TUTORIAL 4

CSCI3230 (2013-2014 First Term)

By Paco WONG (pkwong@cse.cuhk.edu.hk)

Outline

- Introduction
- Basic Concepts
- Queries
- Examples
- Prolog Environment

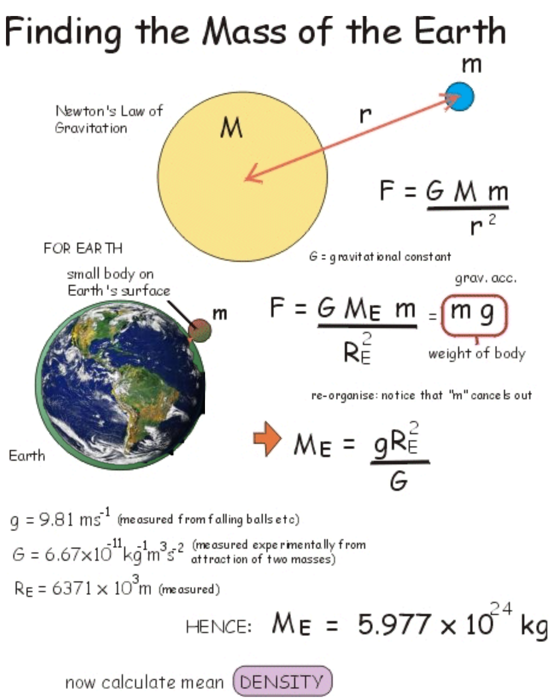
PROgramming LOGic

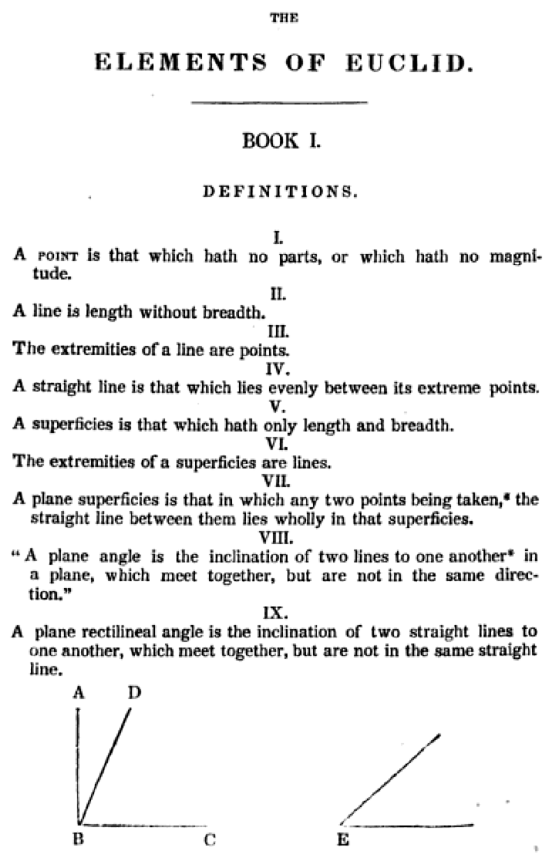
- Old
  - One of the first logic programming language
  - John Alan Robinson contributes to the foundations of automated theorem proving and logic programming in 1965.
  - The first Prolog system was developed in 1972 by Colmerauer with Philippe Roussel.
- Declarative semantics
- Uses the first-order predicate calculus

BASIC CONCEPTS

How will you model the world?

Facts and Rules





THE

ELEMENTS OF EUCLID.

BOOK I.

DEFINITIONS.

I.

A point is that which hath no parts, or which hath no magnitude.

II.

A line is length without breadth.

III.

The extremities of a line are points.

IV.

A straight line is that which lies evenly between its extreme points.

V.

A superficies is that which hath only length and breadth.

VI.

The extremities of a superficies are lines.

VII.

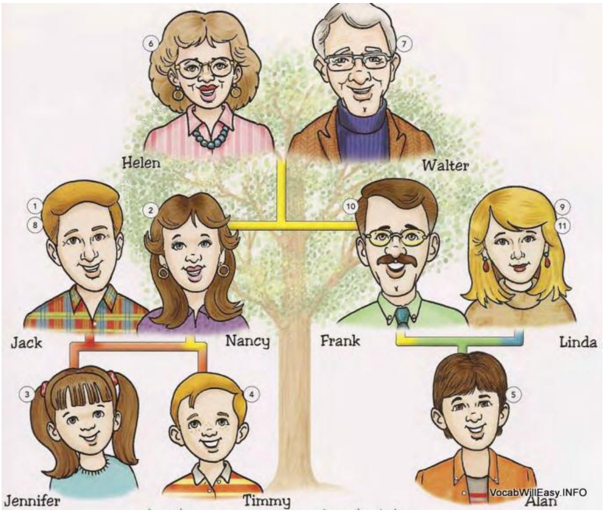
A plane superficies is that in which any two points being taken,* the straight line between them lies wholly in that superficies.

VIII.

"A plane angle is the inclination of two lines to one another* in a plane, which meet together, but are not in the same direction."

IX.

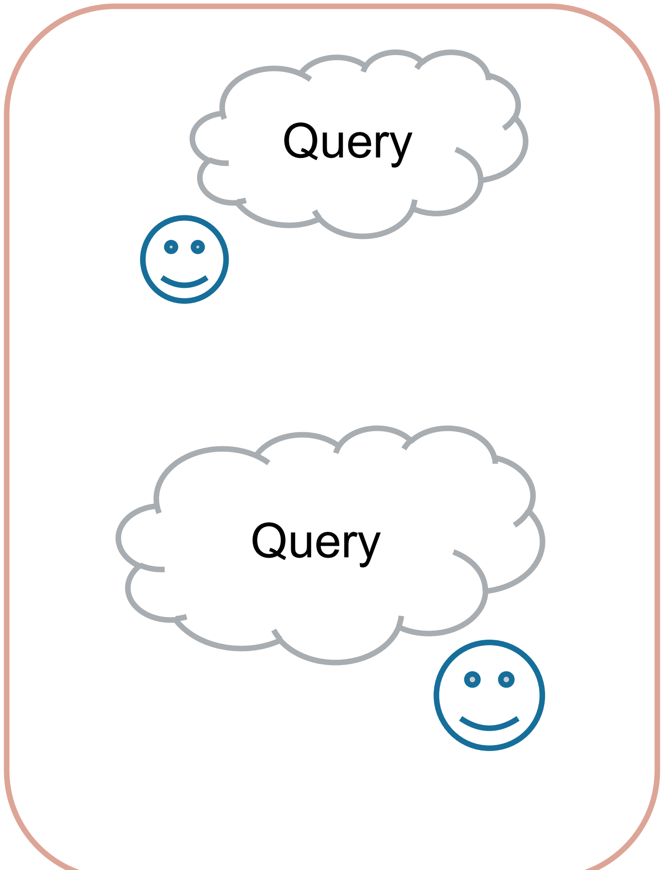
A plane rectilineal angle is the inclination of two straight lines to one another, which meet together, but are not in the same straight line.



http://www.see.leeds.ac.uk/structure/dynamicearth/internal/grav.gif
http://books.google.com.hk/books?id=5lN1sy51SwYC&printsec=frontcover&dq=elements+euclid&hl=zh-TW&sa=X&ei=1qNKUt7JF-mViQflkoCQAw&ved=0CDEQ6AEwAA#v=onepage&q&f=false
http://lh4.ggpht.com/_jR9d0NRexI4/TMKiy4yO4LI/AAAAAAAAB9Y/Bm24FKmtFt0/s800/family%20members-1.jpg

PROLOG

A world governed by facts and rules `End with a dot .`



Facts and rules are stored in a database (.pl file).

