WHAT IS PROLOG?

- Programming language based on predicate logic.
- Developed at Marseille.
- Implemented as an interpreter (written in Fortran).
- Provides :-
  - (Full) Unification
  - Non-determinism (Backtracking)

    } Linear inference system for, essentially, Horn clauses.
  - Extra control primitives
  - Database updating primitives
  - Input-output, arithmetic etc.

    } "Evaluable predicates".
- Speed on DEC10 :-

  ~ 270 unifications per second,

  ~ 30% slower than interpreted pure Lisp.
- Applications so far:-
  - natural language (French) understanding [Colmerauer et.al.]
  - symbolic differentiation and integration [Kanoui].
  - speech analysis from noisy phonemes [Battani + Meloni]
  - geometry theorem proving [Welham]
  - plan generation [Warren]
  - the Prolog front-end translator of source text

EXAMPLE

Predicate Logic Program

$$\text{Factorial}(0,1) \leftarrow .$$
$$\text{Factorial}(x,y) \leftarrow \text{Minus}(x,1,x_1), \text{Factorial}(x_1,y_1), \text{Mult}(x,y_1,y).$$

$$\leftarrow \text{Factorial}(10,y).$$

Corresponding Prolog [exactly as input]

```
- LIREFICHIER.

+ FACTORIAL(0,1).
+ FACTORIAL(*X,*Y) - MOINS(*X,1,*X1)
                   - FACTORIAL(*X1,*Y1)
                   - MULT(*X,*Y1,*Y).
+ FIN.

- FACTORIAL(10,*Y) - SORTER(*Y) - LIGNE.

3628800
```

Program [or Database]

