TUTORIAL 4
PROLOG

CSCI3230 (2019-2020 First Term)

By Ran WANG (rwang@cse.cuhk.edu.hk)

Outline

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

INTRODUCTION

PROgramming LOGic

- Old
  - One of the first logic programming languages
  - 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
  - The program logic is expressed in terms of relations, represented as facts and rules.
- Based on the idea of theorem proof
  - Facts
  - Rules
  - Proof of queries

PROgramming **LOG**ic

- Prolog (under Logic Programming Paradigm)
  - Telling what is true
  - Asking the computer to try and draw conclusions
- Well-suited for tasks that benefit from rule-based logical queries
- Applications
  - Theorem proving
  - Expert system
  - Term rewriting
  - Type systems
  - Automated planning
  - Natural language processing
  - …

PROLOG – How does it work?

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
```

Database

Query

End with a dot .

Ask your question in query mode

The answer to the query will be inferred using the facts and rules in the database.

BASIC CONCEPTS

Terms and Statements

Basic concepts

- Terms – data objects
  - Non-variable: atom, number, compound
  - Variable
- Statements
  - Fact
  - Rule

Terms - Data Objects

<table>
<tr><th colspan="3">Non-variable</th><th>Variable</th></tr>
<tr><td colspan="2">Atomic</td><td>Compound</td><td rowspan="3">X
Csci3230
Dept
_fruit
(Person,Food)</td></tr>
<tr><td>Atom</td><td>Number</td><td rowspan="2">f(f(a),f(X))
[1, 2, 3, 4]
[eric, kate], [[peter, mary]]</td></tr>
<tr><td>csci3230
dept
cuhk_cse
[]</td><td>100</td></tr>
</table>

Terms - Compound Terms

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

- 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.
```

Terms - Compound Terms

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

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

- List

$[\ t_1,t_2,\ldots,t_n\ ]$

$.(t_1,\ .(t_2,\ldots,\ .(t_n,[\ ]\ )\ )\ )$

$[t_1 \mid \boxed{[\ t_2,\ldots,t_n\ ]}\ ]$

Head Tail

$[\ t_2 \mid \underline{[\ t_3,\ldots,t_n]}\ ]$

$[\ t_3 \mid \underline{[\ t_4,\ldots,t_n]}\ ]$

…

Example 2

Some systems do not support this notation.

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

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

Statements - Facts

- A **FACT** states a predicate that holds between terms.

*(Clauses with empty bodies are called **facts**.)*

Example 3

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

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

Statements - Universal Facts

- Using _ for the anonymous variables

Example 5

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

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

Statements - Rules

- A **RULE** defines the relationship among objects.

*(Clauses with bodies are called **rules**.)*

$$r1(t_1,t_2,\ldots,t_n) \text{ :- } con_1(t_1,t_2,\ldots,t_n) \text{ , } con_2(t_1,t_2,\ldots,t_n)$$

$$\underbrace{r2(t_1,t_2,\ldots,t_n)}_{\text{Head}} \text{ :- } \underbrace{con_1(t_1,t_2,\ldots,t_n) \text{ ; } con_2(t_1,t_2,\ldots,t_n)}_{\text{Body: conditions for the rule to be true}}$$

Meaning	Predicate Calculus	PROLOG
And	∧	,
Or	∨	;
If	→	:-
Not	¬	not

Statements - Rules

Example 4.1

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

or

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

Statements - 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.
```

QUERIES

Asking questions about the facts and rules and draw conclusions.

Queries

- Retrieves the information from a logic program
- Asks whether a certain relation holds between terms
- Pattern-directed search
- Patterns in the same logic syntax as the database entries
- Searching the database from 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)

Query
Pattern matching from the first rule to the final rule

Database

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

- Unification *(next part)*
- Controlling the flow of satisfaction, e.g. cut ! *(next tutorial)*

‘Execution’ of Queries

- Can be regarded as
  - 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 alternative matching

Unification

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

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.
```

