LECTURE II
LOGIC PROGRAMMING
PROLOG EXAMPLES

1. PROLOG responds to a PROGRAMMER QUERY

$$\exists x_1 \ldots x_m \; P_1(\;) \wedge P_2(\;) \wedge \ldots \wedge P_m(\;) \quad (\blacksquare)$$

where arities of p_i are n_i, & $x_1, \ldots, x_n$ are all free variables in all p_i.

2. PROLOG uses "program.pl" clauses (data clauses) to satisfy a goal ($\blacksquare$)

3. A goal can be satisfied immediately

4. If not then the first program clause is invoked which head (modulo substitution - with UA) matches the goal under consideration

5. If goal cannot be satisfied backtracking is initiated.

6. Backtracking reviews what was done attempting to re-satisfy the goals by finding an alternative way to

satify them.

7. If one is not content with an answer to the QUERY one can resume backtracking - hitting ";"

8. Matching clauses atoms - UA & substitution is used.

RECALL:

(i) isnumber (zero).
(ii) isnumber (S(X)) :- isnumber (X)

Query:

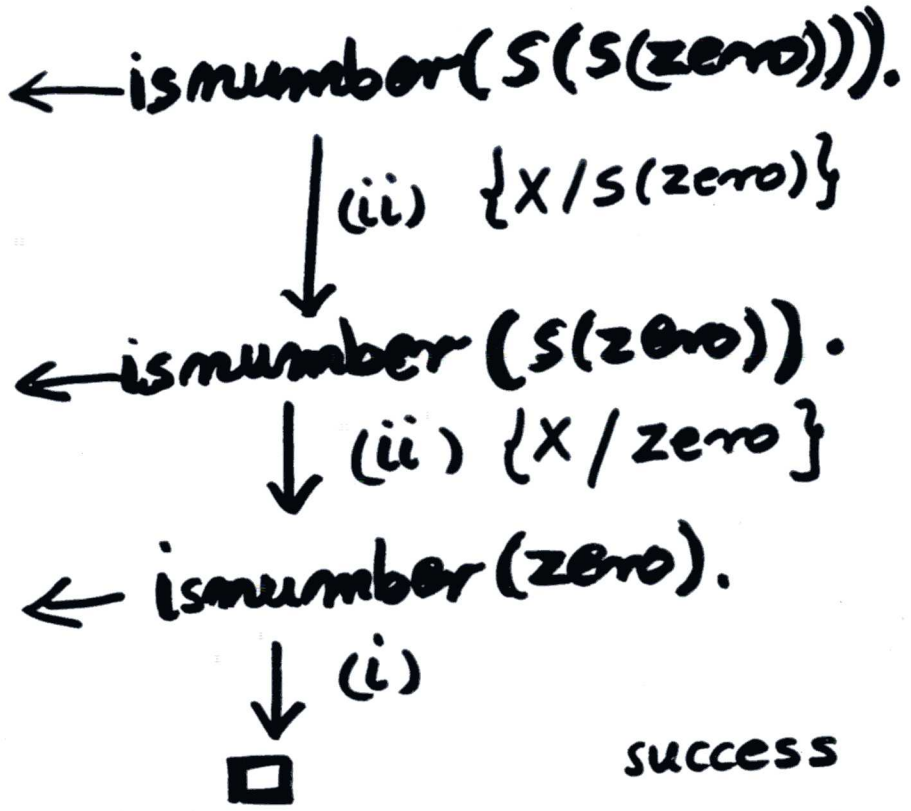
>> isnumber (zero):

yes

>> isnumber (S(S(zero))). (*)

yes.

LECTURE 11
LOGIC PROGRAMMING



SLD-tree for Δ & QUERY (*) collapses to single branch (a trunk).