Example 1

```
thinking(i).                    %Fact
alive(X):-thinking(X).          %Rule
```

```
?- alive(i). %query
true. %fact
```

Ask your question in query mode

Terms: Data Objects

Non-variable			Variable
Atomic		**Compound**	X Csci3230 Dept _fruit (Person,Food)
Atom	**Number**	f(f(a),f(X)) [1, 2, 3, 4] [[eric, kate], [[peter, mary]]	
csci3230 dept cuhk_cse []	100		

Compound Term (a.k.a. Structure)

$$f(t_1, t_2, \ldots, t_n)$$

- f : functor
- T_i : terms
- Arity : number of sub-terms

Example 1

```
likes(fruit(lemon,who(tom,alex))).%Fact
likes(fruit(apple,who(ben,fred))).%Fact
```

```
?- likes(fruit(apple,who(ben,fred))).
true.
```

Compound Term: List

$$f(t_1,t_2,\ldots,t_n)$$

- f : functor
- T_i : terms
- Arity : number of sub-terms

Example 2

```
.(a,.(b,.(c,[]))).        %Fact, this creates a list.
```

```
?- [a|[b,c]].
true. %fact, different representation
?- [a,b,c].
true. %fact, different representation
```

Statements

- **FACTS** states a predicate holds between terms.

Example 3

```
father(harry,james).     %Fact 1
mother(harry,lily).      %Fact 2
```

```
?- father(harry,james).
true.
```

Statements

- **RULES** defines the relationship about objects.

r(...) :- conditions for r(...) be true.

Meaning	Predicate Calculus	PROLOG
And	$\wedge$	,
Or	$\vee$	;
If	$\leftarrow$	:-
Not	$\neg$	not

Rules

Example 4.1

```
father(harry,james).       %Fact 1
mother(harry,lily).        %Fact 2
parent(Child,Person) :-
      father(Child,Person);mother(Child,Person). %Rule 1
```

```
?- parent(harry,albus).
false.
?- parent(harry,james).
true.
?- parent(harry,lily).
true.
```

Rules

Example 4.2

```
father(harry,james).      %Fact 1
mother(harry,lily).       %Fact 2
parent(Child,Person) :- father(Child,Person). %Rule 1
parent(Child,Person) :- mother(Child,Person). %Rule 2
```

```
?- parent(harry,albus).
false.
?- parent(harry,james).
true.
?- parent(harry,lily).
true.
```

Universal Facts

- Uses _ the anonymous variable

Example 5

```
likes(_,pizza). %Everyone likes pizza
```

```
?- likes(james,pizza).
true.
?- likes(daisy,pizza).
true.
```

Arithmetic

- No arithmetic is carried out until commanded by *is* predicate
- Operators: +, -, *, /

Example 6

```
plus(X,Y,R):- R is X+Y.
```

```
?- plus(3,4,R).
R = 7.
```

What if and explain.

```
plus(X,Y,R):- R = X+Y.
```

QUERIES

Asking questions about the facts and rules

Queries

- Retrieves the information from a logic program
- Asks whether a certain relation holds between terms
- Patterns in the same logic syntax as the database entries
- Pattern-directed search
  - The proof will be logically followed.
- Searching the database in left to right depth-first order to find out whether the query is the logical consequence of the database of specifications

Meaning	Predicate Calculus	PROLOG
And	$\wedge$	,
Or	$\vee$	;
Not	$\neg$	not

Flow of Satisfaction (Simplified)

Pattern matching from the first rule to the final rule

1. Fact/Rule
2. Fact/Rule
3. Fact/Rule
4. Fact/Rule
5. Fact/Rule
6. …

```
If it is a fact
    Return true
Else if it is a rule
    For each conditions c
        IsProvableTruth(c)
        If the rule must return false
            Return false
        Else if the rule must return true
            Return true
```

?- MyQuery → IsProvableTruth → True./False.

Simplified

There are some ways to control the flow of satisfaction, e.g. !.

Moreover, if the condition contains variables, these variables will be instantiated.

'Execution' of Queries

- Can be regarded as
  - Selective Linear Definite (SLD) Resolution
  - Depth-First Search of AND-OR tree
- Two main parts
  - **Unification**
    - Match two predicates or terms
    - Consistently instantiates the variables,
      - e.g. p :- f(A,**B**),g(**B**,C). %Both variables B always have the same value.
  - **Backtracking**
    - When some predicate "fails", try alterative matching

Unification

- Try to match two predicates or terms by suitably instantiating variables
- Rules of Unification

First Term	Second term	Condition
Uninstantiated variable X	Any term	The term does not contain X
Atom or Number	Atom or Number	They are equal
Compound Term	Compound Term	Same functors, same arity, and the corresponding terms unify

Unification Examples

1st term	2nd term	Unified?	Variable instantiation
abc	xyz	No	
X	Y	Yes	X→Y
Z	123	Yes	Z→123
f(A)	f(234)	Yes	A→234
f(A)	f(1,B)	No	
f(g(A),A)	f(B,peter)	Yes	A→peter, B→g(peter)
t(L,t(X,b))	t(t(c,d),t([],b))	Yes	L→t(c,d), X→[]
[H|T]	[a,b,c,d]	Yes	H→a, T→[b,c,d]

[a,b,c,d] is the same as [a|[b,c,d]]

Unification Examples

1st term	2nd term	Unified?	Variable instantiation
tree(a,nil)	xyz	No	
add(U,V)	add(5,a)	Yes	U→5, V→a
exp(_,N)	exp(x,add(5,b))	Yes	N→add(5,b), _ ignored
sub(_,_)	sub(5,3)	Yes	_ need NOT be consistent
exp(sin(A),2)	exp(sin(x),1)	No	
[a,X,c]	[a,b,c]	Yes	X→b
[a,sin(X)|Y]	[a,sin(6),c]	Yes	X→6, Y→[c]
[X|_]	[]	No	

