`lessThan` **VS** `notLessThan`

- **notLessThan(** α**,** β **)** $\left\{\begin{array}{l}\text{is } \textit{true}\\ \text{succeeds}\end{array}\right\}$

  whenever

  lessThan(α**,** β **)** $\left\{\begin{array}{l}\text{is } \textit{false}\\ \text{fails}\end{array}\right\}$

  (. . . 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

```
inter([], Y, []).
inter([X|L], Y, [X|Z]) :- member(X,Y), inter(L,Y,Z).
inter([X|L], Y, Z) :- nonmember(X,Y), inter(L,Y,Z).
```

- Need `nonmember` :

$$\textbf{nonmember(}\ \alpha,\ \beta\ \textbf{)} \quad \left\{\begin{array}{l} \text{is } \textit{true} \\ \text{succeeds} \end{array}\right\}$$

whenever

$$\textbf{member(}\ \alpha,\ \beta\ \textbf{)} \quad \left\{\begin{array}{l} \text{is } \textit{false} \\ \text{fails} \end{array}\right\}$$

Special Facilities: \+

- Define `nonmember` as

  nonmember(X,Y) :- \+(member(X,Y)).

  ↖ special fn symbol

  operator

- Goal `\+(P)` succeeds if `P` fails

  fails if `P` succeeds

- "Negation as Failure"

  Assume "I don't know" $\leadsto$ "no"

  Not true negation.

Where can "\+" be used?

- Allowed in RHS of rule

```
a :- \+(b).     a(X) :- c(X), \+(b(X)).
```

or in Goal

```
\+(b(5))        foo(X), \+(b(X))
```

- NOT allowed in HEAD of rule, or Fact:

(*)   \\+(good(mary)).   (*)
(*)   \\+(good(X)) :- bad(X).   (*)

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(···)`

twice, per iteration!

(One for positive, One for negative)

Special Facility: Cut Symbol: !

- In general:

```
P :- Q, R.
P :- \+(Q), S.
```

requires trying Q twice!

Solution: **cut symbol: !**

```
P :- Q, !, R.
P :- S.
```

Behaves like

```
P :- if Q   then R
            else S
```

If `Q` succeeds, pursue `R`
but `S` is NOT attempted!

(If `Q` fails, then try `S`.)

Eden Example – Version 1

$KB_1 \ =$

$$\left\{ \begin{array}{ll} \textit{(1)} & \text{num-pars(X,Y) :- eden(X), =(Y, 0).} \\ \textit{(2)} & \text{num-pars(X,Y) :- =(Y, 2).} \\ \textit{(3)} & \text{eden(adam).} \\ \textit{(4)} & \text{eden(eve).} \end{array} \right\}$$

Goal $\boxed{\texttt{num-pars(adam,N)}}$:

Using *(1)* $\leadsto$ $\boxed{\texttt{eden(adam), =(N,0)}}$.

Using *(3)* $\leadsto$ $\boxed{\texttt{=(N,0)}}$.

$\leadsto$ $\boxed{\texttt{success: } \{\texttt{N/0}\}}$

[If user asks for another solution]

Using *(2)* $\leadsto$ $\boxed{\texttt{=(N,2)}}$.