$Q = \exists x \, \text{isnumber}(x)$

Again

>> isnumber(X). (**)

success 1

X = zero ;

X = s(zero); success 2

X = s(s(zero)) (RTN) success 3

yes

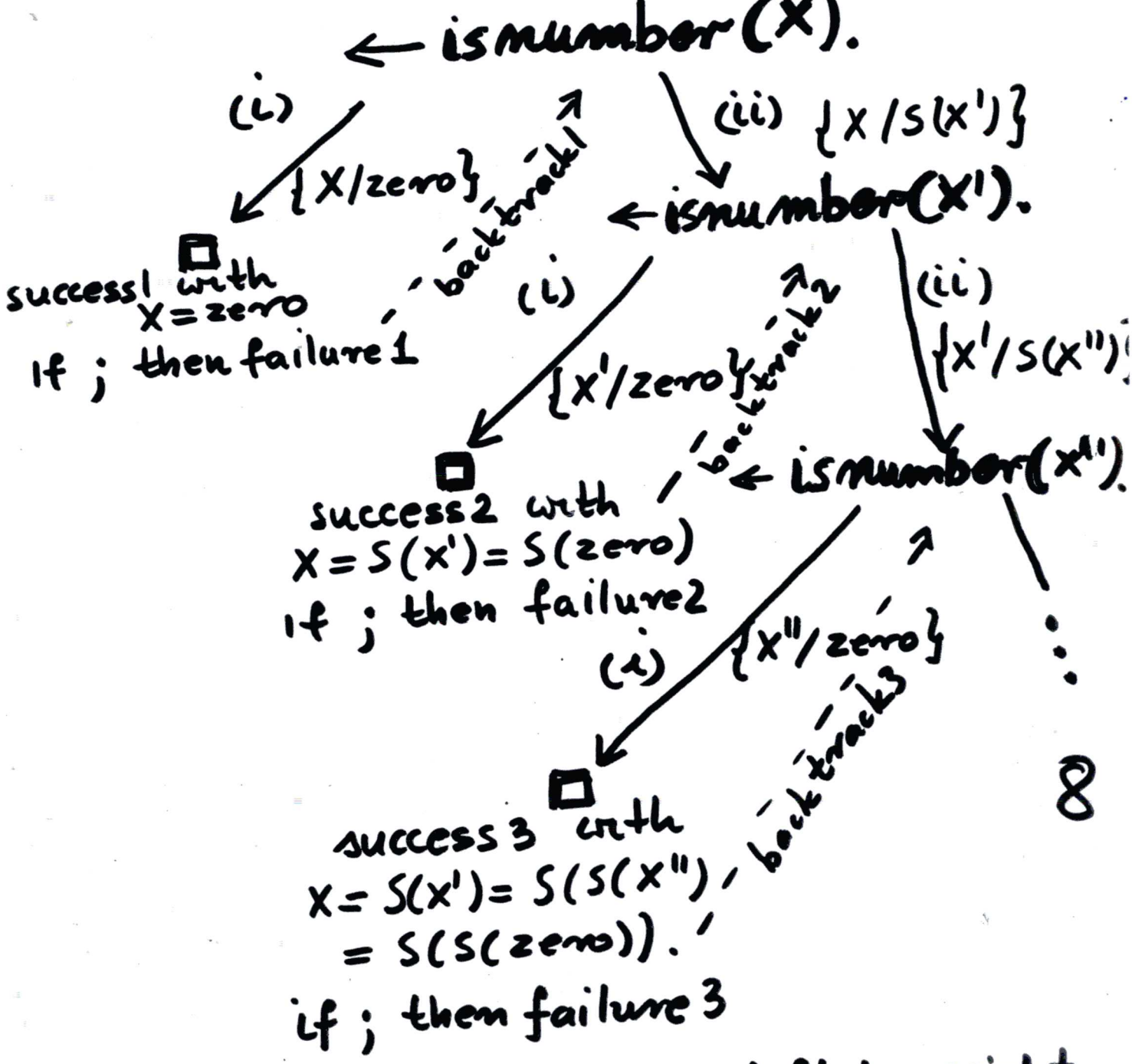
Once we find 1st success ";" converts this to failure and prompts PROLOG to resume searching SLD-tree

LECTURE 11

LOGIC PROGRAMMING

success! becomes failure 1 & PROLOG back-tracks to the first moment, where there is a choice.