Backtracking

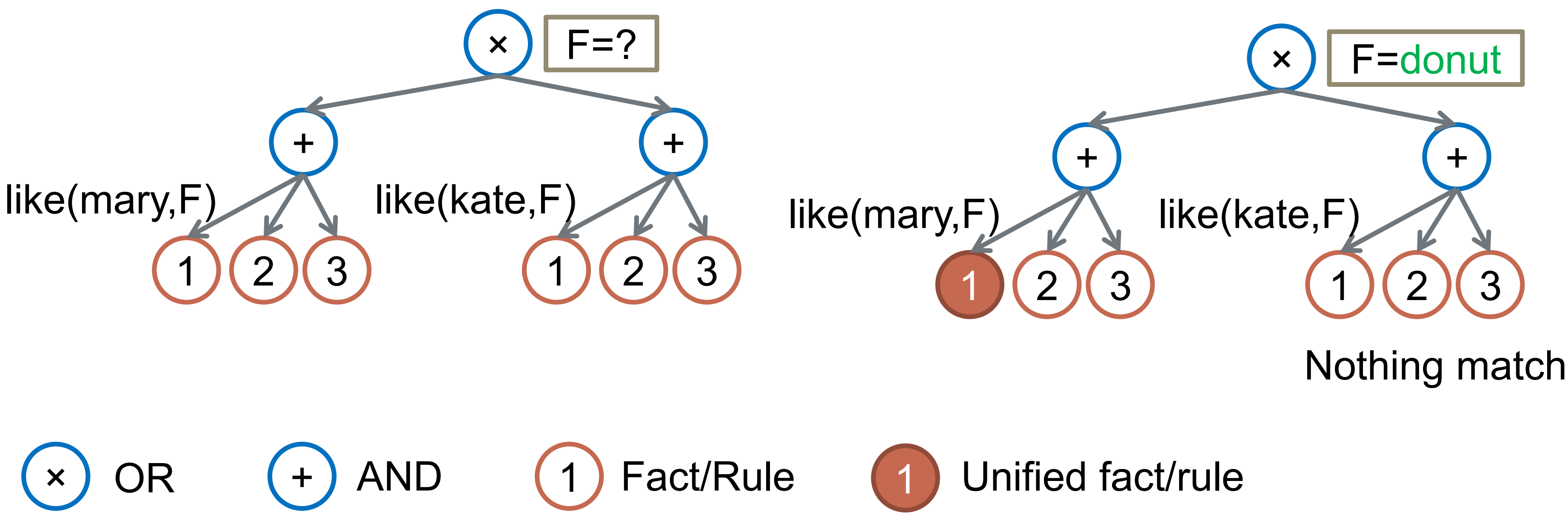
- When asking $P_1(..),P_2(..),\ldots,P_n(..).$
  - If anyone fails (due to instantiation), say P_i, Prolog backtracks, and try an alternative of P_{i-1}
- After a successful query,
  - If user presses ‘`;`’, backtrack and try alternatives.

Backtracking Example

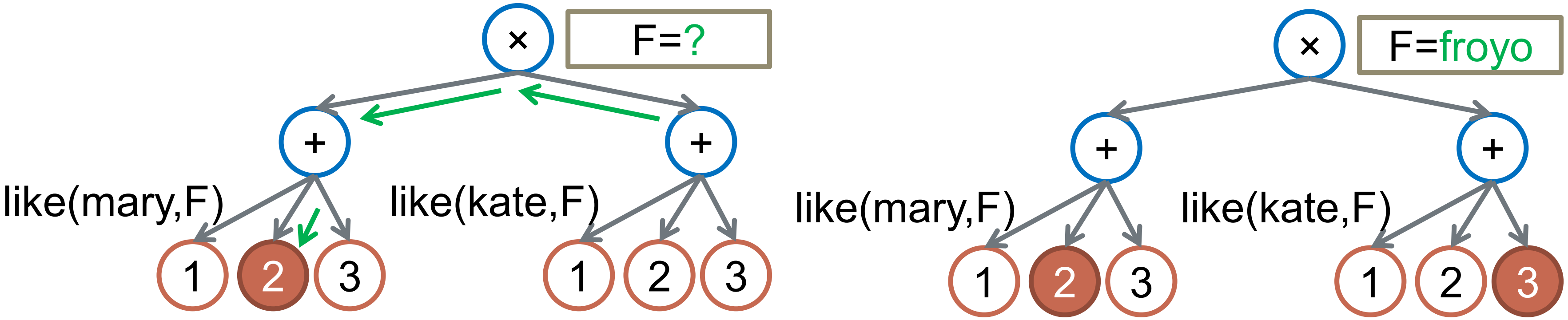
Example 6

```
likes(mary,donut). %Fact 1
likes(mary,froyo). %Fact 2
likes(kate,froyo). %Fact 3
```

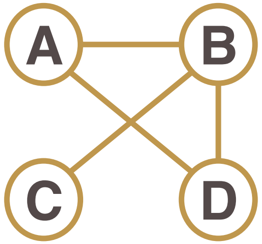
```
?- likes(mary,F),likes(kate,F). %Sth both Mary and Kate like
F = froyo.
```



Backtracking Example (cont.)



EXAMPLES



Satisfying Goals

1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).

<u>Explanation</u>

i. <u>**1.**</u> → Return true → Press .

ii. 1. → 2.→ <u>**3.**</u>→ Return true → Press .

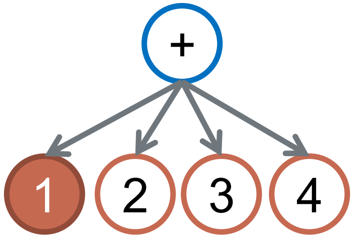
?- link(a,b).

true. /* See i */

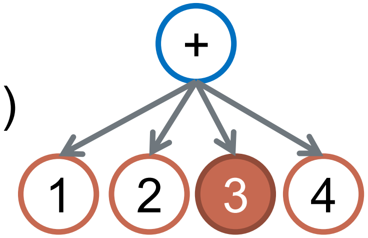
?- link(a,d).

true. /* See ii */

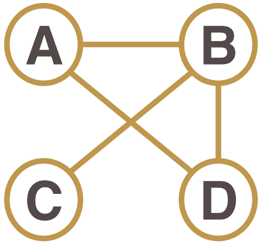
link(a,b)



link(a,d)



Satisfying Goals



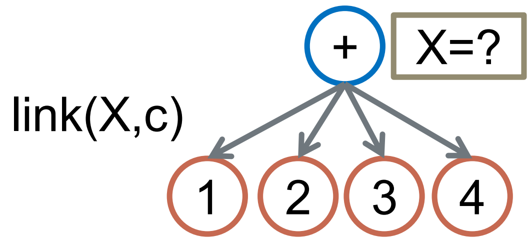
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).

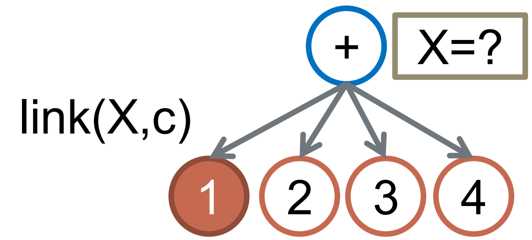
?- link(X,c).

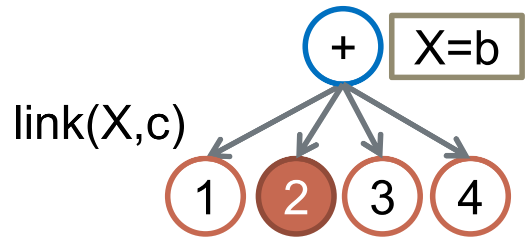
X = b.

Explanation

1. → **2.** → Instantiate X to b → Return true → Press .



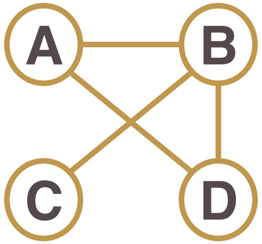




Match and return true.

Press . Done.