① $+fact(0,1).$

② $+fact(X,Y) - minus(X,1,X_1) - fact(X_1,Y_1) - mult(X,Y_1,Y).$

③ $+minus(2,1,1).$

④ $+minus(1,1,0).$

⑤ $+mult(1,1,1).$

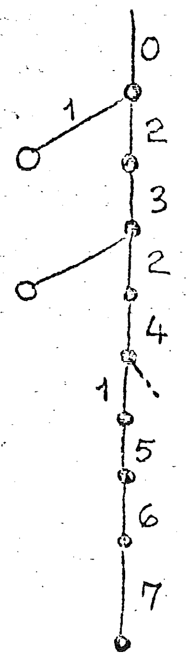
⑥ $+mult(2,1,2).$

⑦ $-ans(X) - sorter(X) - ligne.$

Command

⓪ $-fact(2,Y) + ans(Y).$

Search Space



$-fact(2,Y) + ans(Y).$

$-minus(2,1,X_1) - fact(X_1,Y_1) - mult(2,Y_1,Y) + ans(Y).$

$-fact(1,Y_1) - mult(2,Y_1,Y) + ans(Y).$

$-minus(1,1,X_2) - fact(X_2,Y_2) - mult(1,Y_2,Y_1) - mult(2,Y_1,Y) + ans(Y).$

$-fact(0,Y_2) - mult(1,Y_2,Y_1) - mult(2,Y_1,Y) + ans(Y).$

$-mult(1,1,Y_1) - mult(2,Y_1,Y) + ans(Y).$

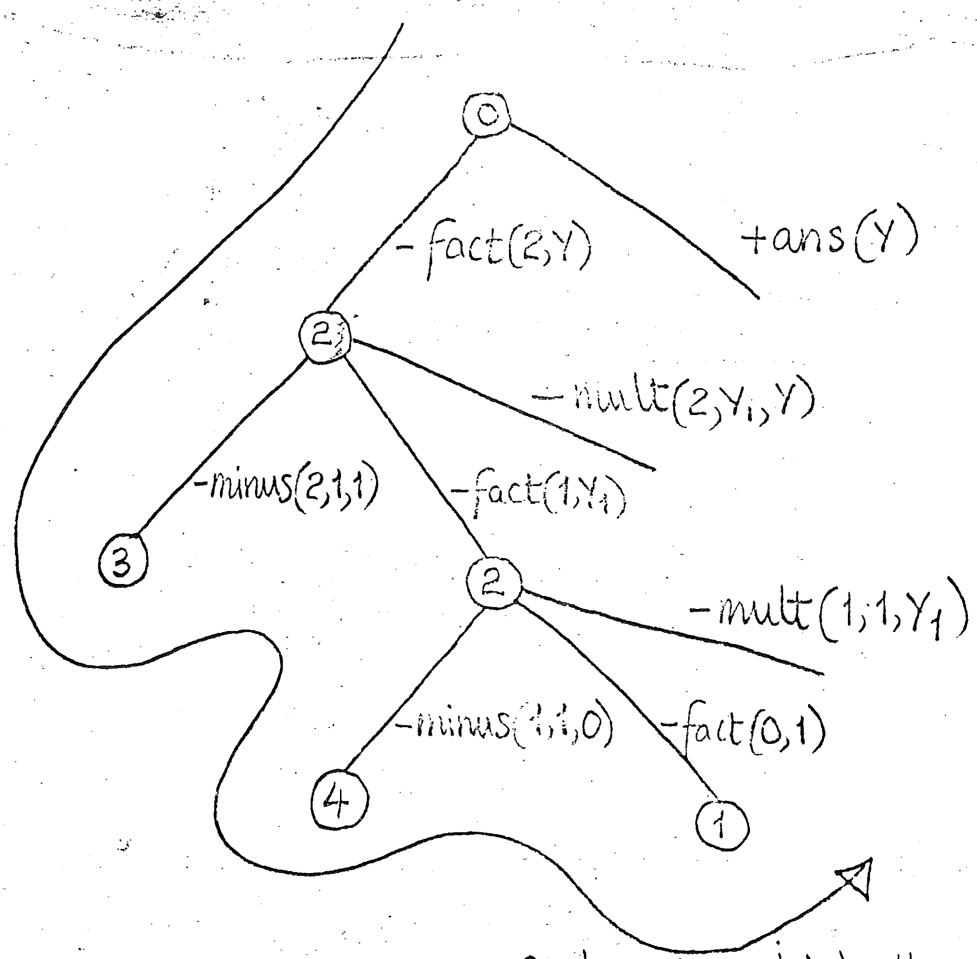
$-mult(2,1,Y) + ans(Y).$

$+ans(2).$

$\Rightarrow 2$

$\square$

The (Partially-Generated) Proof



Order in which the branches of the tree are constructed.

Corresponding Goal Statement (Command):-

$$-mult(1,1,Y_1) - mult(2,Y_1,Y) + ans(Y).$$

ANOTHER EXAMPLE

Program

$+ \cdot (dg, 5).$

ie. function '·' is a right-to-left (droite-à-gauche), precedence 5 operator.

ie. $X \cdot Y \cdot Z \cdot nil = X \cdot (Y \cdot (Z \cdot nil)) = \cdot(X, \cdot(Y, \cdot(Z, nil)))$

① + member(X, X·L).

② + member(X, Y·L) − member(X, L).

③ + printintersection(L1, L2) − member(X, L1) − member(X, L2) − report(X).

④ + report(X) − sorter(X) − ligne − fail.

