lessThan vs notLessThan

- **notLessThan( \( \alpha, \beta \) )** \{ is *true* succeeds \}  
  whenever  
  **lessThan( \( \alpha, \beta \) )** \{ is *false* fails \}  
  (...and vice versa ...)  

- DisAdvantages:  
  - Two predicates, doing the work of one  
    Unnatural  
    Error-prone  
  - Inefficient  
    (Must do computation twice)  

- Common situation:
Intersection of Two Lists

• Axioms
  \[\text{inter}([], Y, []).\]
  \[\text{inter}([X|L], Y, [X|Z]) :- \text{member}(X,Y), \text{inter}(L,Y,Z).\]
  \[\text{inter}([X|L], Y, Z) :- \text{nonmember}(X,Y), \text{inter}(L,Y,Z).\]

• Need \text{nonmember}:

  \[
  \text{nonmember}(\alpha, \beta) \begin{cases} 
    \text{is true} & \text{succeeds} \\
    \text{is false} & \text{fails}
  \end{cases}
  \]

  whenever

  \[
  \text{member}(\alpha, \beta) \begin{cases} 
    \text{is false} & \text{fails}
  \end{cases}
  \]
Special Facilities: \(+\)

- Define nonmember as
  \[\text{nonmember}(X, Y) :\quad \vdash (\text{member}(X, Y)).\]
  \[\downarrow \text{special fn symbol operator}\]

- Goal \[\vdash \neg(P)\] succeeds if \(P\) fails
  fails if \(P\) succeeds

- “Negation as Failure”
  Assume “I don’t know” \(\not\sim\) “no”
  \textit{Not} true negation.
Where can "\+" be used?

• Allowed in RHS of rule

\[
a : - \ +(b). \quad a(X) : - c(X), \ +(b(X)).
\]

or in Goal

\[
\ +(b(5)) \quad \text{foo}(X), \ +(b(X))
\]

• NOT allowed in HEAD of rule, or Fact:

\[
(*) \quad \ +( \ good(mary) ). \quad (*)
\]

\[
(*) \quad \ +( \ good(X) ) : - \ bad(X). \quad (*)
\]
Examples of Negation

- append(X, Y, [a]), \+(=(X, []))
  succeeds with X=[a], Y=[] only.

- Axioms for inter
  
  inter([], Y, []).  
  inter([X|L], Y, [X|Z]) :- member(X, Y), inter(L, Y, Z).
  inter([X|L], Y, Z) :- \+(member(X, Y)), inter(L, Y, Z).

  inter([a,b,c], [c,a,d], X )
  returns  X/[a,c].

Problem: Have to perform

  member(\ldots)

twice, per iteration!

(One for positive, One for negative)
Special Facility:  Cut Symbol: !

- In general:

\[
\begin{align*}
P &: Q, R. \\
P &: \neg+(Q), S. \\
\end{align*}
\]

requires trying \[Q\] twice!

Solution: [cut symbol: !]

\[
\begin{align*}
P &: Q, !, R. \\
P &: S. \\
\end{align*}
\]

Behaves like

\[
P : \text{if } Q \text{ then } R \text{ else } S
\]

If Q succeeds, pursue \[R\]
but \[S\] is NOT attempted!

(If Q fails, then try \[S\].)
Eden Example – Version 1

$$KB_1 = \begin{cases} 
(1) & \text{num-pars}(X,Y) :- \text{eden}(X), =\!(Y, 0). \\
(2) & \text{num-pars}(X,Y) :- =\!(Y, 2). \\
(3) & \text{eden}(adam). \\
(4) & \text{eden}(eve). 
\end{cases}$$

Goal \textbf{num-pars}(adam,N):

Using (1) \(\Rightarrow\) \textbf{eden}(adam), =\!(N,0).

Using (3) \(\Rightarrow\) =\!(N,0).

\(\Rightarrow\) success: \{N/0\}

[If user asks for another solution]

Using (2) \(\Rightarrow\) =\!(N,2).

\(\Rightarrow\) success: \{N/2\}

[If user asks for another solution]

\(\Rightarrow\) \(\times\)
Eden Example #1a – Graphically

\[ KB_1 = \begin{cases} 
(1) & \text{num-pars}(X,Y) :- \text{eden}(X), =(Y, 0). \\
(2) & \text{num-pars}(X,Y) :- =((Y, 2)). \\
(3) & \text{eden}(\text{adam}). \\
(4) & \text{eden}(\text{eve}). 
\end{cases} \]
Eden Example #1b — Graphically

\[ KB_1 = \]
\[
\begin{align*}
(1) & \quad \text{num-pars}(X,Y) :- \text{eden}(X), = (Y, 0). \\
(2) & \quad \text{num-pars}(X,Y) :- = (Y, 2). \\
(3) & \quad \text{eden}(\text{adam}). \\
(4) & \quad \text{eden}(\text{eve}).
\end{align*}
\]
Eden Example – Version 2

\[ KB_2 = \]
\[
\begin{align*}
(1') & \quad \text{num-pars(}X,Y) \leftarrow \text{eden}(X), !, = (Y, 0). \\
(2) & \quad \text{num-pars}(X,Y) \leftarrow = (Y, 2). \\
(3) & \quad \text{eden}\text{(adam).} \\
(4) & \quad \text{eden}\text{(eve).}
\end{align*}
\]