Using ; for more



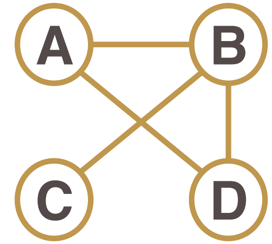
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).

```
?- link(a,X).
X = b; /* press ; */
X = d.
```

Pressing ';' asks Prolog to find more answers.
Pressing 'enter' will end the query

Explanation
1. → Instantiate X to b → Return true → Press ;
→ 2. → **3.** → Instantiate X to d → Return true → Press .

Using ; for more

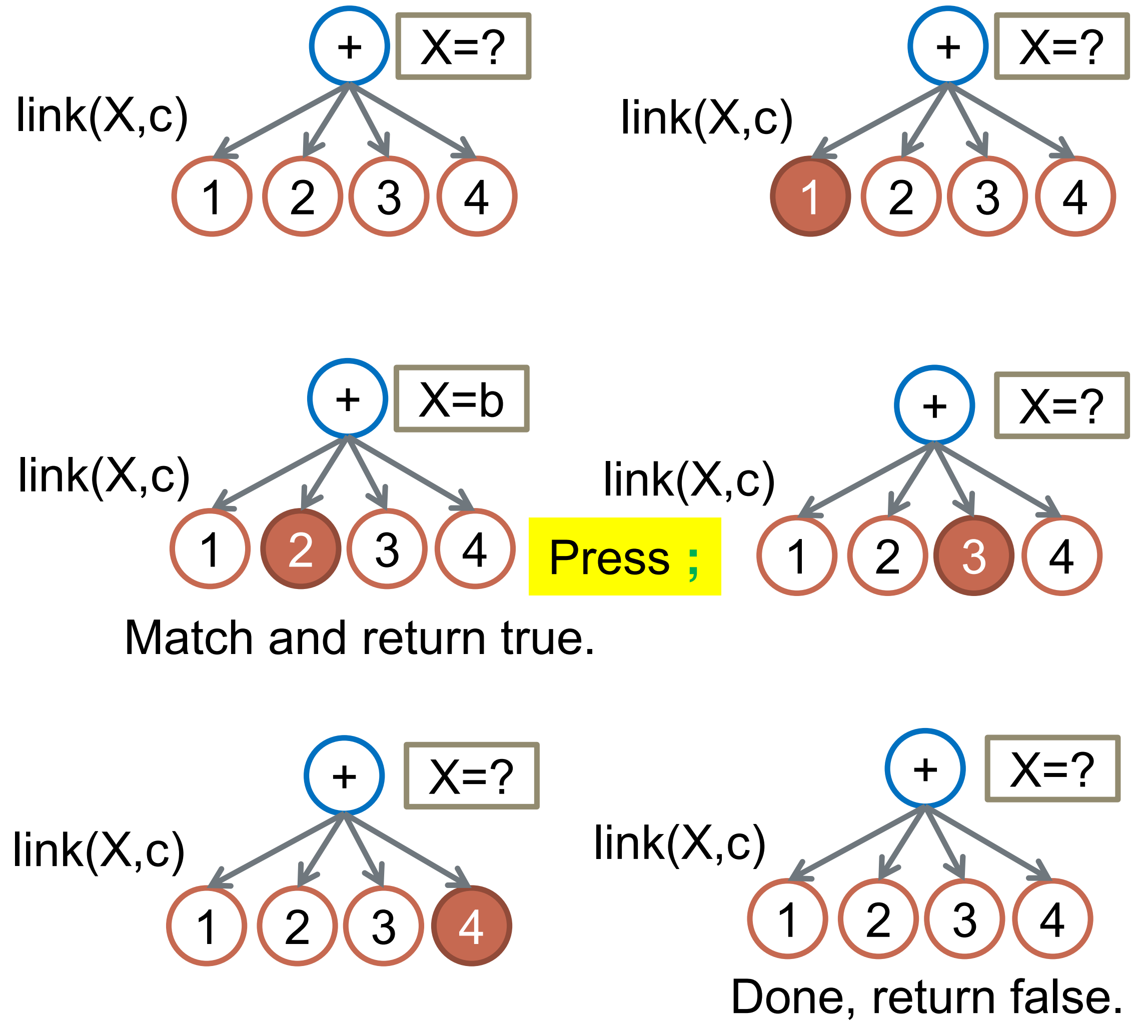


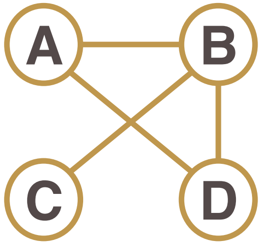
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).

```
?- link(X,c).
X = b;
false.
```

Explanation

1. → **2.** → Instantiate X to b → Return true → Press ; → 3.→ 4. → Return false





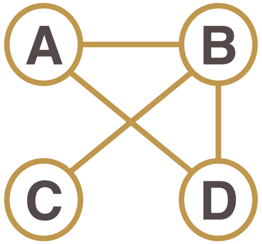
False != Can’t be true

1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).

<u>Explanation</u>
1. → 2.→ 3.→ 4.→ Return false

?- link(a,c).
false.

If Prolog answers “no”, it doesn’t mean that answer is definitely false. It means that the system cannot deduce that it is true given its database – **Closed World Assumption**



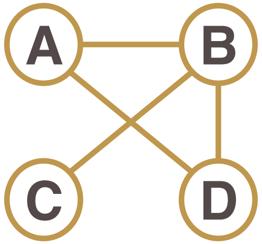
Queries - Example

1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

?- link(a,c).

true.

1. → 2.→ 3.→ 4.→ **5.**
   → X = a, Y = c
   → Match link(a,Z)
     → **1.** → Z = b → Return true
   → *Result* = true
   → Match link(b,c)
     → 1. → **2.** → Return true
   → *Result* = *Result* and true = true
   → Return *Result*



Queries - Example

1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

?- link(a,K).

K = b ;

K = d ;

K = c ;

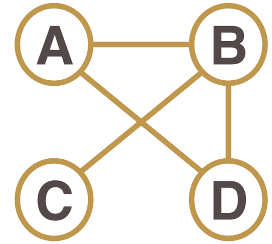
K = d ;