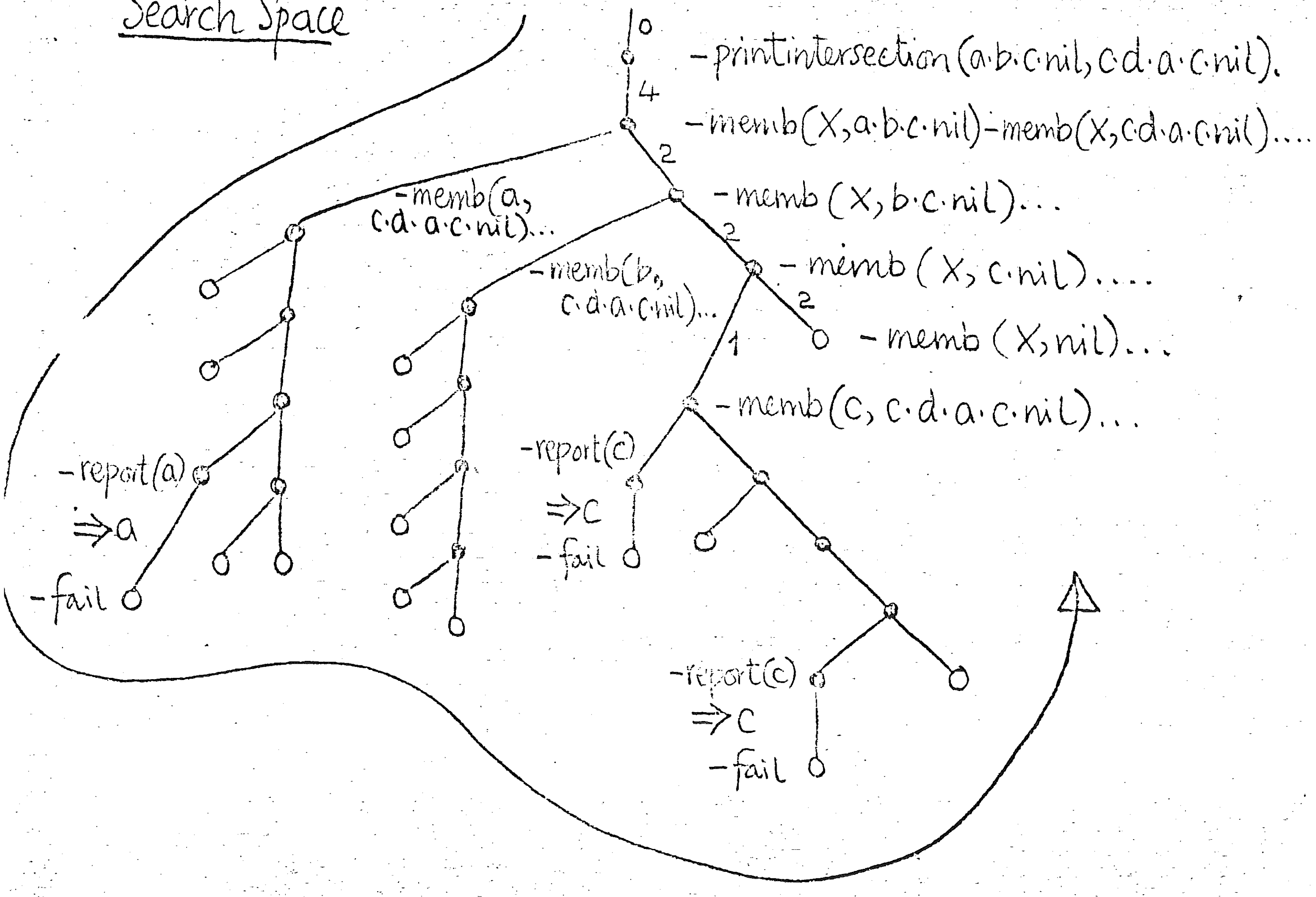
Command

⓪ − printintersection(a·b·c·nil, c·d·a·c·nil).

a
c
c
?
.