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

F = donut

Unification

- 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 are unified

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

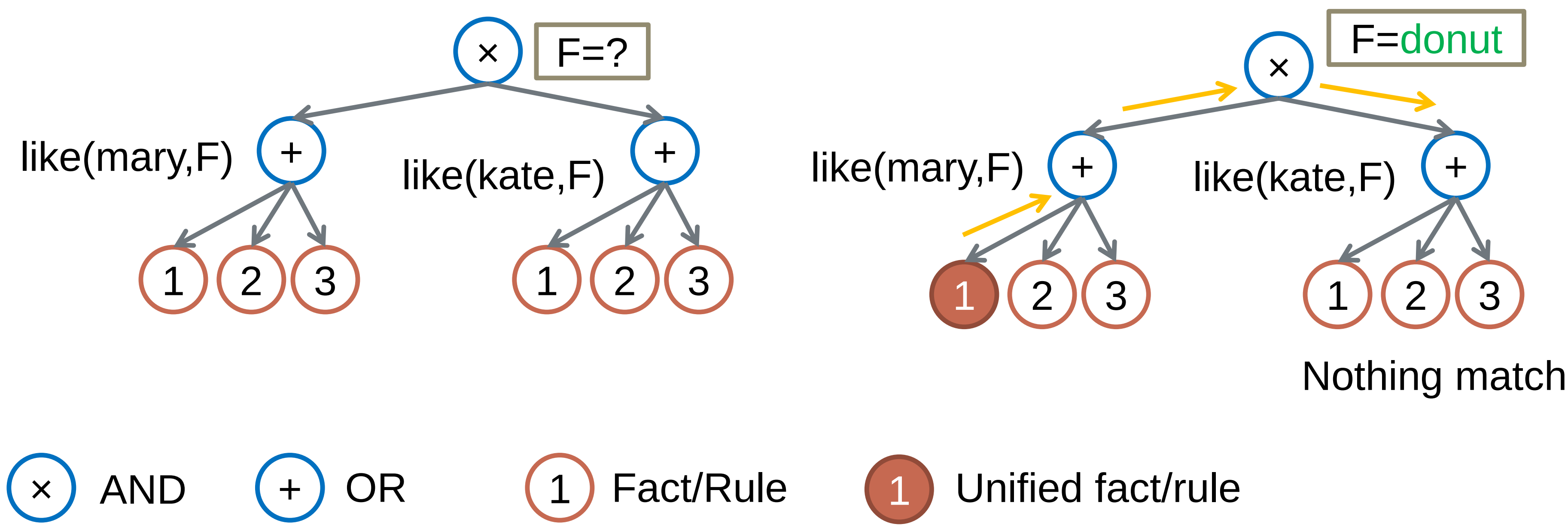
- When asking $P_1(..), P_2(..), …, P_n(..).$
  - If anyone fails (due to instantiation), say P_i, Prolog backtracks, and try an alternative of P_{i-1}
- After a successful query,
  - If user presses ‘;’, backtrack and try alternatives.
  - If user presses ‘.’, the query ends.

Backtracking Example

<u>Example 6</u>

```
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.
```

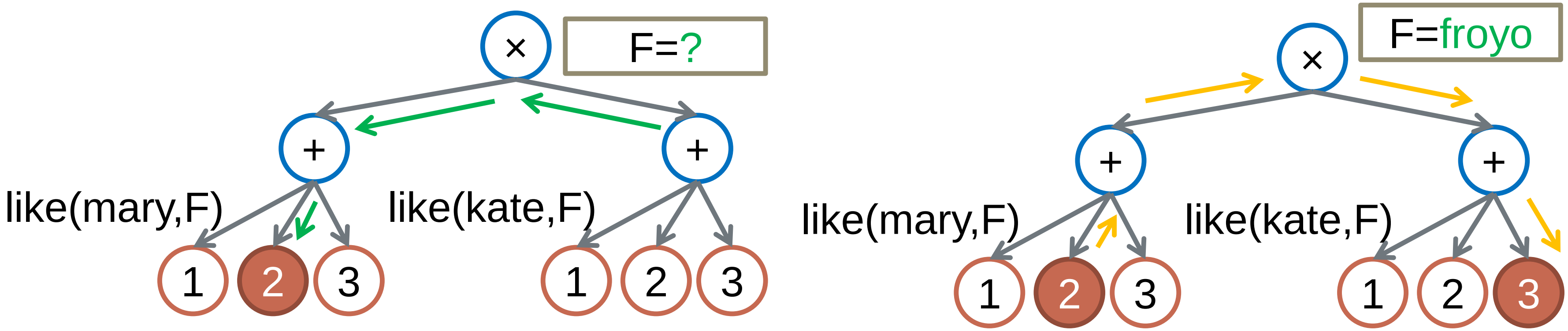


Backtracking Example (cont.)

Example 6

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

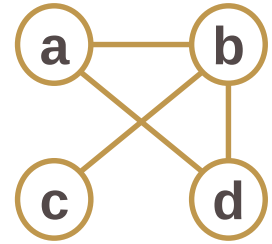
```
?- likes(mary,F),likes(kate,F).%Sth both Mary and Kate like
```



A BAD EXAMPLE

Why does my query fail using my database?