ERROR: Out of local stack

Skip the previous parts

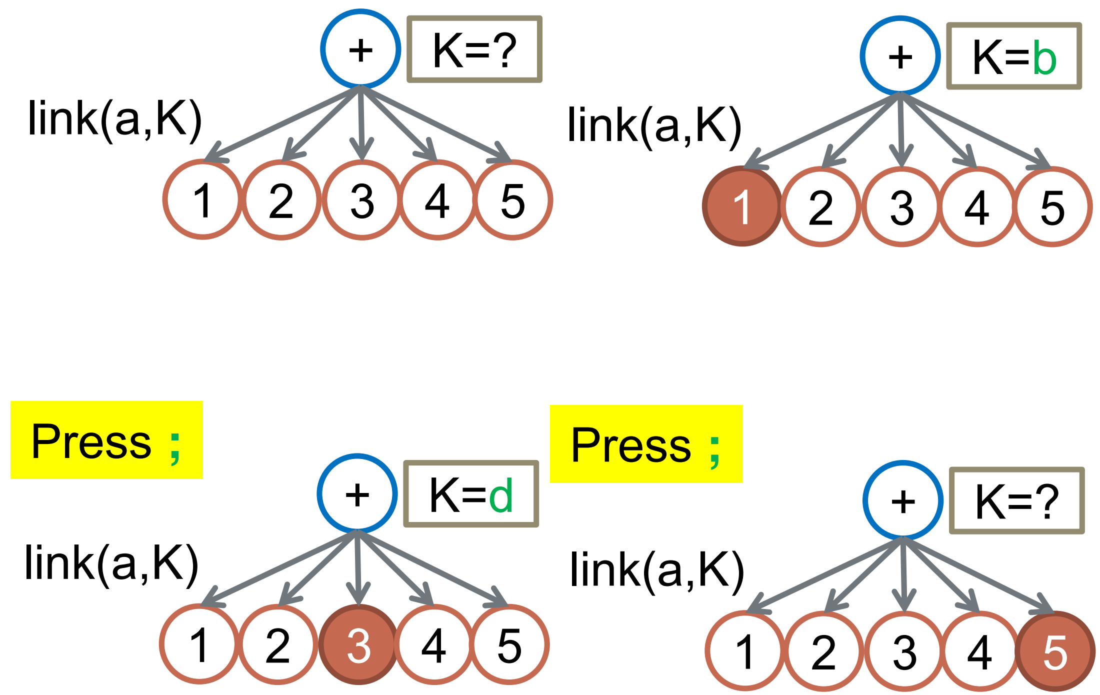
→ **5.**
- → X = a, Y = K
- → Match link(a,Z)
  - → 1. → Z = b → Return true
- → *Result* = true
- → Match link(b,Y)
  - → 1. → 2. → Y=c → Return true
- → *Result* = *Result* and true = true
- → Return *Result*
- → Press ;
- → Undo the last assignment of *Result*
- → Continue the matching link(b,Y)
  - → 3. → 4. → Return true
- → *Result* = *Result* and true = true
- → Return *Result*
- → Press ;