$\leadsto$ $\boxed{\texttt{success: } \{\texttt{N/2}\}}$

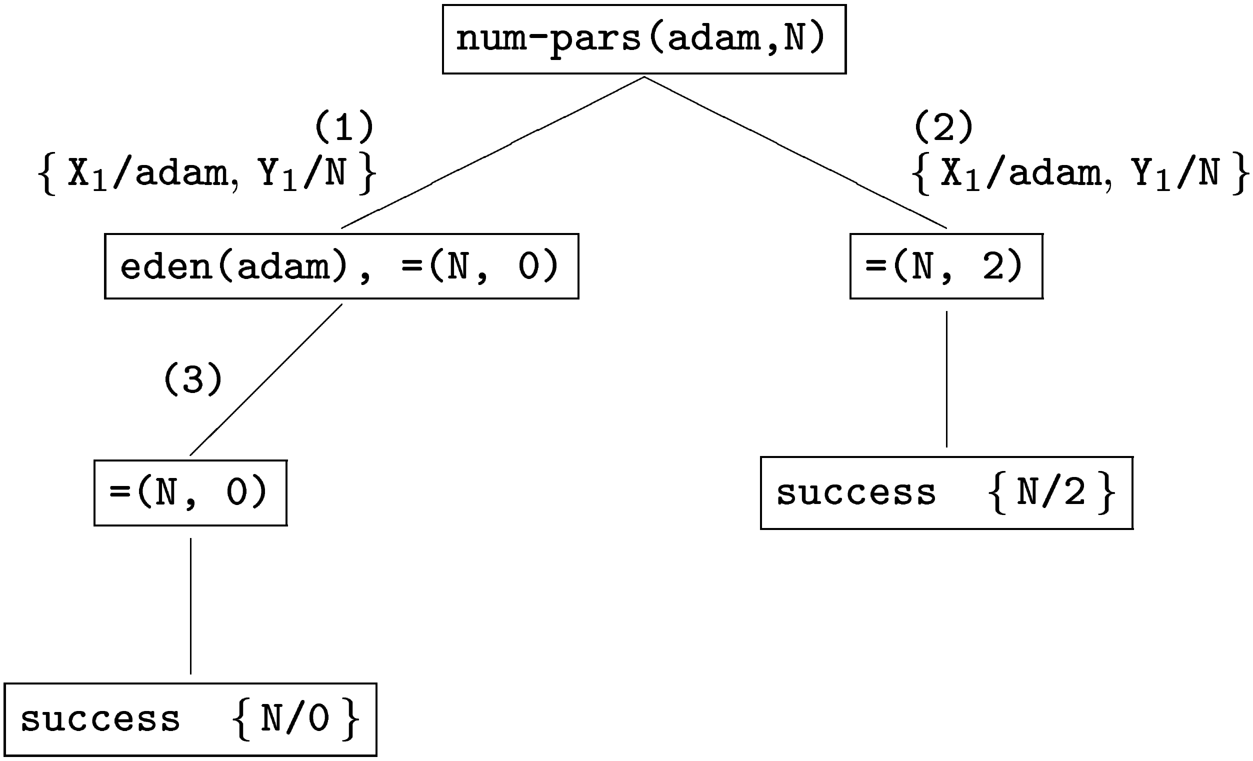
[If user asks for another solution]

$\leadsto$ $\times$

Eden Example #1a – Graphically

$KB_1 \; =$

$$\left\{ \begin{array}{ll} \textit{(1)} & \text{num-pars(X,Y) :- eden(X), =(Y, 0).} \\ \textit{(2)} & \text{num-pars(X,Y) :- =(Y, 2).} \\ \textit{(3)} & \text{eden(adam).} \\ \textit{(4)} & \text{eden(eve).} \end{array} \right\}$$



num-pars(adam,N)

(1)

{ X_1/adam, Y_1/N }

(2)

{ X_1/adam, Y_1/N }

eden(adam), =(N, 0)

=(N, 2)

(3)

=(N, 0)