Satisfying Goals



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

Database

Explanation

i. **1.** → Return true → Press .

ii. 1. → 2.→ **3.**→ Return true → Press .

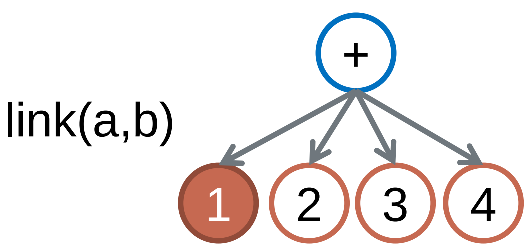
?- link(a,b).

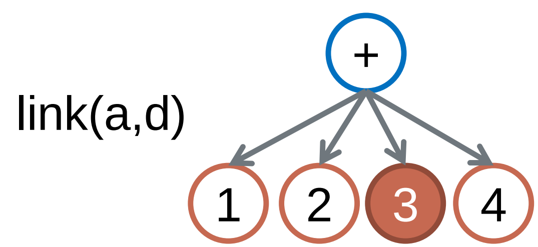
true. /* See i */

?- link(a,d).

true. /* See ii */

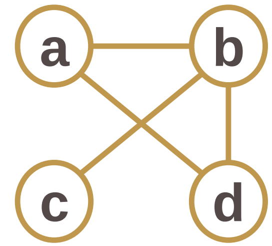
Queries





× AND + OR 1 Fact/Rule 1 Unified fact/rule

Satisfying Goals



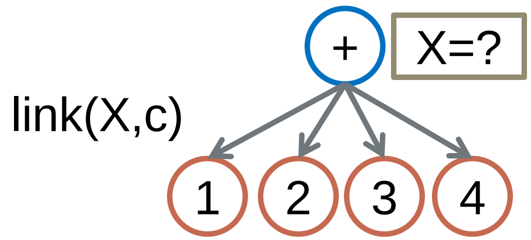
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).

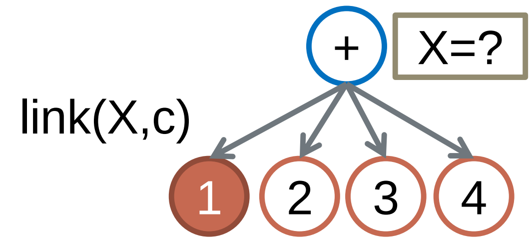
?- link(X,c).

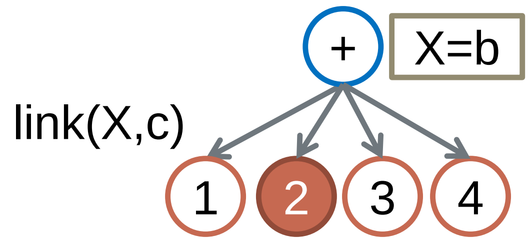
X = b.

Explanation

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





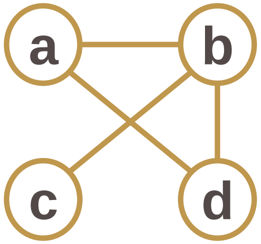


Match and return true.

Press . Done.