Queries - Example

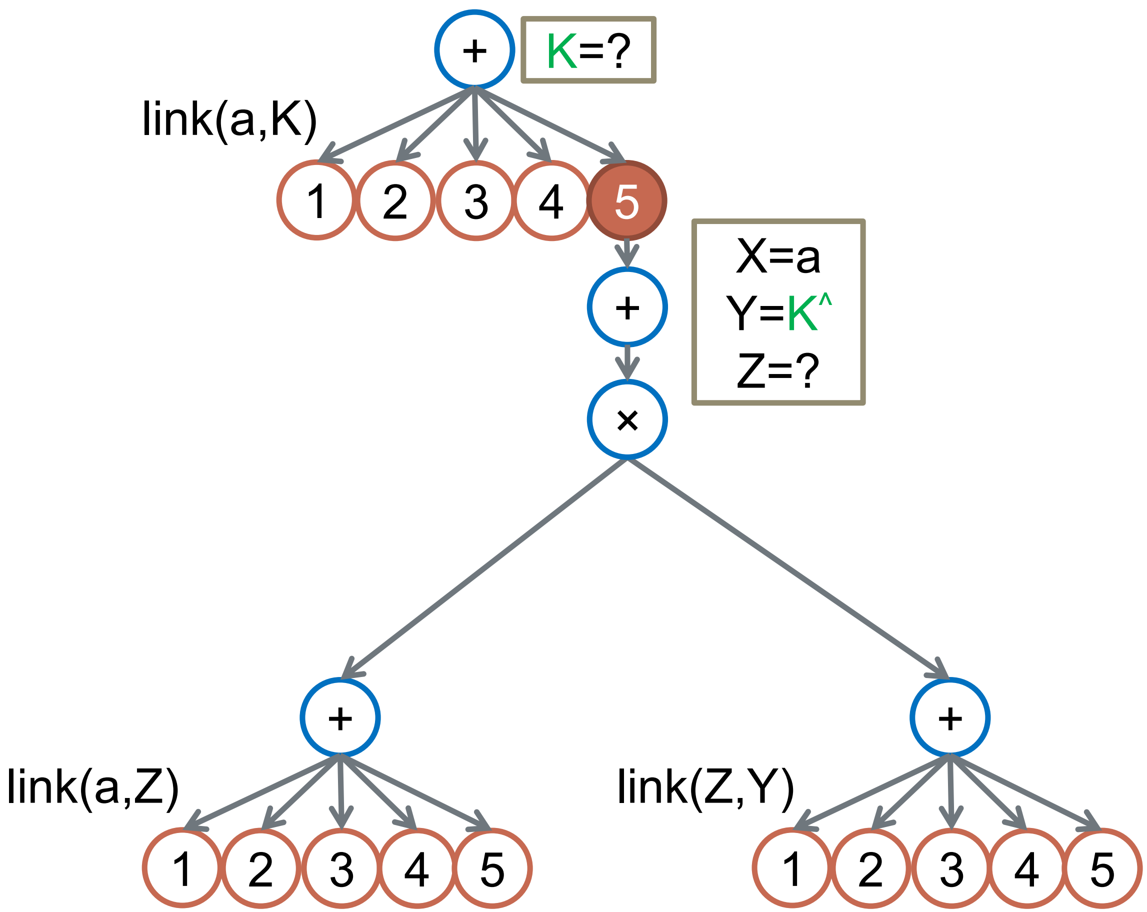


1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

?- link(a,K).
K = b ;
K = d ;
K = c ;
K = d ;
ERROR: Out of local stack

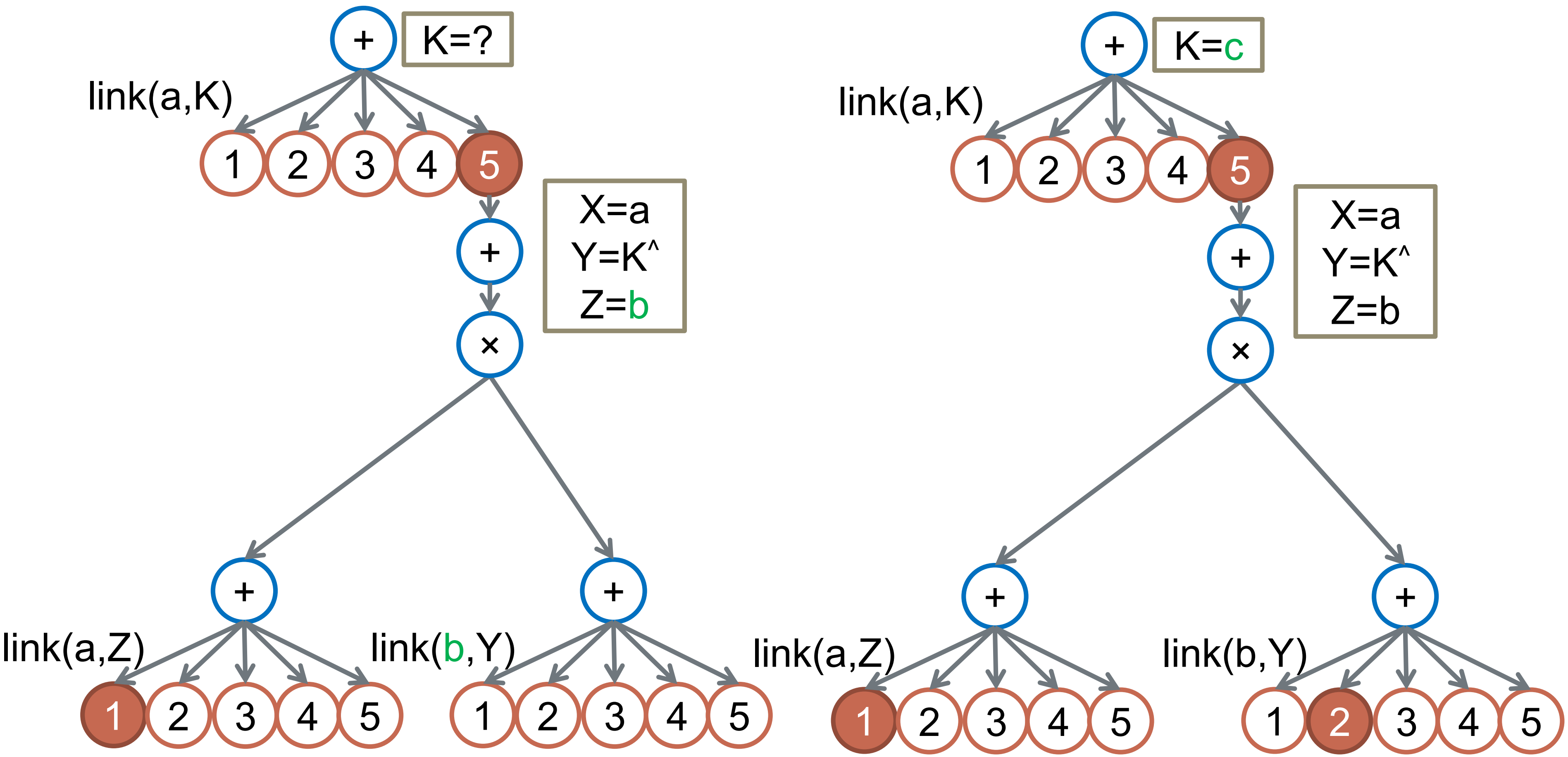


Node Expansion



```
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).
```

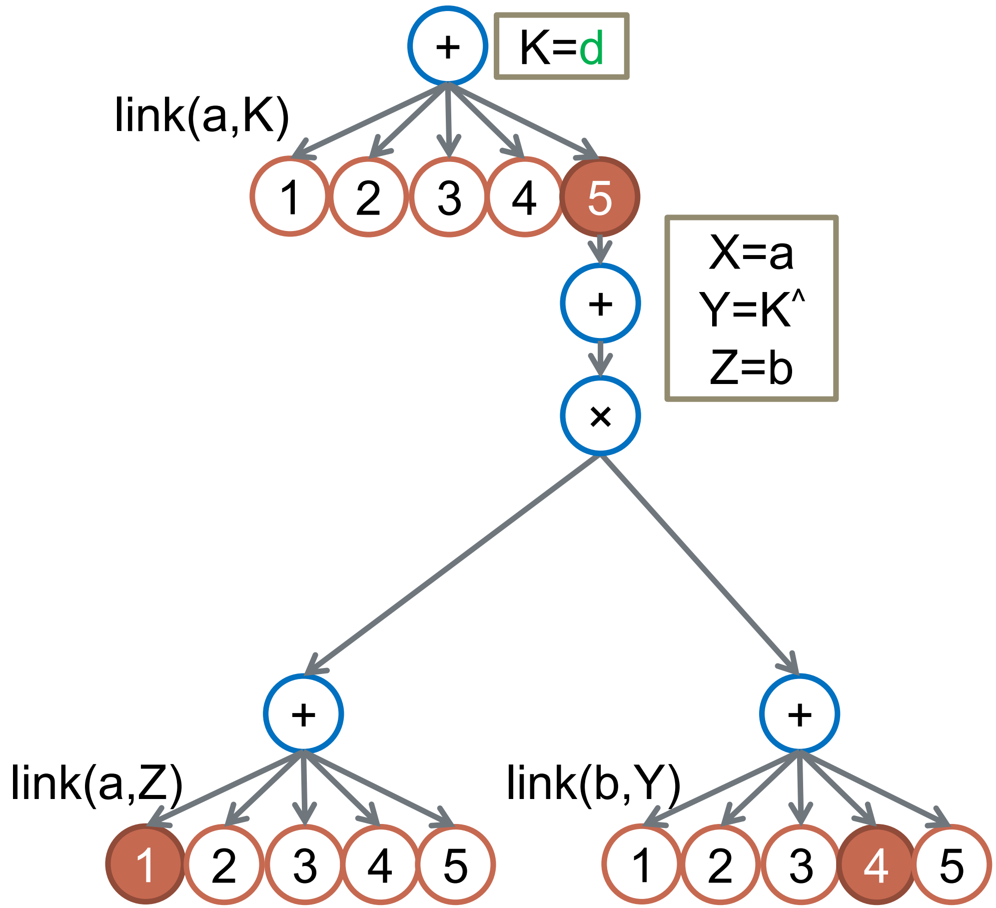
```
K = c ;
K = d ;
ERROR: Out of local stack
```



1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

K = c ;
K = d ;
ERROR: Out of local stack

Press ;



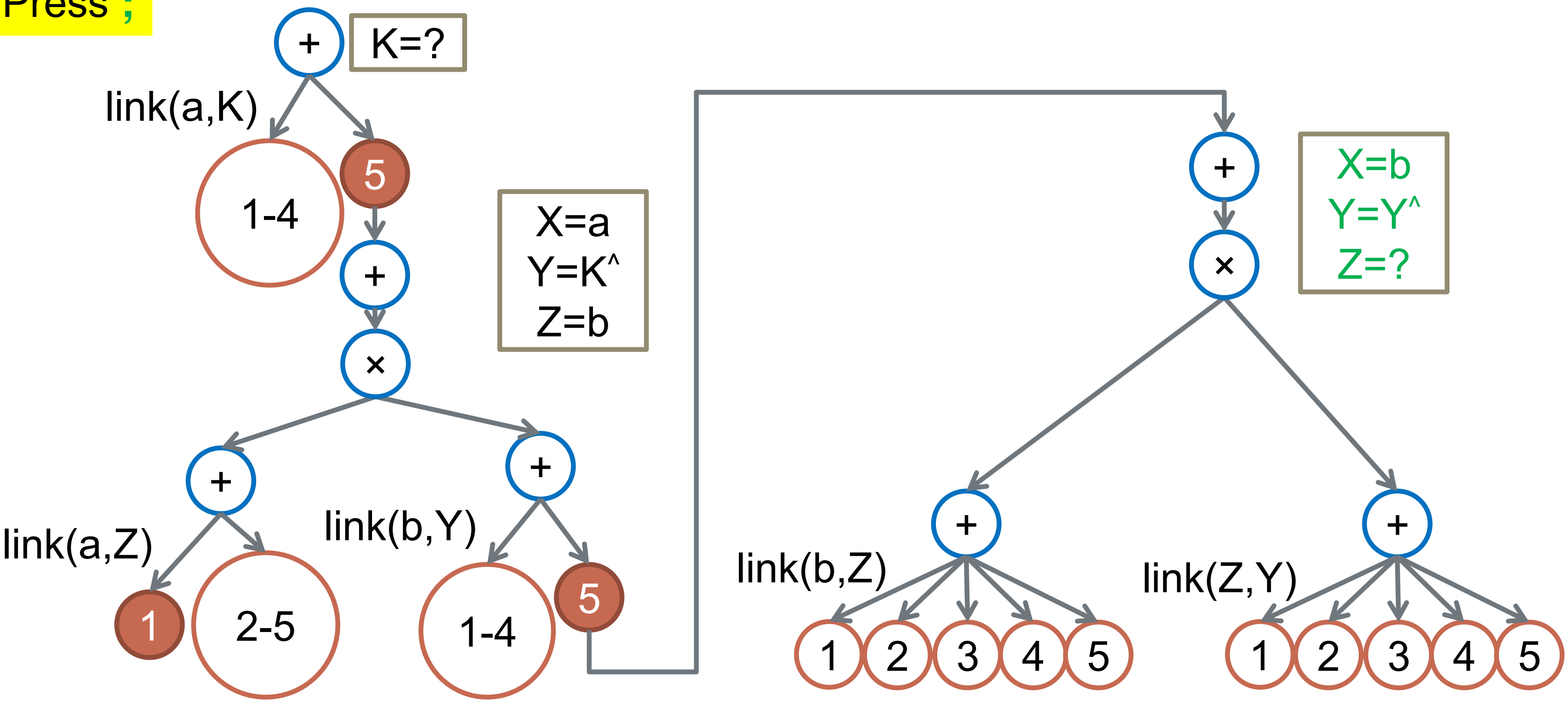
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