SLD-tree for ▲ & (**)



Note: SLD tree grows from left to right.
The branch on right-hand side is ∞.

LECTURE II.
LOGIC PROGRAMMING

If we now change the order of clause in Δ to $\tilde{\Delta}$:

(i) isnumber(S(X)):- isnumber(X).
(ii) isnumber(zero).

Of course $\Delta \equiv \tilde{\Delta}$.

```
>> isnumber(zero).
   yes
>> isnumber(s(zero)).
   yes
```

The same responses - as expected since $\Delta \equiv \tilde{\Delta}$.

But

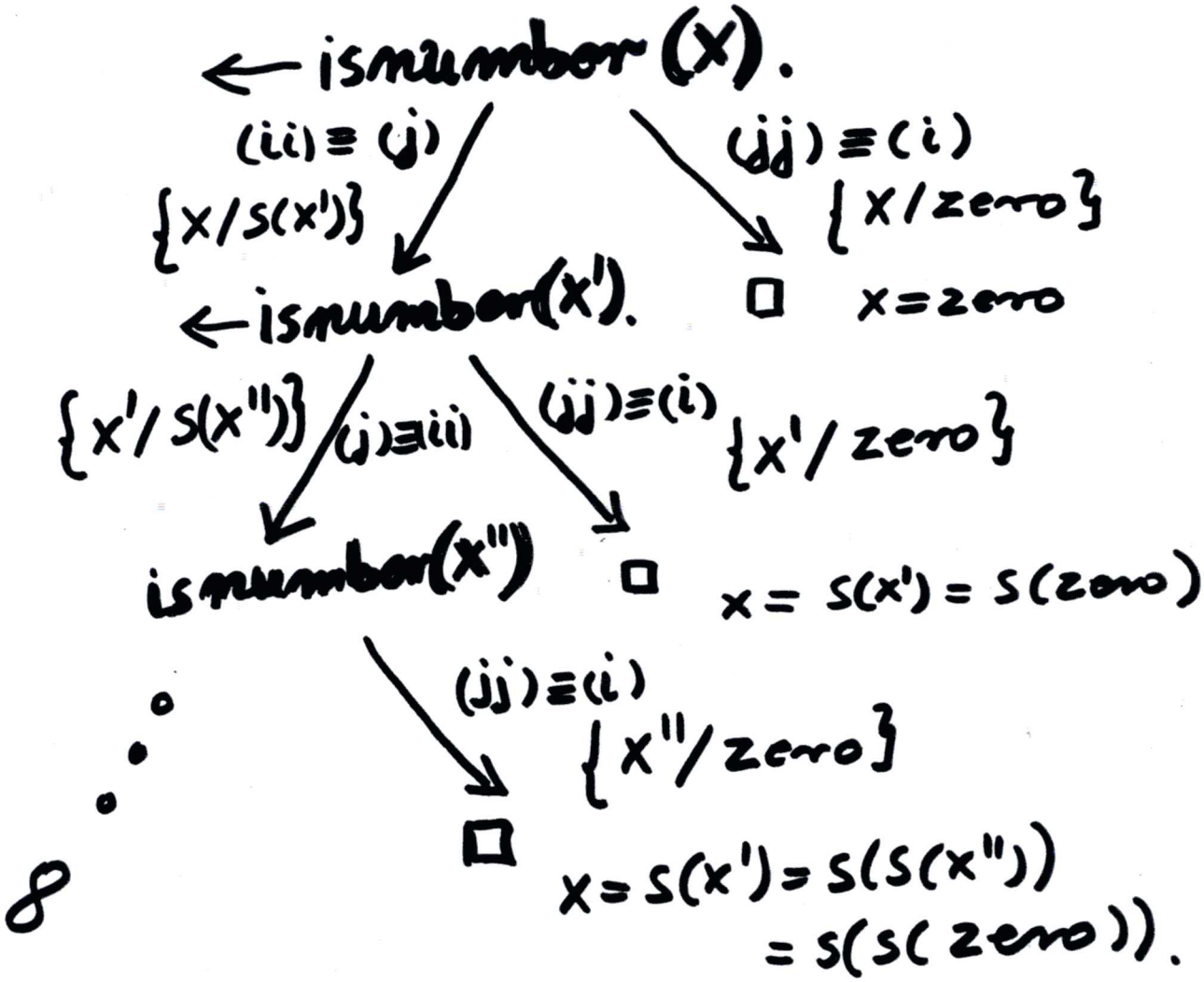
```
>> isnumber(X).          (*)
   Out of local stack
   [execution aborted]
```