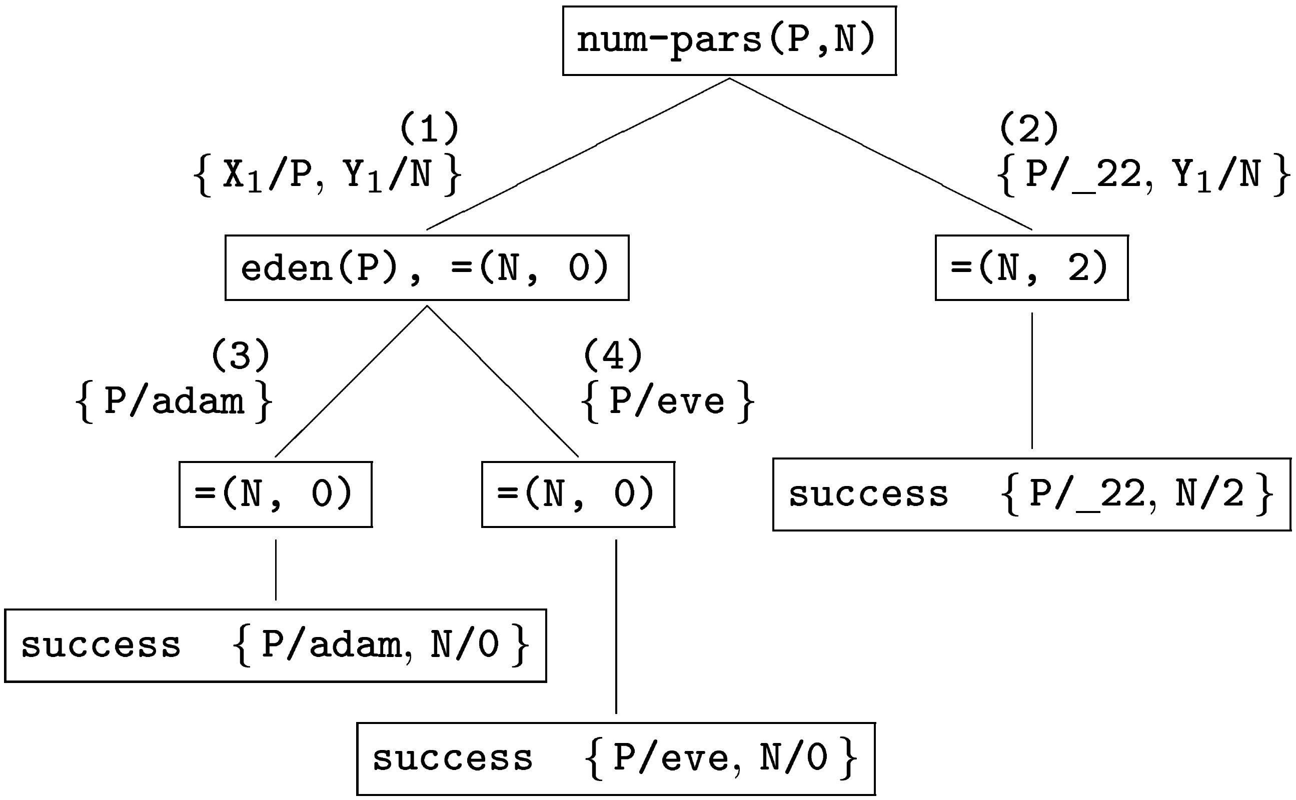
success {N/2}

success {N/0}

Eden Example #1b – Graphically

KB_1 =

(1) num-pars(X,Y) :- eden(X), =(Y, 0).
(2) num-pars(X,Y) :- =(Y, 2).
(3) eden(adam).
(4) eden(eve).



num-pars(P,N)

(1) { X_1/P, Y_1/N }

eden(P), =(N, 0)

(2) { P/_22, Y_1/N }

=(N, 2)

(3) { P/adam }

=(N, 0)

(4) { P/eve }

=(N, 0)

success { P/_22, N/2 }

success { P/adam, N/0 }

success { P/eve, N/0 }

Eden Example – Version 2

$$KB_2 =$$

$$\left\{ \begin{array}{ll} \textit{(1')} & \text{num-pars(X,Y) :- eden(X), !, =(Y, 0).} \\ \textit{(2)} & \text{num-pars(X,Y) :- =(Y, 2).} \\ \textit{(3)} & \text{eden(adam).} \\ \textit{(4)} & \text{eden(eve).} \end{array} \right\}$$

Goal $\boxed{\texttt{num-pars(adam,N)}}$:

Using *(1')* $\rightsquigarrow$ $\boxed{\texttt{eden(adam), !, =(N,0)}}$.

Using *(3)* $\rightsquigarrow$ $\boxed{\texttt{!, =(N,0)}}$

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

$\rightsquigarrow$ $\boxed{\texttt{=(N,0)}}$

$\rightsquigarrow$ $\boxed{\texttt{success: } \{\texttt{N/0}\}}$

[If user asks for another solution]

