**Negation and the Cut**

- **Negation as Failure**
  - Negation succeeds if search fails.
  - Not Constructive - Unification does not produce any bindings.
  - Consistent Interpretation depends on Closed World Assumption
- **The Cut '!'**
  - A device for controlling the search
  - Used to increase efficiency
  - BUT can alter semantics of a program -- change its solution set.

---

**Negation as Failure**

```
single_student(X) :-
  (\+ married(X)),
  student(X).
student(bill).
student(joe).
married(joe).
```

```
:- single_student(bill).
  \+ yes.
:- single_student(joe).
  \+ no.
```

?-) single_student(X)
  \+ no.
**Negation as Failure**

```
single_student(X) :- 
\(+ married(X)), 
student(X).
student(bill). 
student(joe). 
marged(joe).

:- single_student(bill). 
  \( yes. 
:- single_student(joe). 
  \( no.
```

\(?- single_student(X) \rightarrow no.\)

**Negation as Failure 2nd Try**

```
single_student(X) :- 
\(+ married(X)). 
student(X), student(X), 
marged(joe).
student(bill). 
student(joe). 
marged(joe).

:- single_student(bill). 
  \( yes. 
:- single_student(joe). 
  \( no.
```

\(?- single_student(X) \rightarrow X=bill.\)

**Closed World Assumption**

- Assumption that the world is defined in its entirety
  - The representation is "complete"/"closed"
- No true statement is missing from the representation
- In practice, assumed for conventional databases
  - "Sorry, sir you must NOT exist your social security number is NOT IN our database, bye, bye".
- From a logic program, P, allows us to conclude
  - the negation of A
  - If A is NOT IN the meaning of P

**Negation as Failure & the CWA**

```
single_student(X) :- 
  student(X), 
\(+ married(X)). 
student(bill). 
student(joe). 
marged(joe). 
student(jim)

:- single_student(bill). 
  \( yes. 
:- single_student(joe). 
  \( no. 
:- single_student(jim). 
  \( yes.
```

\(?- single_student(X) \rightarrow yes. \)

\(?- single_student(X) \rightarrow X=bill.\)

\(?- single_student(X) \rightarrow no.\)

But Jim IS married.
Maybe I should read up on the CWA.
The Cut (!)

- The one and only "!
- There are GOOD, BAD and Ugly ones (usages).
- GREEN and RED ones (usages).
- Goes before a cut produce first set and only the first set of bindings for named variables.
- Commits a choice.
- No alternative matches considered upon backtracking.
- Green Cuts
  - Exclude clauses (solution attempts), but NOT solutions.
  - Removal of Cut does NOT change the meaning of the program. The cut's positioning just effects efficiency.
- Red Cuts
  - Alter the actual meaning of the program.
- Bad Cut
  - A cut used in such a way as to make the actual meaning diverge from the intended meaning.
- Ugly Cut
  - Obscures intended meaning but does not loose it.

A Green Cut

\[
\text{fact}(N, 1) :- N = 0, !. \\
\text{fact}(N, F) :- N > 0, M \text{ is } N \text{ --1}, \text{fact}(M, F_1) \rightarrow F \text{ is } N \times F_1.
\]

Goals before a cut produce first set and only the first set of bindings for named variables.

A Good Red Cut

\[
\text{if}_\text{then}_\text{else}(\text{If, Then, Else}) :- \text{If, Then}, !, \text{Then.} \\
\text{if}_\text{then}_\text{else}(\text{If, Then, Else}) :- \text{else.}
\]

If we take out the cut we change the meaning -- so the cut is RED. But it is used to produce the meaning we want -- so the cut is GOOD.

A BAD Red Cut

\[
\text{R1. pension}(X, \text{disabled}) :- \text{disabled}(X), !. \\
\text{R2. pension}(X, \text{senior}) :- \text{over65}(X), \text{paid_up}(X), !. \\
\text{R3. pension}(X, \text{supplemental}) :- \text{over65}(X), !. \\
\text{R4. pension}(X, \text{nothing}) :- \text{"The Default" If everything else fails.}.
\]

\[
\text{F1. disabled(joe), !.} \\
\text{F2. over65(joe), !.} \\
\text{F3. paid_up(joe), !.} \\
\text{F4. over65(lou), !.} \\
\text{F5. paid_up(lou), !.}
\]

\[
\text{Q1. ?- pension(joe, nothing) } \rightarrow \text{yes.} \\
\text{Q2. ?- pension(joe, disabled) } \rightarrow \text{yes,} \\
\text{Q2. ?- pension(joe, senior) } \rightarrow \text{over65(joe), paid_up(joe), !.} \\
\text{Q2. ?- pension(X, senior) } \rightarrow X = \text{joe.}
\]

What happened to Lou's pension? Isn't he a senior?
Joe's Revenge

R1. pension(X, disabled_pension) :- disabled(X).
R2. pension(X, senior_pension) :- over65(X), paid_up(X).
R3. pension(X, supplemental_pension) :- over65(X), paid_up(X).
R4. entitled(X, nothing) :- !+(pension(X, Pension)).

R5. entitled(X, nothing).%\+

F1. disabled(joe).
F2. over65(joe).
F3. paid_up(joe).
F4. over65(lou).
F5. paid_up(lou).

Q1. ?- entitled(joe, nothing) = no.
Q3. ?- entitled(X, senior_pension) :- 1. X = joe 2. X = lou.
Q4. ?- entitled(X, disabled_pension) :- 1. X = joe.

Is it Good, Bad or Just Ugly?

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

not(P).