Order of boundary conditions matters!

LECTURE 11

LOGIC PROGRAMMING

SLD tree for $\tilde{\Delta}$ & (*)



The infinite branch is flipped to the left-hand siole for program $\tilde{\Delta}$.

Since PROLOG applies <u>depth-first search</u>

from left to night we first get entangled in ∞-branch before reaching success leaf.

LECTURE 11
LOGIC PROGRAMMING

Boundary conditions should be put first.

Example 1:

(i) plus(zero, X, X) :- isnumber(X).
(ii) plus(s(X), Y, s(Z)) :- plus(X, Y, Z).

" 0 + X = X "

" if X + Y = Z ⇒ (X+1) + Y = Z+1 "

>> plus(s(zero), s(zero), s(s(zero))). (a)

>> plus(s(zero), s(zero), X). Yes (b)

" 1 + 1 = ? "

X = s(s(zero))

>> plus(s(zero), X, s(s(zero))).

" 1 + x = 2 " ?

X = s(zero).

>> plus(X, Y, s(s(zero))).

" X + Y = 2 "

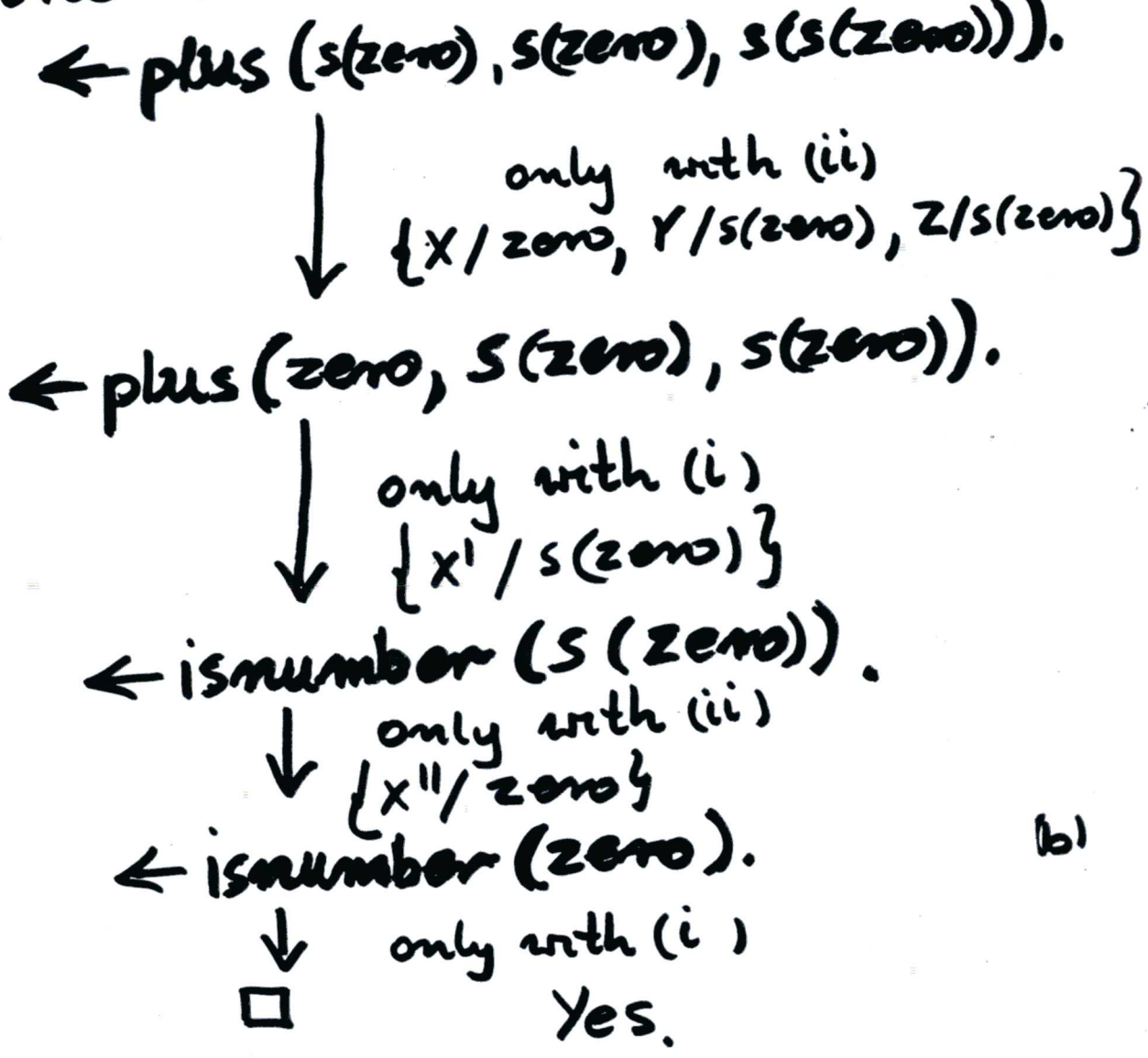
X = zero
Y = s(s(zero));