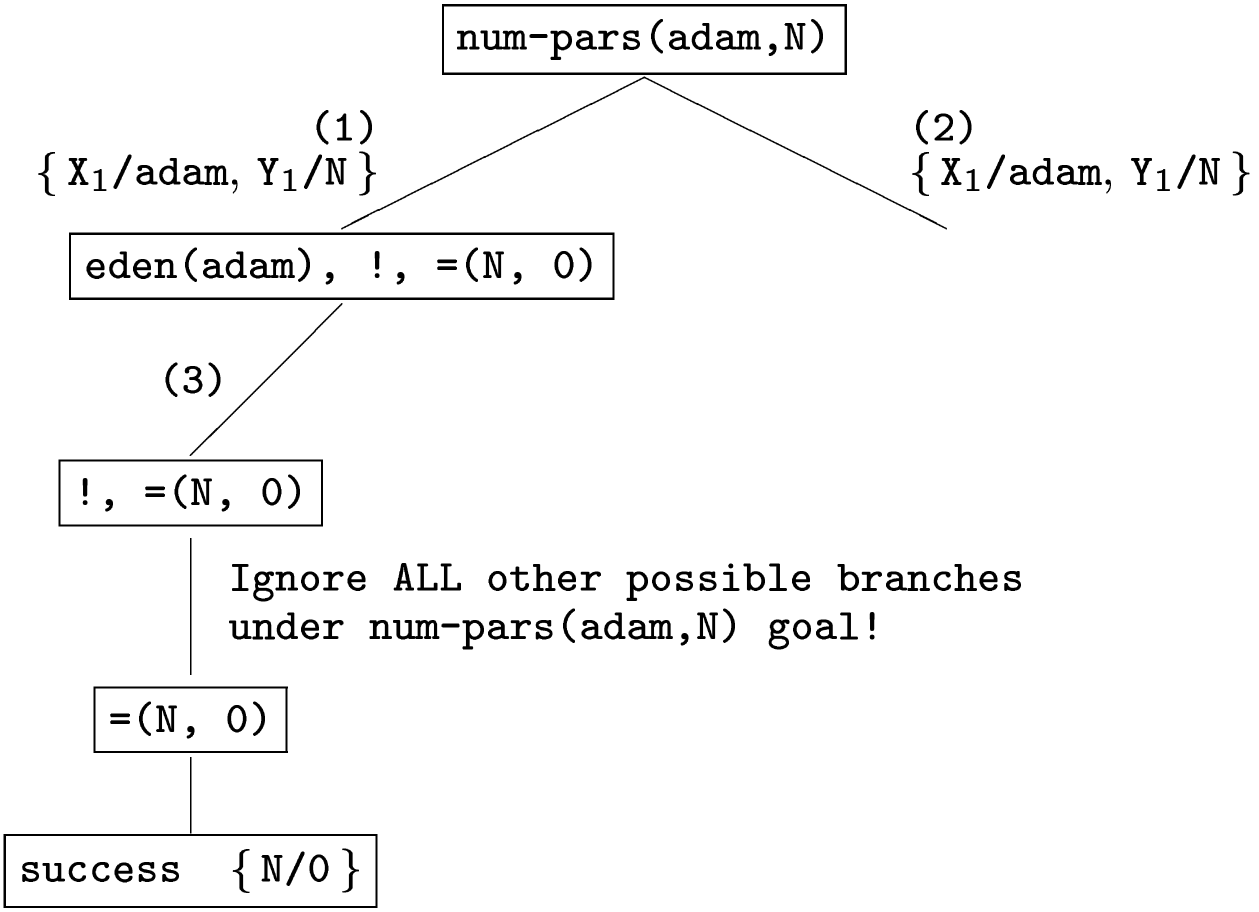
$\rightsquigarrow \times$

N.B: *Prolog* does NOT now consider *(2)* branch!
. . . does NOT return $\{\texttt{N/2}\}$

Eden Example #2a – Graphically

$KB_2 =$

(1') num-pars(X,Y) :- eden(X), !, =(Y, 0).
(2) num-pars(X,Y) :- =(Y, 2).
(3) eden(adam).
(4) eden(eve).



num-pars(adam,N)

(1)
{ X_1/adam, Y_1/N }

(2)
{ X_1/adam, Y_1/N }

eden(adam), !, =(N, 0)

(3)

!, =(N, 0)

Ignore ALL other possible branches
under num-pars(adam,N) goal!