K = c ;

K = d ;

ERROR: Out of local stack

Press ;



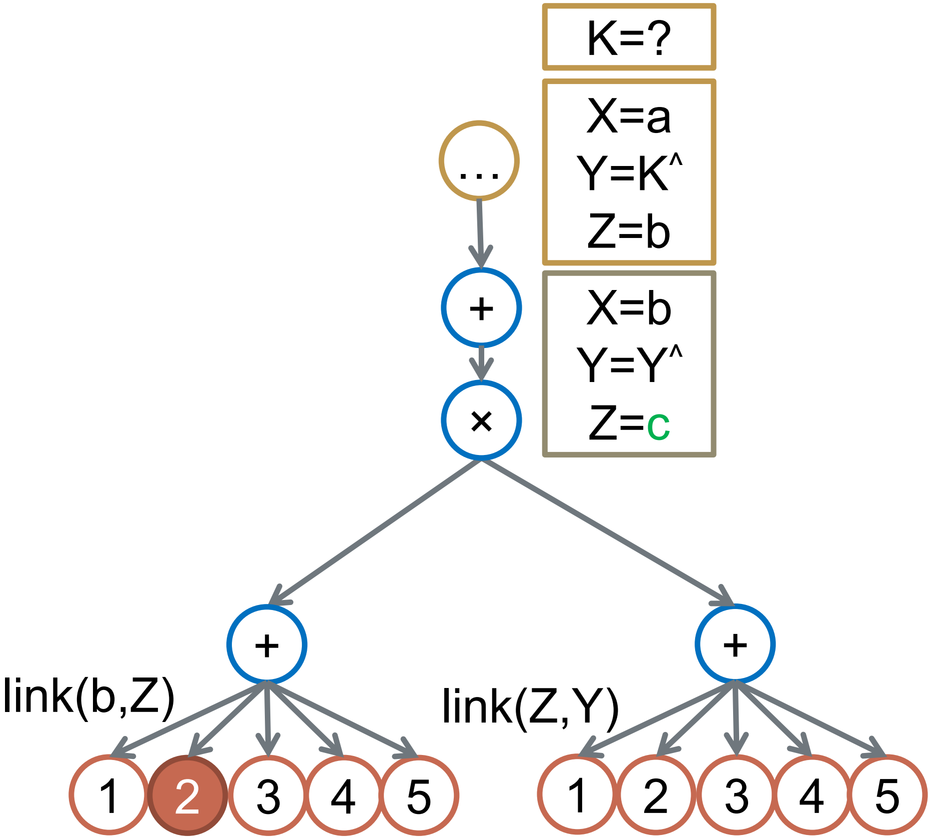
Expand to prove link(b,Y) is true

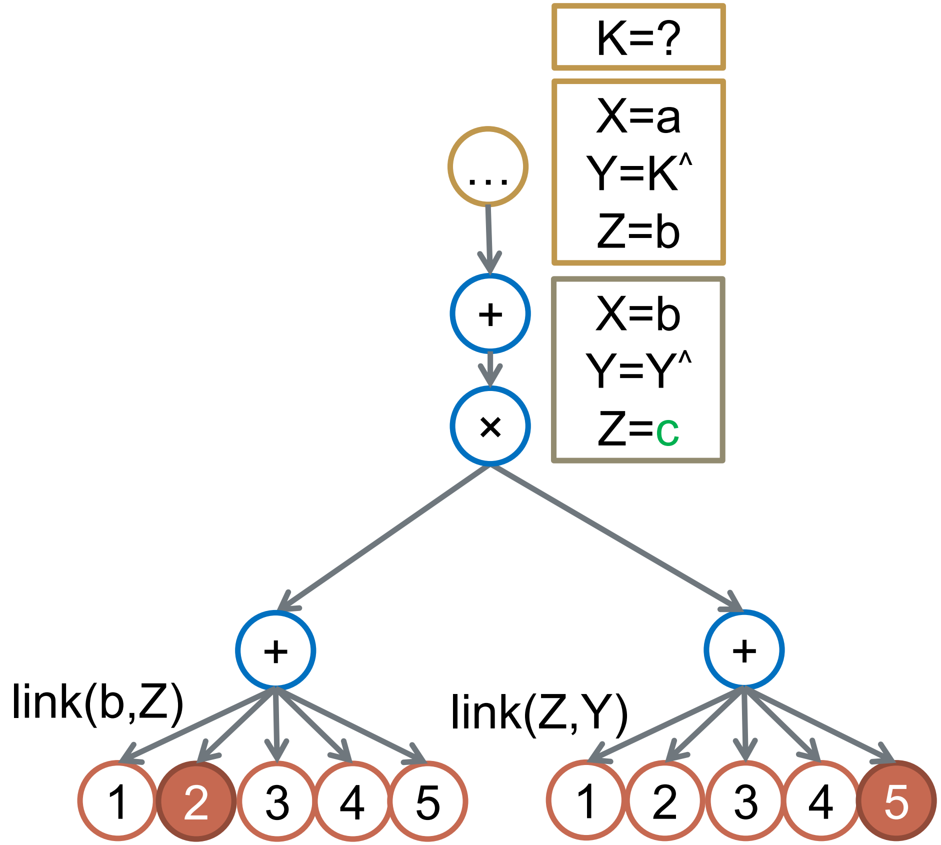
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

K = c ;

K = d ;