X = s(zero)
Y = s(zero);

X = s(s(zero))
Y = zero;
No.

LECTURE 11.
LOGIC PROGRAMMING

SLD tree for ▲ & a)



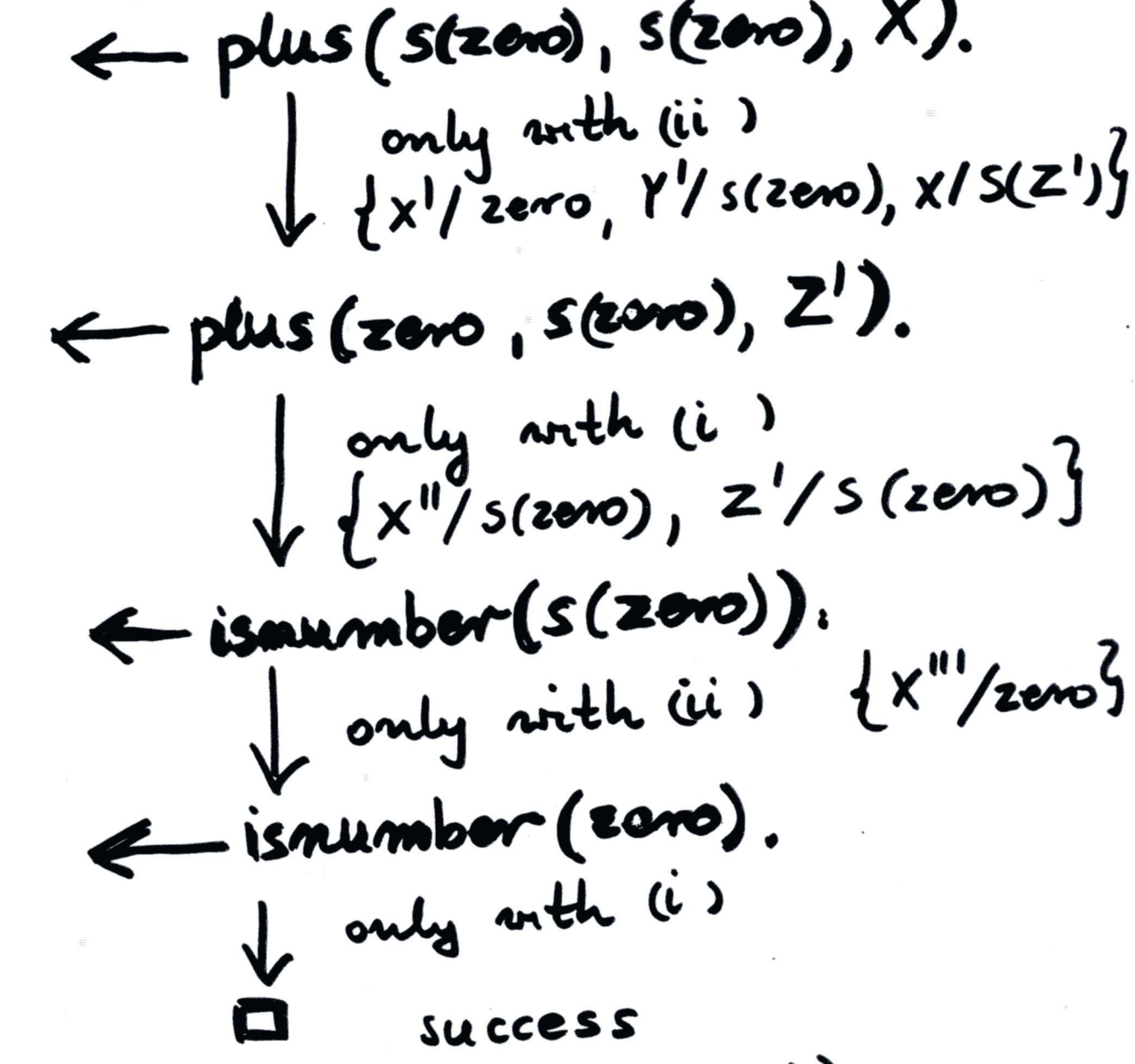
b)

Note that PROLOG does not report the substitution for which success took place as a) does not involve a free variable.

SLD-tree here is also a single trunk.

LOGIC PROGRAMMING
LECTURE II

SLD tree for ▲ & b)



$X \to S(Z') \to S(S(zero))$

$X = s(s(zero))$.