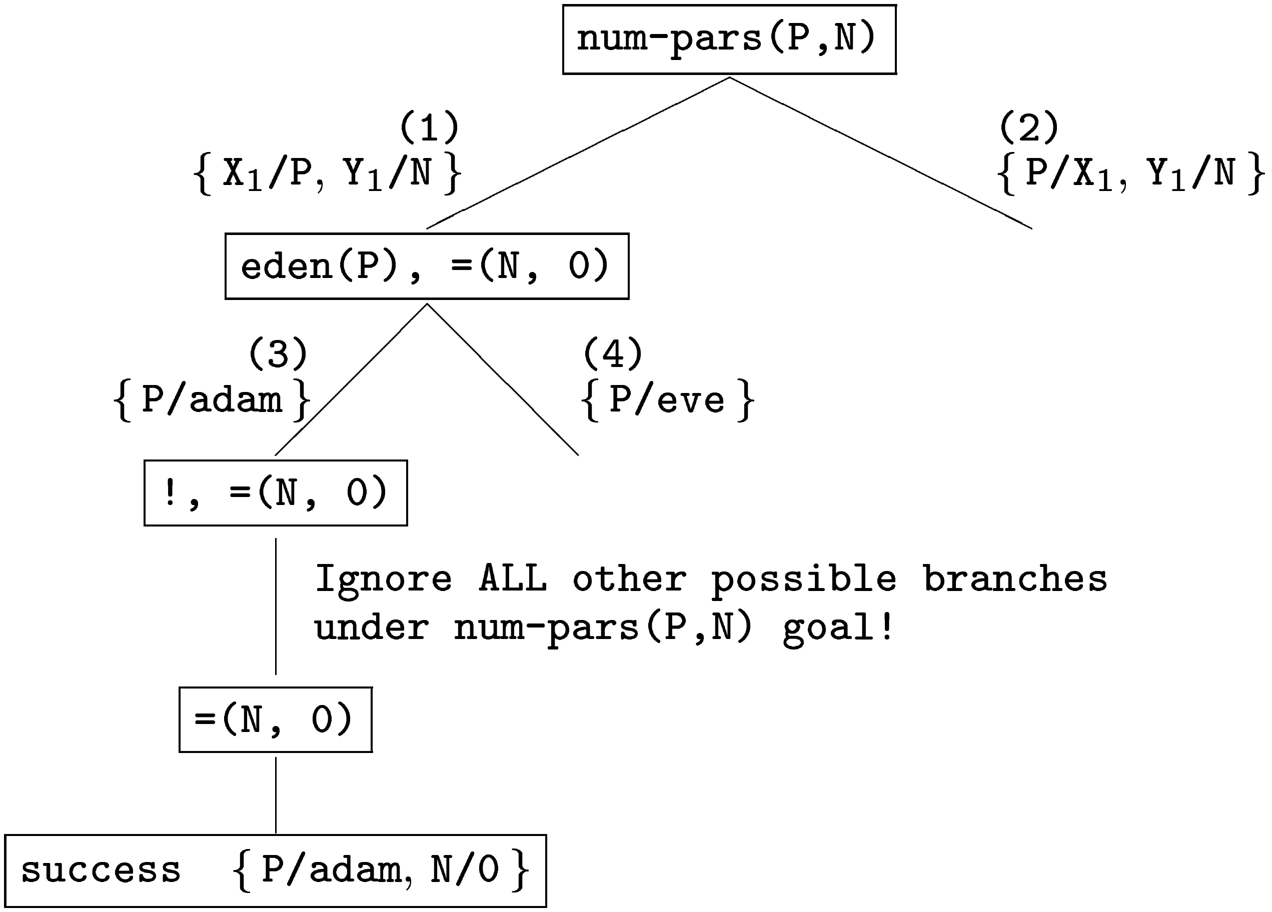
=(N, 0)

success { N/0 }

Eden Example #2b – Graphically

$KB_2 =$

$\left\{ \begin{array}{ll} \textit{(1')} & \text{num-pars(X,Y) :- eden(X), !, =(Y, 0).} \\ \textit{(2)} & \text{num-pars(X,Y) :- =(Y, 2).} \\ \textit{(3)} & \text{eden(adam).} \\ \textit{(4)} & \text{eden(eve).} \end{array} \right\}$



num-pars(P,N)

(1) $\{ X_1/P, Y_1/N \}$

(2) $\{ P/X_1, Y_1/N \}$

eden(P), =(N, 0)

(3) { P/adam }

(4) { P/eve }

!, =(N, 0)

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

=(N, 0)

success { P/adam, N/0 }

Other Answers Eden Example

Using KB_2:

Goal $\boxed{\texttt{num-pars(fred,N)}}$:

Using *(1')* $\rightsquigarrow \boxed{\texttt{eden(fred), !, =(N,0)}}$.

$\rightsquigarrow \times$

Using *(2)* $\rightsquigarrow \boxed{\texttt{=(N,2)}}$.