} complete output

Search Space



SUMMARY OF THE EVALUABLE PREDICATES AVAILABLE

Input-Output

- lu(C)
- lub(C)
- ecrit(C)
- ligne
- tty
- lirefichier
- booliste
- imprime
- sorcha(S)
- sorter(T)
- sauve
- stop

Arithmetic and Characters

- plus(X,Y,Z) ie. $X+Y=Z$
- moins(X,Y,Z) ie. $X-Y=Z$
- mult(X,Y,Z) ie. $X \times Y=Z$
- div(X,Y,Z) ie. $X/Y=Z$
- reste(X,Y,Z) ie. $X \bmod Y=Z$

- inf(X,Y) ie. $X<Y$

- lettre(C) ie. $C \in \{'A','B',\ldots,'Z'\}$
- chiffre(C) ie. $C \in \{'0','1',\ldots,'9'\}$
- blanc(C) ie. C=' '
- etoile(C) ie. C='*'
- virg(C) ie. C=','
- parg(C) ie. C='('
- pard(C) ie. C=')'
- guillemet(C) ie C='"'

Extra-Control

- /
- /(L)
- ancetre(L)
- etat(S)

Database Manipulation and Meta-predicates

- ajout(C)
- ajoutb(C)
- ajoutc(C)
- supp(C)
- X
- univ(T,D)
- atome(T,D)
- egalf(T_1,T_2)

THE EVALUABLE PREDICATE /

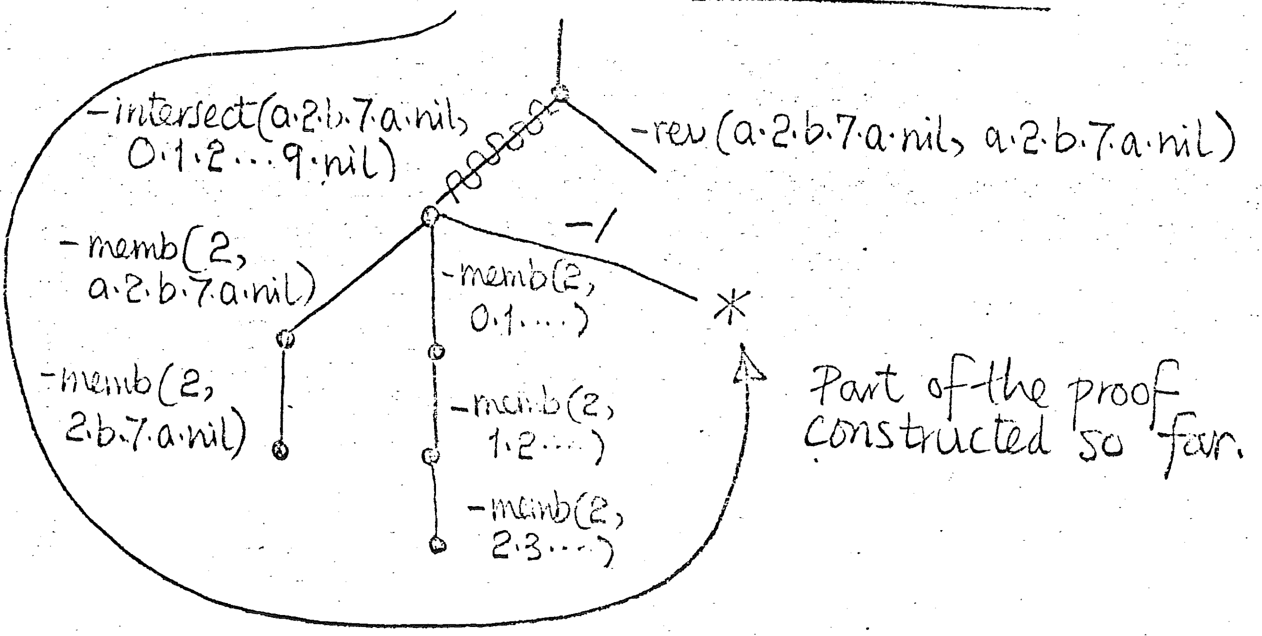
Program

```
+reverse(L0, L)-revappend(L0, nil, L).
+reverseappend(X·L0, L1, L)-reverseappend(L0, X·L1, L).
+reverseappend(nil, L0, L0).
+intersect(L1, L2)-member(X, L1)-member(X, L2)-/.
+palindromiclistcontainingadigit(L)
        -intersect(L, 0·1·2·3·4·5·6·7·8·9)
        -reverse(L, L).
```

Command

```
-palindromiclistcontainingadigit(a·2·b·7·a·nil).
```

State of the Proof when '-/' is encountered



If backtracking returns to this state, then skip back to the state immediately preceding the one in which the parent branch (marked) was expanded. [In this case, skip back to the initial state.].