If you prompt here PROLOG with ; for further search (as SLD tree yields a unique here branch) the answer is <u>No.</u>

LECTURE 11
LOGIC PROGRAMMING

It is characteristic for <u>logic programming</u> that there are many equivalent solutions.

But we must be careful in PROLOG:

- order of clauses matters
- order of atoms in program clause matters

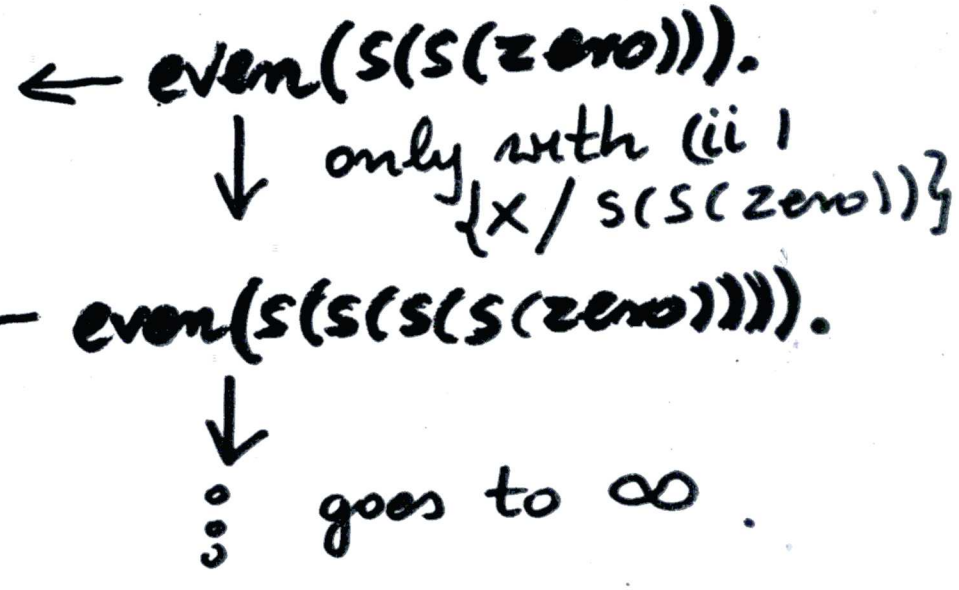
```
even(zero).                                   (i)
even(X) :- even(s(s(X))).                     (ii)
```