$\rightsquigarrow \boxed{\texttt{success: } \{\texttt{N/2}\}}$

[If user asks for another solution]

$\rightsquigarrow \times$

[Same answer using KB_1]

$\boxed{\texttt{num-pars(P,0)}} \quad \rightsquigarrow \quad \{\texttt{P/adam}\}$ only

$\boxed{\texttt{num-pars(P,2)}} \quad \rightsquigarrow \quad \times$

Comments on Eden Example

- Order is IMPORTANT!

$$KB_3 = \left\{ \begin{array}{ll} \text{num-pars(X,Y) :- =(Y, 2).} & \\ \text{num-pars(X,Y) :- eden(X), !, =(Y, 0).} & \\ \text{eden(adam).} & \text{eden(eve).} \end{array} \right\}$$

Goal: `num-pars(adam,N)` returns

  – { `N/2` }

  – { `N/0` }

- Tempting Variant:

$$KB_4 = \left\{ \begin{array}{ll} \textit{(1'')} & \text{num-pars(adam,0) :- !.} \\ \textit{(2'')} & \text{num-pars(eve,0) :- !.} \\ \textit{(3'')} & \text{num-pars(X,2).} \end{array} \right\}$$

Goal: `num-pars(adam,N)` returns

  – { `N/0` }

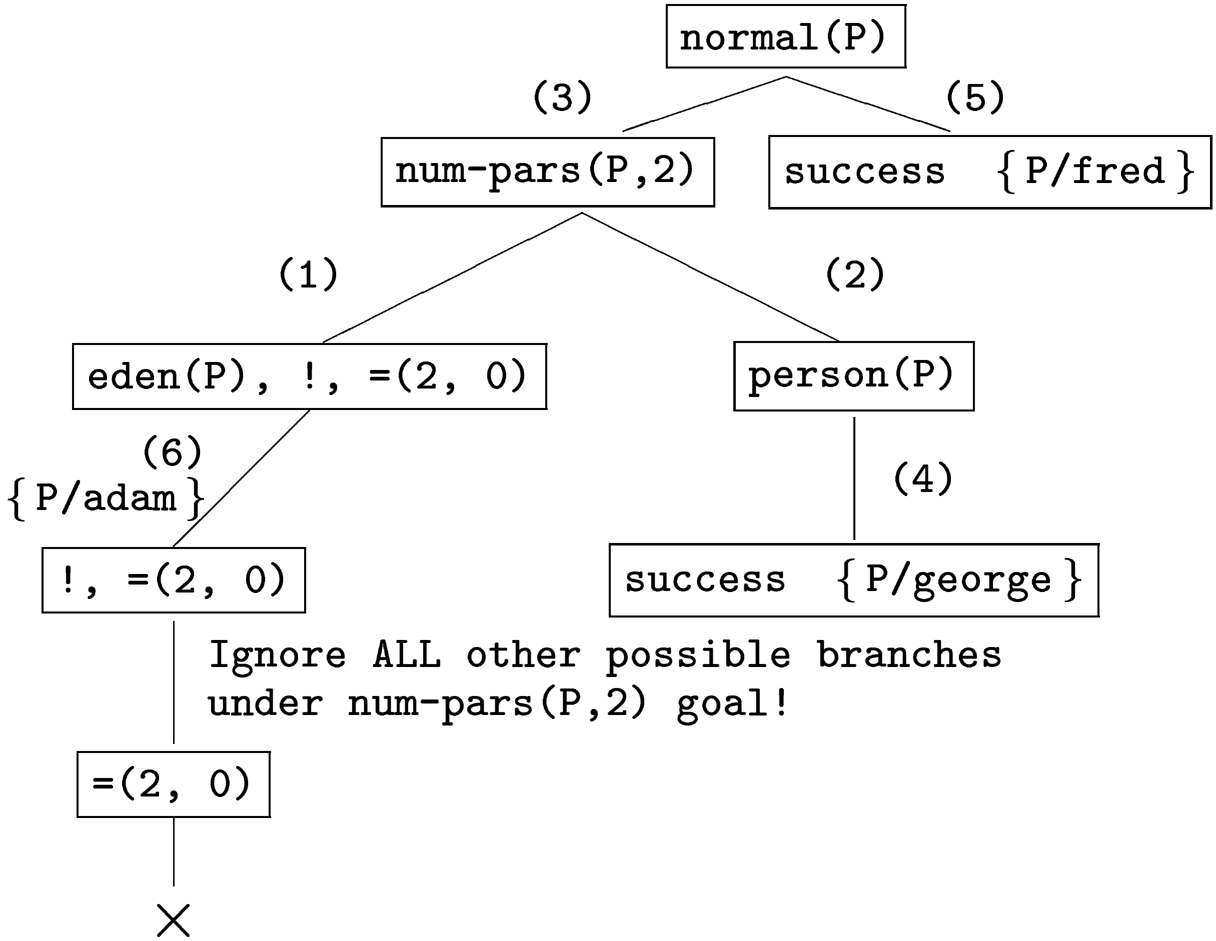
Goal: `num-pars(adam,2)` returns

  – `success!` (matches *(3")*)

