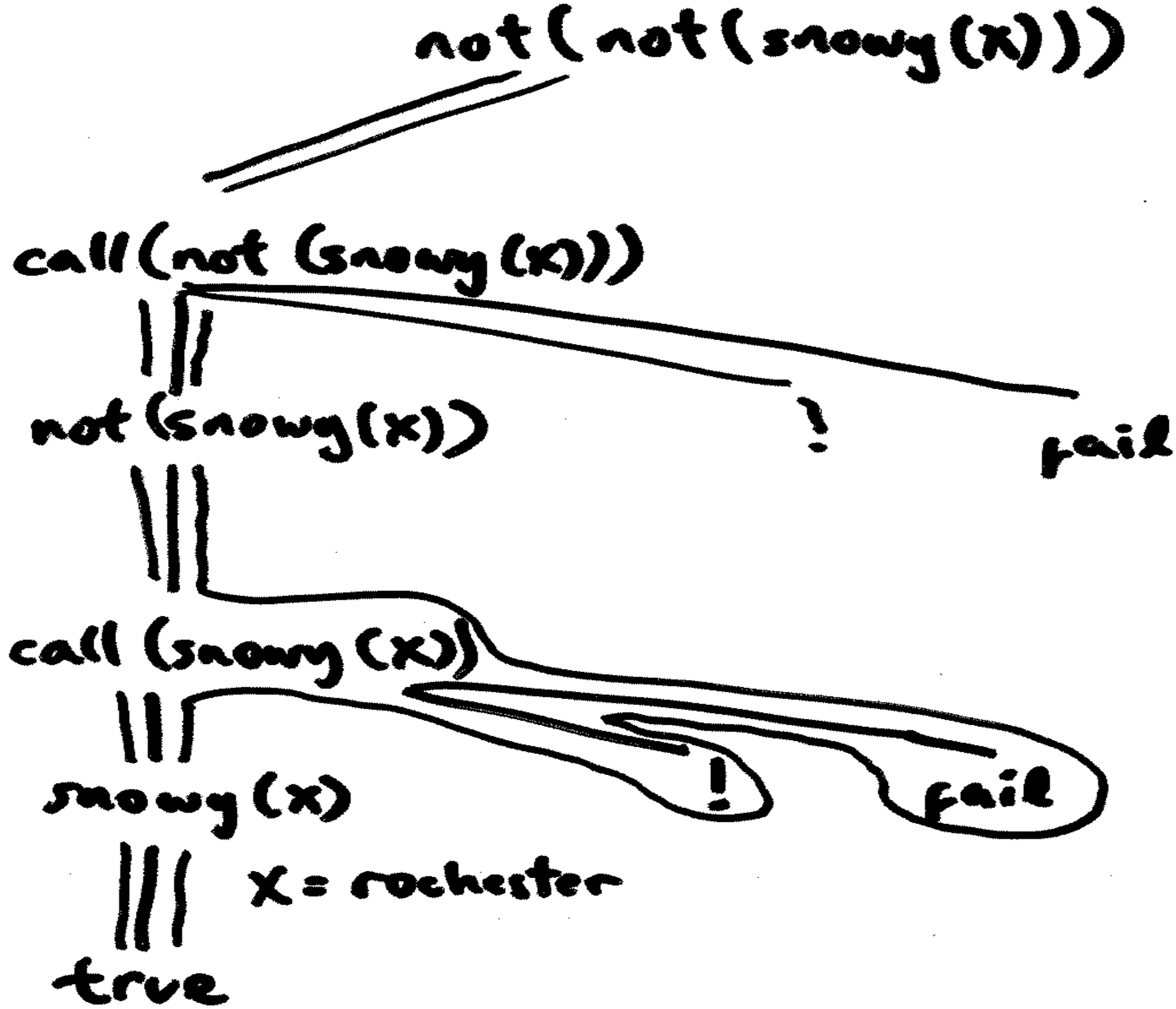
// it does NOT reply:
X = seattle

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

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

rather than:

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


not(not(snowy(X)))
call(not(snowy(X)))
not(snowy(X))
!
fail
call(snowy(X))
snowy(X)
!
fail
X = rochester
true

fail, true, ...

fail	fail current goal.
true	always succeed.
repeat	always succeed, provides infinite choice points

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

Exercise:

what do the following queries do?

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

NOT

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

IF-THEN (->)

```
->(If, Then, Else)
    :- If, !, Then.
->(If, Then, Else) :- Else.

->(If, Then, Else)
    :- If, Then, !
```

CONDITIONALS AND LOOPS

```
statement :- condition, !, then_part.
statement :- else_part.
```

```
natural(1).
natural(N) :- natural(M),
              N is M+1.

my_loop(N) :- natural(I), I<=N,
              write(I), nl,
              I = N,          // test-
              !, fail.        // cut-fail.
```

Also called generate-and-test.