" zero is even"
" if X is even then X-2 even"

```
>> even(s(s(zero))).
```

"Is 2 even?"

out of local stack!



LOGIC PROGRAMMING
LECTURE 11

Note that source of

"out of local stack"

[execution aborted]

is

— either repeating current goal.

— or compounding composition of function: ie. f(f(.....

Another warning:

(i) badplus(zero, X, X) :- isnumber(X).

(ii) badplus(X, S(Y), S(Z)) :- badplus(X, Y, Z).

" 0+X = X"

" if X+Y = Z then X+(Y+1) = Z+1"

(i) & (ii) look fine but we cannot add

s(zero) + zero !

So one pair of correct adding is lost!

LECTURE 11
LOGIC PROGRAMMING

There is in-built arithmetics in PROLOG:

The predicate:

N is X ≡ is(N, X).
↑ predicate

assign the value (a numeric one) to variable X.

There are other arithmetic operators:

"+", "-", "*", "/" and " mod ".

So >> X is 3+2.

X=5.

>> X is X+3.

Error in arithmetic expression

no

LECTURE 11
LOGIC PROGRAMMING

```
>> X is 17 * 23.
   X = 391
```

Also non-logical way of comparison operators:

X > Y	X =:= Y	is equal
x < Y	X =\= Y	$x \neq Y$
x >= Y		
x =< Y		

```
>> X=3.
   X=3
```

Symbolic computation introduced before

- mainly to show recursive nature of logic programming
- arithmetic is done to do practical computations in PROLOG

LECTURE 11
LOGIC PROGRAMMING

We define a predicate which allows transition from symbolic arithmetic to a real one:

e.g.

testing

multiply(X,Y,Z).

for X = s(s(s(s(s(zero)))))
|||
"5"

Y = s(s(s(s(s(s(s(zero)))))))
|||
"7"

If PROLOG gives symbolic correct result

Z = s(..... s(zero)....)
||| (35 times)
"35"

LECTURE 11
LOGIC PROGRAMMING

one has to count number of composed functor s.

AWKWARD !

So we can define a predicate:

(i) show(zero, 0).

(ii) show(S(X), N) :- show(X, M),
N is M+1.

Now

>> show(X,5), show(Y,7),
multiply(X,Y,Z), show(Z,N).

1. first numeric 5 is assigned to symbolic "5" i.e.:

X = S(S(S(S(S(zero)))))

2. then numeric 7 is assigned to symbolic "7" i.e.:

   Y= s(s(s(s(s(s(s(zero)))))))

3. We pass both X & Y with symbolic values to predicate multiply which returns Z symbolicly expressed as "35"

   Z = s(s.... (zero)...)
   (s(s.... : 35-times)

4. Then Z is passed to show & this predicate retranslate Z (symbolicly) into N as 35.

LECTURE 11
LOGIC PROGRAMMING

a)

```
>> show(X,2).
   X = s(s(zero)).   ← correct answer
```

" ; " ← if PROLOG is prompted for further searching

⇓

" out of local stack " !!! incorrect answer

b)

```
>> show(s(s(zero)),N).
        N = 2.
```

" ; "

No ← correct answer

So most of the predicates in PROLOG may not operate symmetricly!