“Scope” of Cut

$KB_5 =$

(1) num-pars(X,Y) :- eden(X), !, =(Y, 0).
(2) num-pars(X,2) :- person(X).
(3) normal(X) :- num-pars(X,2).
(4) person(george).
(5) normal(fred).
(6) eden(adam).



normal(P)

(3) num-pars(P,2)

(5) success {P/fred}

(1) eden(P), !, =(2, 0)

(2) person(P)

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

(4) success {P/george}

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

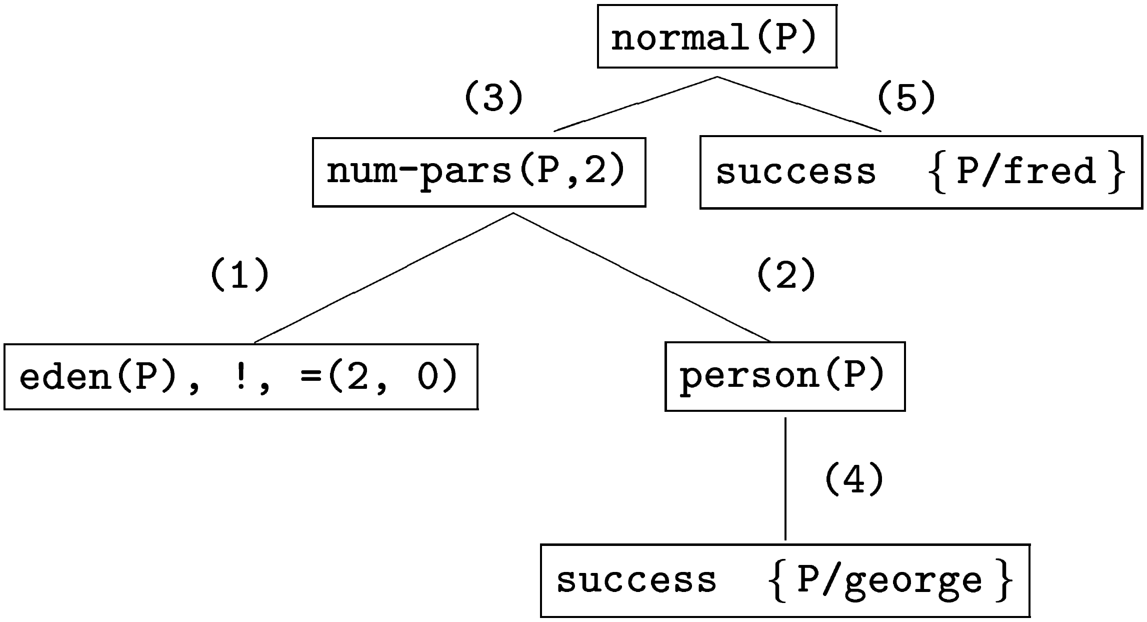
=(2, 0)