**Goal** \[ \text{num-pars(adam,N)}: \]

Using \( (1') \) \( \leadsto \) \[ \text{eden(adam), !, = (N,0)}. \]

Using \( (3) \) \( \leadsto \) \[ !, = (N,0) \]

[Commits to this branch, for \( \text{num-pars(adam,N)} \) goal. Throws away 1 other subgoal, from \( (2) \)]

\( \leadsto \) \[ = (N,0) \]

\( \leadsto \) \[ \text{success: \{N/0\}} \]

[If user asks for another solution]

\( \leadsto \) \[ \times \]

**N.B:** Prolog does NOT now consider \( (2) \) branch!

...does NOT return \{N/2\}
Eden Example #2a – Graphically

\[ KB_2 = \]

\[
(1') \quad \text{num-pars}(X,Y) :- \text{eden}(X)!,=(Y,0).
\]

\[
(2) \quad \text{num-pars}(X,Y) :- =(Y,2).
\]

\[
(3) \quad \text{eden}(\text{adam}).
\]

\[
(4) \quad \text{eden}(\text{eve}).
\]
Eden Example #2b – Graphically

\[ KB_2 = \]
\[
\begin{align*}
(1') & \quad \text{num-pars}(X, Y) : \text{eden}(X), !, = (Y, 0). \\
(2) & \quad \text{num-pars}(X, Y) : = (Y, 2). \\
(3) & \quad \text{eden}(\text{adam}). \\
(4) & \quad \text{eden}(\text{eve}).
\end{align*}
\]

\[
\text{num-pars}(P, N)
\]
\[
\begin{align*}
(1) & \quad \{ X_1/P, Y_1/N \} \\
(2) & \quad \{ P/X_1, Y_1/N \} \\
(3) & \quad \{ P/\text{adam} \} \\
(4) & \quad \{ P/\text{eve} \}
\end{align*}
\]

\[
\text{eden}(P), = (N, 0)
\]

\[
! = (N, 0)
\]

Ignore ALL other possible branches under \text{num-pars}(P, N)\ goal!

\[
= (N, 0)
\]

\[
\text{success} \quad \{ P/\text{adam}, N/0 \}
\]
Other Answers

Eden Example

Using $KB_2$:

Goal \textbf{num-pars(fred,N)}:

Using (1') $\leadsto$ \textbf{eden(fred), !, =}(N,0).
$\leadsto \times$

Using (2) $\leadsto$ \textbf{=(N,2)}.
$\leadsto$ \textbf{success: \{N/2\}}

[If user asks for another solution]
$\leadsto \times$

[Same answer using $KB_1$]

\begin{align*}
\textbf{num-pars(P,0)} & \leadsto \{\text{P/adam}\} \text{ only} \\
\textbf{num-pars(P,2)} & \leadsto \times
\end{align*}
Comments on Eden Example

• Order is IMPORTANT!

\[ KB_3 = \begin{cases} 
\text{num-pars}(X,Y) :& \equiv (Y, 2). \\
\text{num-pars}(X,Y) :& \equiv \text{eden}(X), !, \equiv (Y, 0). \\
\text{eden}(adam) :& \equiv \text{eden}(eve). 
\end{cases} \]

Goal: \fbox{num-pars(adam,N)} returns
- \{N/2\}
- \{N/0\}

• Tempting Variant:

\[ KB_4 = \begin{cases} 
(1'') \quad \text{num-pars}(adam,0) :& \equiv !. \\
(2'') \quad \text{num-pars}(eve,0) :& \equiv !. \\
(3'') \quad \text{num-pars}(X,2). 
\end{cases} \]

Goal: \fbox{num-pars(adam,N)} returns
- \{N/0\}

Goal: \fbox{num-pars(adam,2)} returns
- \text{success!} \quad \text{(matches (3''))}
"Scope" of Cut

\[ KB_5 = \]

\[
\begin{align*}
(1) & \quad \text{num-pars}(X, Y) :- \text{eden}(X), !, = (Y, 0). \\
(2) & \quad \text{num-pars}(X, 2) :- \text{person}(X). \\
(3) & \quad \text{normal}(X) :- \text{num-pars}(X, 2). \\
(4) & \quad \text{person}(\text{george}). \\
(5) & \quad \text{normal}(\text{fred}). \\
(6) & \quad \text{eden}(\text{adam}).
\end{align*}
\]