The '/' prevents any alternative proofs of '-intersect(a·2·b·a·7·nil, 0·1·2···9·nil)' being tried.

USE OF '/' TO IMPLEMENT 'THNOT'

```
+thnot(X)-X-/-fail.
+thnot(X).
```

'thnot' is a meta-predicate, meaning "there is no instance of the statement X which is provable from the database".
e.g.

```
+onfloor(X)-thnot(on(X,Y)).
```

"If it is impossible to prove X is 'on' something, you may assume X is 'onfloor'."

```
+thexists(X)-thnot(thnot(X)).
```

"Some instance of statement X is provable from the database." Note that this clause will never result in instantiating X; the proof, if any, of X is thrown away by the backtracking induced by '-fail'.
e.g.

```
+supported(X)-thexists(on(X,Y)).
```

BOTTOM-UP (ANTECEDENT THEOREMS) IN PROLOG.

+→(dg,2).+&(dg,3).+·(dg,4).++(gd,5).+-(gd,5).

-assert(C&S)+assert(C).

-(-ant(V))

-assert(C&S)-/-assert(S).

-assert(X&C→Y)-/+assert(X→(C→Y)).

-assert(X→Y)-/-ajoutc(-X·+assert(Y)·nil)
-X+assert(Y).

-assert(X)-ajoutc(+X·nil)+X.

Example

Command	Prolog Clauses Added
+assert(	
(noun(X,Y)&verb(Y,Z)→sent(X,Z))&	-noun(X,Y)+assert(verb(Y,Z)→sent(X,Z)).
(verb(X,Y)&noun(Y,Z)→sent(X,Z))&	-verb(X,Y)+assert(noun(Y,Z)→sent(X,Z)).
(time(X,Y)→noun(X,Y))&	-time(X,Y)+assert(noun(X,Y)).
(flies(X,Y)→noun(X,Y))&	-flies(X,Y)+assert(noun(X,Y)).
(time(X,Y)→verb(X,Y))&	-time(X,Y)+assert(verb(X,Y)).
(flies(X,Y)→verb(X,Y))&	-flies(X,Y)+assert(verb(X,Y)).
time(0,1) &	+time(0,1).
	+noun(0,1).
	-verb(1,Z)+assert(sent(0,Z)).
	+verb(0,1).
	-noun(1,Z)+assert(sent(0,Z)).
flies(1,2)	+flies(1,2).
	+noun(1,2).
	-verb(2,Z)+assert(sent(1,Z)).
	+sent(0,2). "Time flies."
	+verb(1,2).
	-noun(2,Z)+assert(sent(2,Z)).
	+sent(0,2). "Time flies."
).	

EXAMPLE OF THE USE OF ANCESTOR RESOLUTION.

Non-clausal predicate logic program

①+② connected(X,Y) ← (canreach(X) → canreach(Y)).
③ canreach(Y) ← connected(X,Y) & canreach(X).

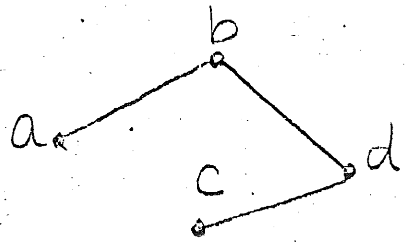
Corresponding clauses

① + connected(X,Y) − canreach(Y).
② + connected(X,Y) + canreach(X).
③ + canreach(Y) − connected(X,Y) − canreach(X).

Regard 'canreach' as a <u>local</u> procedure which is only meaningful in the environment of procedure 'connected'.

