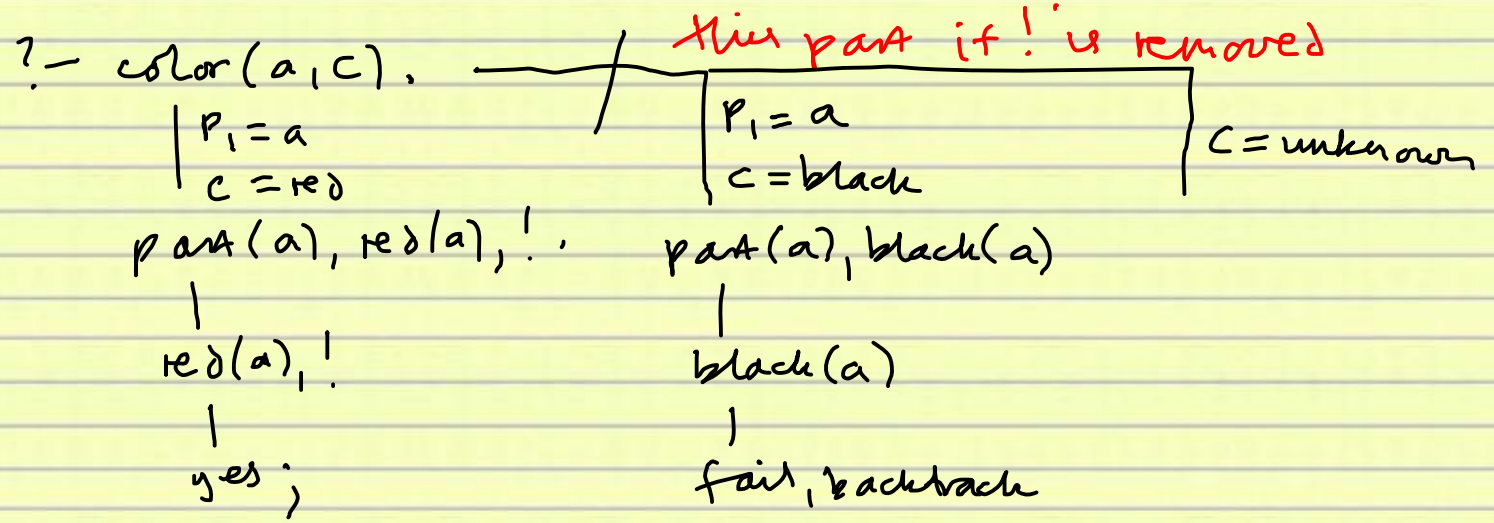
```
part(a).
part(b).
part(c).
red(a).
black(b).
color(P, red) :- part(P), red(P), !.
color(P, black) :- part(P), black(P), !.
color(_, unknown).
```



The cut disallows the possibility of a part having two colors.

Negation

- big topic in logic, but very limited in Prolog
- Prolog can only have a true head term for a rule (Horn clause logic)
- can have negation on RHS of a rule

e.g.
```
bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
married(tom).
```

'negation as failure'
- if the enclosed term succeeds, not fails
- if the term fails, not succeeds

```
?- bachelor(henry)
   |P = henry
male(henry), not(married(henry))
   |
not(married(henry))
   |
  yes
```

```
? - bachelor(Who).
    |P1 = Who
male(Who), not(married(Who))
    |Who = henry
not(married(henry))
    |
yes, Who = henry
```

? - bachelor(Who). with male facts swapped

```
    | P1 = Who
male(Who), not(married(Who))
    |Who = tom
not(married(tom))
    |
fail
```

```
?- not(married(Who)).
   |Who = tom
   no
```

This is not true negation from logic. Some results are counterintuitive

We can write not in Prolog:

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

If call Q success then the ! ensures immediate failure

If call Q fails the fact will succeed, therefore not succeeds.

```
?- not(married(henry))
   | Q = married(henry)
   call married(henry), !, fail
   |
   fail
```

yes (matches the fact)

```
?- not(married(tom))
    | Q1 = married(tom)
   call married(tom), !, fail
    |
    !, fail
    |
   fail
    |
   no
```

```
?- not(married(Who))
    | Q1 = married(Who)
   call married(Who), !, fail
    | Who2 = tom
    !, fail
    |
   fail
    |
   no
```

tom prevents us from finding the negative fact about Henry.