×

“Scope” of Cut

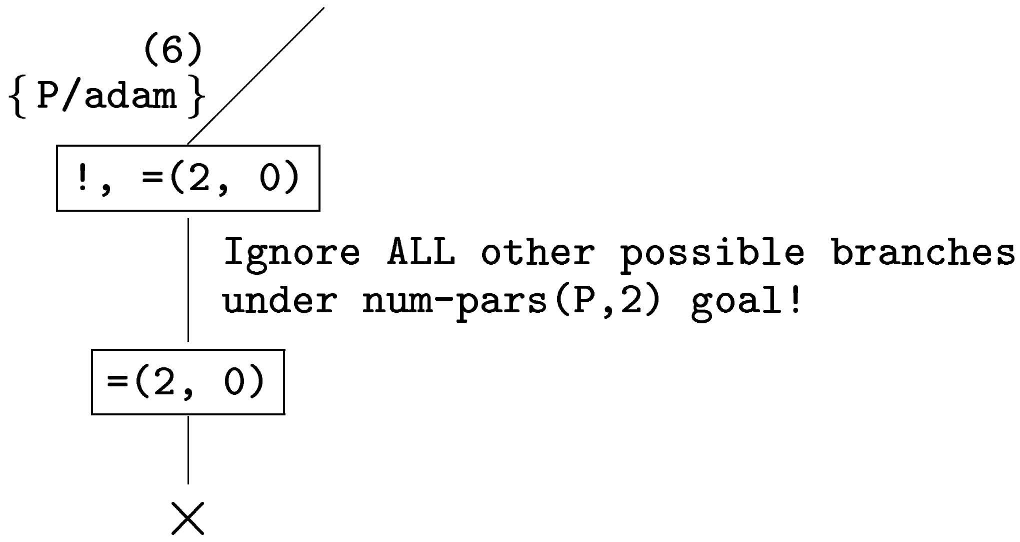
$KB_5 =$

{
(1) num-pars(X,Y) :- eden(X), !, =(Y, 0).
(2) num-pars(X,2) :- person(X).
(3) normal(X) :- num-pars(X,2).
(4) person(george).
(5) normal(fred).
}



normal(P)
(3)
(5)
num-pars(P,2)
success {P/fred}
(1)
(2)
eden(P), !, =(2, 0)
person(P)
(4)
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

- $\boxed{!}$ 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$_1$:- Q$_1$, . . . , Q$_n$, !, R$_1$, . . . , R$_m$.
(2) P$_2$:- S$_1^2$, . . . , S$_{q2}^2$.
. . .
(p) P$_p$:- S$_1^p$, . . . , S$_{qp}^p$.

If goal $\boxed{\texttt{P}}$ matches `P`$_1$,

and if $\boxed{\texttt{Q}_1\texttt{, }\cdots\texttt{, Q}_n}$ all pass (with σ)

Then *Prolog* will only use this σ on

$\boxed{\texttt{R}_1\sigma\texttt{, }\cdots\texttt{, R}_m\sigma}$

and *Prolog* will ignore ALL other `P`$_i$ rules!

EVEN if $\boxed{\texttt{R}_1\sigma\texttt{, ..., R}_m\sigma}$ fails!

(Ie, *Prolog* commits to **CURRENT** clause only.)

! Procedure

Given goal $\boxed{\texttt{P}}$,
which unifies with $\texttt{P}_1$, $\texttt{P}_2$, . . . $\texttt{P}_p$:

1. Using *(1)*: P_1 :- Q_1, . . . , Q_n, !, R_1, . . . , R_m.

   Try to satisfy $\boxed{\texttt{Q}_1\texttt{, Q}_2\texttt{, ..., Q}_n}$.

2. If find a binding, σ,
   use it in $\boxed{\texttt{R}_1\sigma\texttt{, ..., R}_m\sigma}$

   Ignore ALL other bindings for $\boxed{\texttt{Q}_1\texttt{, Q}_2\texttt{, ..., Q}_n}$!

   Ignore ALL other facts/rules.

   (Ie, ignore $\boxed{\texttt{S}^i_1\texttt{, ..., S}^i_{qi}}$.)

   Even if $\boxed{\texttt{R}_1\sigma\texttt{, ..., R}_m\sigma}$ fails!

3. If no bindings for $\boxed{\texttt{Q}_1\texttt{, Q}_2\texttt{, ..., Q}_n}$,
   then try other facts/rules.

   (Ie, try $\boxed{\texttt{S}^i_1\texttt{, ..., S}^i_{qi}}$.)

Comments on Cut

- Many “!”-literals within rule
  (ie, many $\mathtt{R}_i$s can be “!”)
  Each is “FENCE”.

- Many rules (with $\mathtt{P}_i$ head) can have “!”s;
  They are executed in order
  When “!” reached and “achieved”,
  ALL remaining rules are dropped.

- “!” is strictly procedural

$$\boxed{\text{No declarative Semantics}}$$

Uses of Cut Symbol

- Define **`naf`** predicate ≈ \+

```
naf(G) :- G, !, fail.
naf(G).
```

If `G` succeeds, then committed to failure.
otherwise, will succeed.

- Make `member` look for only 1^{st} occurrence.

```
member(X, [X | _]) :- !.
member(X, [_ | L]) :- member(X,L).
```

If first rule succeeds,
will ignore second rule.
[Ie, will not examine rest of list.]

`member(c,[a,b,c,d])` `member(X,[a,b,c,d])`

- Remember to deal with both
success & failure!
… variables …