ERROR: Out of local stack





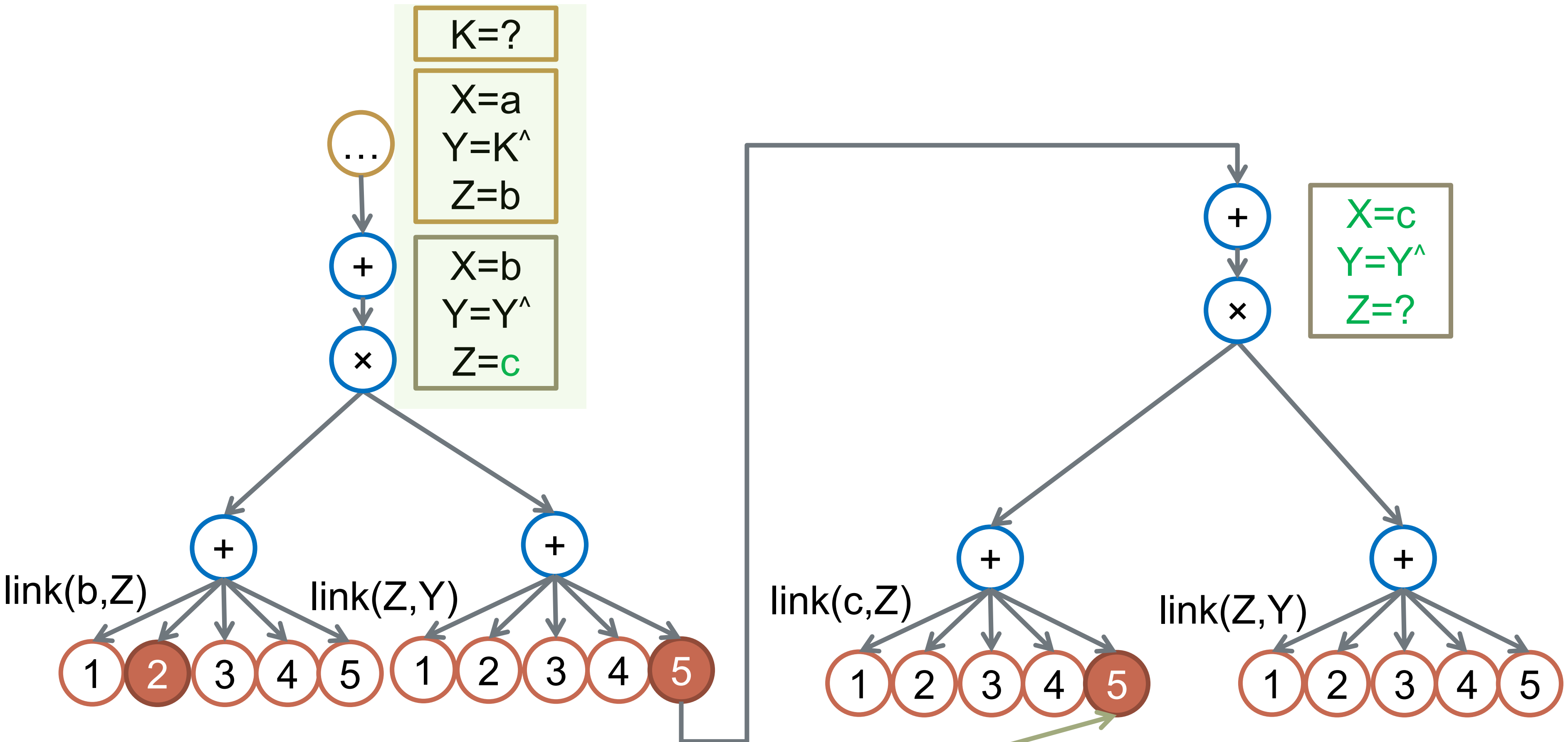
Expand to prove link(c,Y) is true

1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

K = c ;

K = d ;

ERROR: Out of local stack

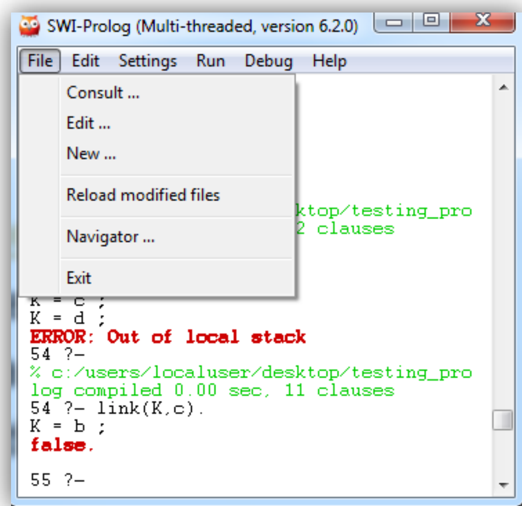


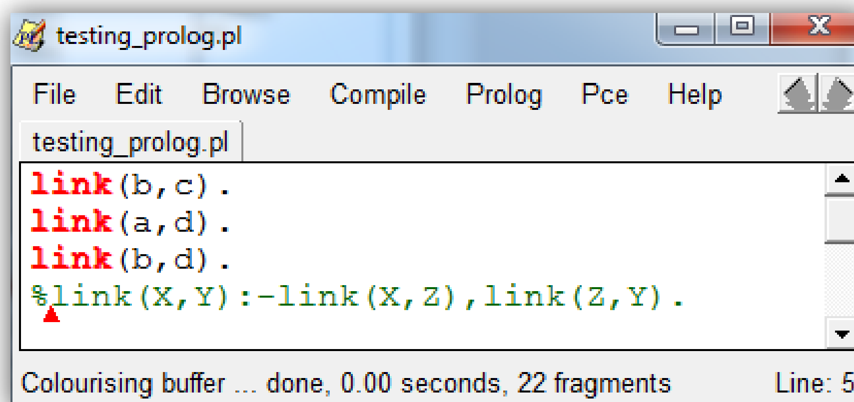
Expanding rule 5 to prove link(c,Z) is true which will repeat this step again!

?

SWI-Prolog (used in our testing system)

- Download from http://www.swi-prolog.org/
- Consult: Load the database
- New: Create a database (a text file)
- Edit: Modify a database with the editor
- Reload modified files: Re-consult the database to update the facts and rules





Summary

- Terms
- Statements
  - Facts and Rules
- Queries
  - Flow of satisfaction
  - Unification and Backtracking
- Examples
- Prolog Environment

Feedback: email or http://goo.gl/5VLYA

Announcement

1. Postpone the deadline of Lisp Assignment to 7th Oct.
2. Written Assignment 1
3. Does anyone need to apply for a CSE Unix account?
   - Please send a email to hyszeto@cse.cuhk.edu.hk
     - SID
     - Name
     - Major

Appendix

- The Closed World Assumption http://www.dtic.upf.edu/~rramirez/PL2/PrologIntro.pdf
- Horn Clause and SLD resolution
- http://en.wikipedia.org/wiki/Horn_clause
- http://www.cis.upenn.edu/~cis510/tcl/chap9.pdf

The Closed World Assumption

In Prolog, `Yes` means a statement is *provably true*. Consequently, `No` means a statement is *not provably true*. This only means that such a statement is *false*, if we assume that all relevant information is present in the respective Prolog program.

For the semantics of Prolog programs we usually do make this assumption. It is called the *Closed World Assumption*: we assume that nothing outside the world described by a particular Prolog program exists (is true).

Reference

- http://ktiml.mff.cuni.cz/~bartak/prolog/data_struct.html
- http://www.dtic.upf.edu/~rramirez/PL2/PrologIntro.pdf