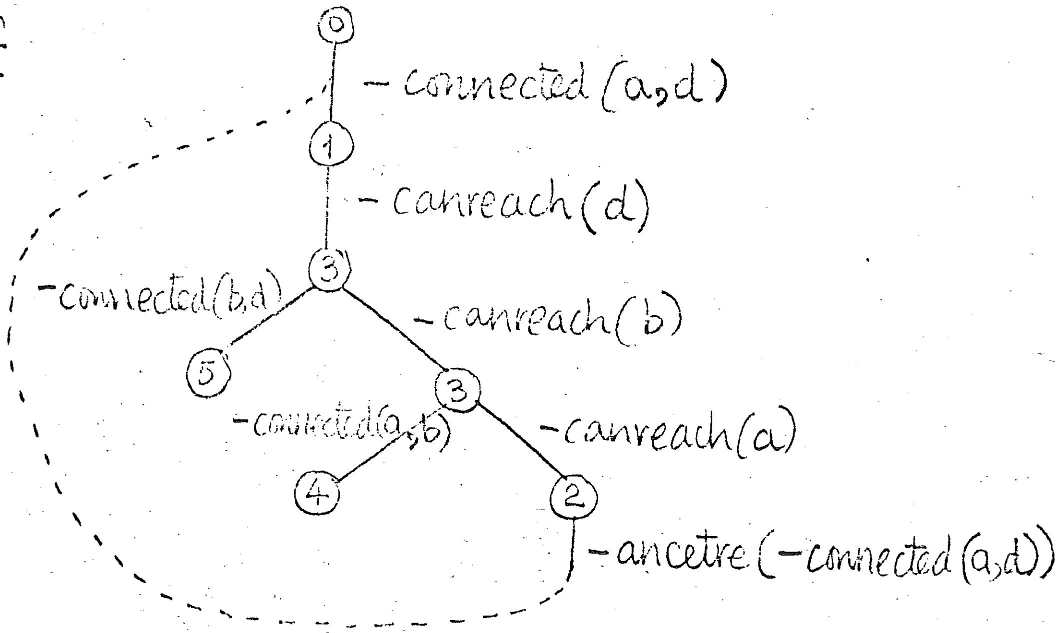
Corresponding Prolog

④ + connected(a,b).
⑤ + connected(b,d).
⑥ + connected(c,d).



① + connected(X,Y) − canreach(Y).
② + canreach(X) − ancetre(− connected(X,Y)).
③ + canreach(Y) − connected(X,Y) − canreach(X).

Typical Proof



THE META-PREDICATE UNIV

Example

```
+ univ( foo(a, X, fie(Y, b)),
        (f.o.o.nil) . a . X . fie(Y, b) . nil ).
```

A Prolog Program for 'Reverse'

```
+ rev(X.L, L1) - rev(L, L2) - app(L2, X.nil, L1).
+ rev(nil, nil).

+ app(X.L1, L2, X.L3) - app(L1, L2, L3).
+ app(nil, L, L).
```

The Same Program in a more readable form

```
+ let(

    :rev(X.L)       =   :app(:rev(L), X.nil)  &
    :rev(nil)       =   nil  &

    :app(X.L1, L2)  =   X. :app(L1, L2)  &
    :app(nil, L)    =   L

).
```

A Prolog Program to perform the translation

```
+&(dg,2).                ++(gd,8).
+=(dg,4).                +-(gd,8).
+.(dg,6).                +:(gd,9).
-let(P)+let1(P).
-let(P).

+let1(P&Q)+let1(P).
-let1(P&Q)-/+let1(Q).
-let1(:T0 = V0)
    -univ(T0, F.A0)
    -trans(V0, V, nil, C)
    -univ(T, (r.F).V.A0)
    -ajoutc(+T.C) - fail.

+trans(T0, T0, C0, C0) - isvar(T0)-/.
+trans(:T0, X, C0, C)-/
    -univ(T0, F.A0)
    -translist(A0, A, -T.C0, C)
    -univ(T, (r.F).V.A).
+trans(T0, T, C0, C)
    -univ(T0, F.A0)
    -translist(A0, A, C0, C)
    -univ(T, F.A).

+translist(T0.A0, T.A, C0, C)-translist(A0, A, C0, C1)-trans(T0, T, C1, C).
+translist(nil, nil, C0, C0).

+isvar(X)-notisvar(X)-/-fail.
+isvar(X).

+notisvar(999999)-/-fail.
+notisvar(X).
```