Ignore ALL other possible branches under \text{num-pars}(P,2) goal!
"Scope" of Cut

\[ KB_5 = \]
\[
\begin{align*}
(1) & \quad \text{num-pars}(X,Y) :- \text{eden}(X), !, \equiv(Y, 0). \\
(2) & \quad \text{num-pars}(X,2) :- \text{person}(X). \\
(3) & \quad \text{normal}(X) :- \text{num-pars}(X,2). \\
(4) & \quad \text{person}(\text{george}). \\
(5) & \quad \text{normal}(\text{fred}).
\end{align*}
\]

Diagram:

- **normal\(\text{(P)}\)**
  - (3)
  - **num-pars\(\text{(P,2)}\)**
  - (1)
    - **eden\(\text{(P), !, =}(2, 0)\)**
    - (2)
      - **person\(\text{(P)}\)**
      - (4)
        - **success \{ P/fred \}**
  - (5)
    - **success \{ P/george \}**
(6)  eden(adam).

(6)
{P/adam}
!! = (2, 0)

Ignore ALL other possible branches under num-pars(P,2) goal!

= (2, 0)
×
(Formal) Description of Cut

• ![subgoal always succeeds, but Side-Effect: Prevents future backtracking! (By throwing away ALL other backtracking choices for present goal!)

• Within Rule:
  Like ONE-WAY GATE:
  Can try bindings until reaching “!”
  Then stick with FIRST VALUE found.

• Among Rules:
  Like SELECTOR:
  If “!” is reached,
  Then eliminate all other rules (w/same head)
Description of Cut (con't)

\[ KB = \]

(1) \( P_1 \leftarrow Q_1, \ldots, Q_n, !, R_1, \ldots, R_m. \)

(2) \( P_2 \leftarrow S_1^2, \ldots, S_{q2}^2. \)

\[ \ldots \]

(p) \( P_p \leftarrow S_1^p, \ldots, S_{q_p}^p. \)

If goal \( \boxed{P} \) matches \( P_1 \), and if \( \boxed{Q_1, \ldots, Q_n} \) all pass (with \( \sigma \))

Then \( Prolog \) will only use this \( \sigma \) on

\( \boxed{R_1\sigma, \ldots, R_m\sigma} \)

and \( Prolog \) will ignore ALL other \( P_i \) rules!

EVEN if \( \boxed{R_1\sigma, \ldots, R_m\sigma} \) fails!

(Ie, \( Prolog \) commits to CURRENT clause only.)
Procedure

Given goal $P$, which unifies with $P_1, P_2, \ldots P_p$:

1. Using (1): $P_1 := Q_1, \ldots, Q_n, !, R_1, \ldots, R_m$.
   
   Try to satisfy $Q_1, Q_2, \ldots, Q_n$.

2. If find a binding, $\sigma$,
   
   use it in $R_1\sigma, \ldots, R_m\sigma$

   Ignore ALL other bindings for $Q_1, Q_2, \ldots, Q_n$!

   Ignore ALL other facts/rules.

   (Ie, ignore $S_1^i, \ldots, S_{qi}^i$.)

   Even if $R_1\sigma, \ldots, R_m\sigma$ fails!

3. If no bindings for $Q_1, Q_2, \ldots, Q_n$,
   
   then try other facts/rules.

   (Ie, try $S_1^i, \ldots, S_{qi}^i$.)
Comments on Cut

• Many “!”-literals within rule
  (ie, many \( R_i \)s can be “!”)
  Each is “FENCE”.

• Many rules (with \( P_i \) head) can have “!”s;
  They are executed in order
  When “!” reached and “achieved”,
  ALL remaining rules are dropped.

• “!” is strictly procedural

  No declarative Semantics
Uses of Cut Symbol

• Define \texttt{naf} predicate $\equiv \ \setminus +$
\[
\text{naf}(G) :- G, !, \text{fail.} \\
\text{naf}(G).
\]
If $G$ succeeds, then committed to failure.
otherwise, will succeed.

• Make \texttt{member} look for only $1^{st}$ occurrence.
\[
\text{member}(X, [X | \_]) :- !. \\
\text{member}(X, [\_ | L]) :- \text{member}(X, L).
\]
If first rule succeeds,
will ignore second rule.
[I.e, will not examine rest of list.]

\[
\begin{array}{ll}
\text{member}(c, [a, b, c, d]) & \text{member}(X, [a, b, c, d])
\end{array}
\]

• Remember to deal with both success & failure!
... variables ...