EXAMPLE TO ILLUSTRATE HOW PROLOG IS IMPLEMENTED

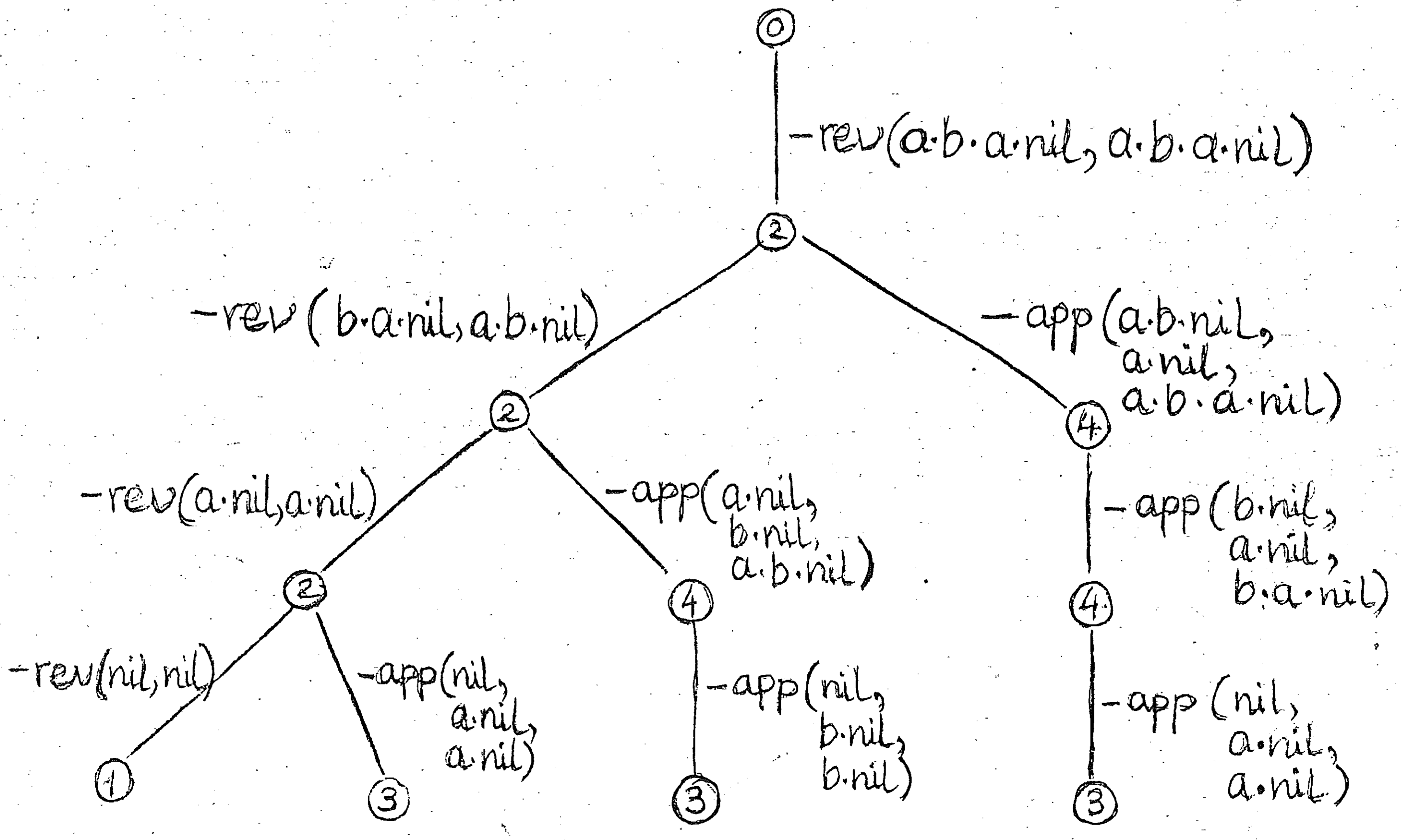
Program

① +rev(nil, nil).
② +rev(X·Y, Z) − rev(Y, W) − app(W, X·nil, Z).
③ +app(nil, X, X).
④ +app(X·Y, Z, X·W) − app(Y, Z, W).

Command

⓪ −rev(a·b·X, a·b·X)

First Proof Found



Snapshot 1 (and 3)

No.	Ancestor	Clause	X	Y	Z	W	Assignment
1	o	$-rev(a \cdot b \cdot X, a \cdot b \cdot X)$					
2		$+rev(X \cdot Y, Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	a	$b \cdot X_1$	$a \cdot b \cdot X_1$		
3		$+rev(X \cdot Y, Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	b	X_1	W_2		
4							

Seeking match for ⌜$-rev(X_1, W_3)$⌝ : <u>two</u> choices.

Snapshot 2

No.	Ancestor	Clause	X	Y	Z	W	Assignments
1	o	$-rev(a \cdot b \cdot X, a \cdot b \cdot X)$	nil				
2		$+rev(X \cdot Y \cdot Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	a	$b \cdot X_1$	$a \cdot b \cdot X_1$	$X_3 \cdot nil$	
3		$+rev(X \cdot Y \cdot Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	b	X_1	W_2	nil	
4		$+rev(nil, nil)$					X_1, W_3
5		$+app(nil, X, X)$	$X_3 \cdot nil$				W_2
6							

Seeking match for ⌜$-app(b \cdot nil, a \cdot nil, a \cdot b \cdot nil)$⌝ : fails.

Snapshot 4

No.	Ancestor	Clause	X	Y	Z	W	Assignment
1	o	$-rev(a \cdot b \cdot X, a \cdot b \cdot X)$	$X_4 \cdot Y_4$				
2	•	$+rev(X \cdot Y, Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	a	$b \cdot X_1$	$a \cdot b \cdot X_1$	$X_7 \cdot W_7$	
3	•	$+rev(X \cdot Y, Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	b	X_1	W_2	$X_4 \cdot nil$	
4	•	$+rev(X \cdot Y, Z) - rev(Y, W) - app(W, X \cdot nil, Z)$	a	nil	W_3	nil	X_1
5	•	$+rev(nil, nil)$					Y_4, W_4
6	•	$+app(nil, X, X)$	$X_4 \cdot nil$				W_3
7	•	$+app(X \cdot Y, Z, X \cdot W) - app(Y, Z, W)$	X_4	nil	$X_3 \cdot nil$	$X_3 \cdot nil$	W_2
8	•	$+app(nil, X, X)$	$X_3 \cdot nil$				W_7
9	•	$+app(X \cdot Y, Z, X \cdot W) - app(Y, Z, W)$	X_4	$X_3 \cdot nil$	$X_2 \cdot nil$	$b \cdot X_1$	X_4
10	•	$+app(X \cdot Y, Z, X \cdot W) - app(Y, Z, W)$	b	nil	$X_2 \cdot nil$	$X_4 \cdot Y_4$	
11	•	$+app(nil, X, X)$	$X_2 \cdot nil$				

2 words   1 word $(1+2n)$ words $(1+m)$ words

Total for this proof $= (5 \times 11) + (2 \times 28) + 7$

$= 55 + 56 + 7$

$= 118$ words