× AND   + OR   1 Fact/Rule   1 Unified fact/rule

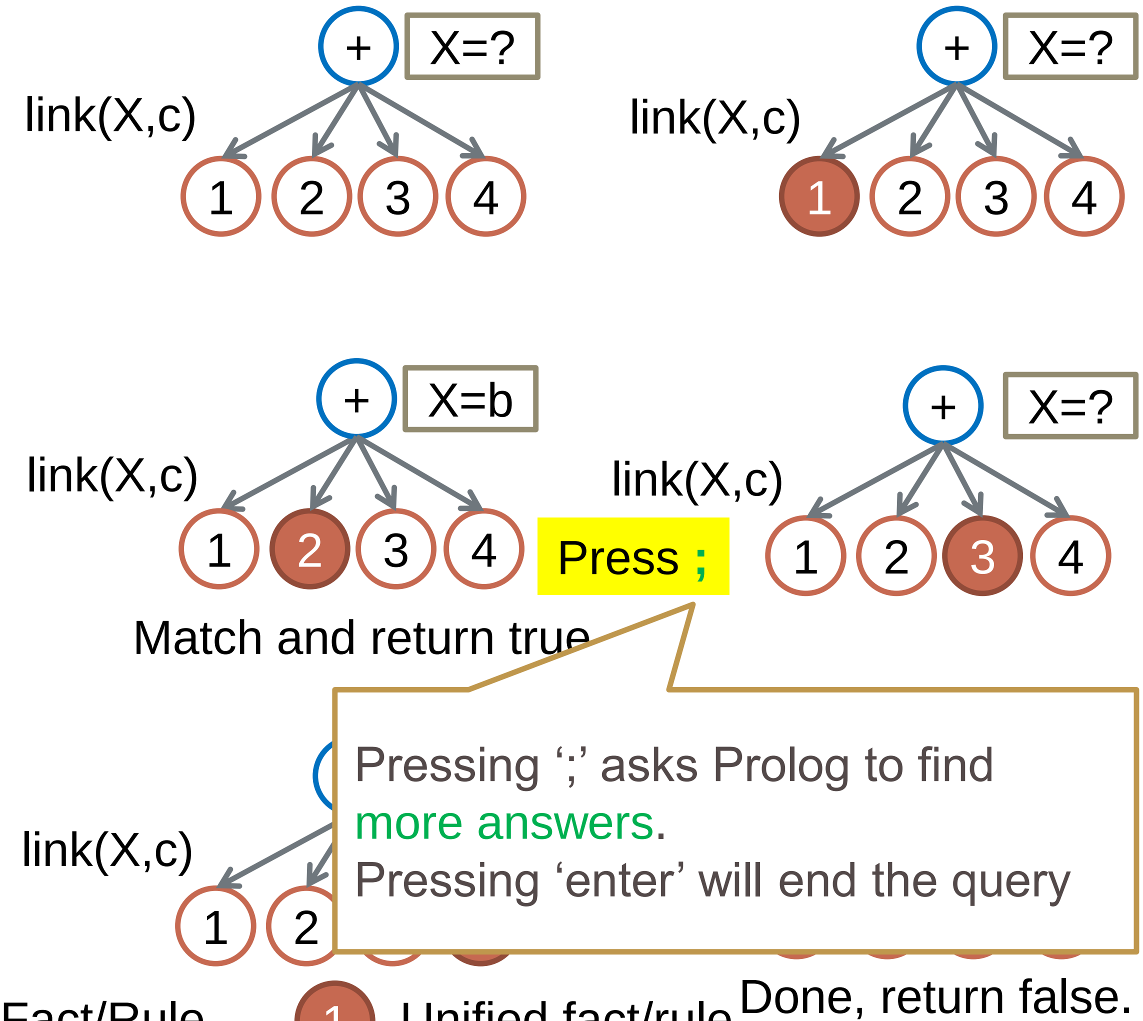
Using ; for more



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

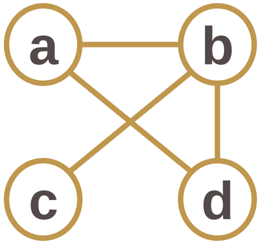
?- link(X,c).
X = b;
false.

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



× AND + OR 1 Fact/Rule 1 Unified fact/rule

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 ; */

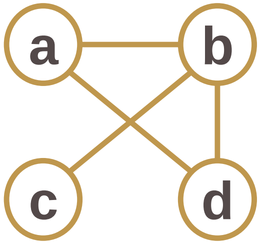
X = d.

Explanation

1. → Instantiate X to b → Return true → Press ;

→ 2. → **3.** → Instantiate X to d → Return true → Press .

False != Can't be true



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

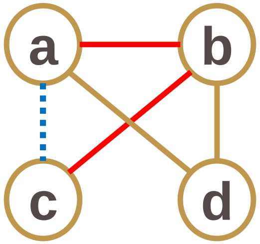
?- link(a,c).

false.

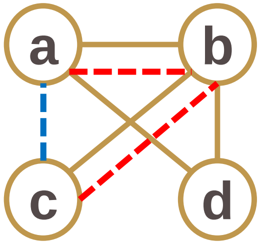
If Prolog answers "no", it doesn't mean that answer is definitely false. It means that the system cannot deduce that it is true given its database – **the Closed World Assumption**

Explanation

1. → 2.→ 3.→ 4.→ Return false



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

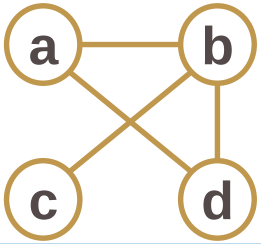
→ Match link(b,c)

→ 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 ;

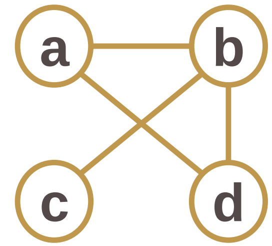
K = d ;

K = c ;

K = d ;

ERROR: Out of local stack

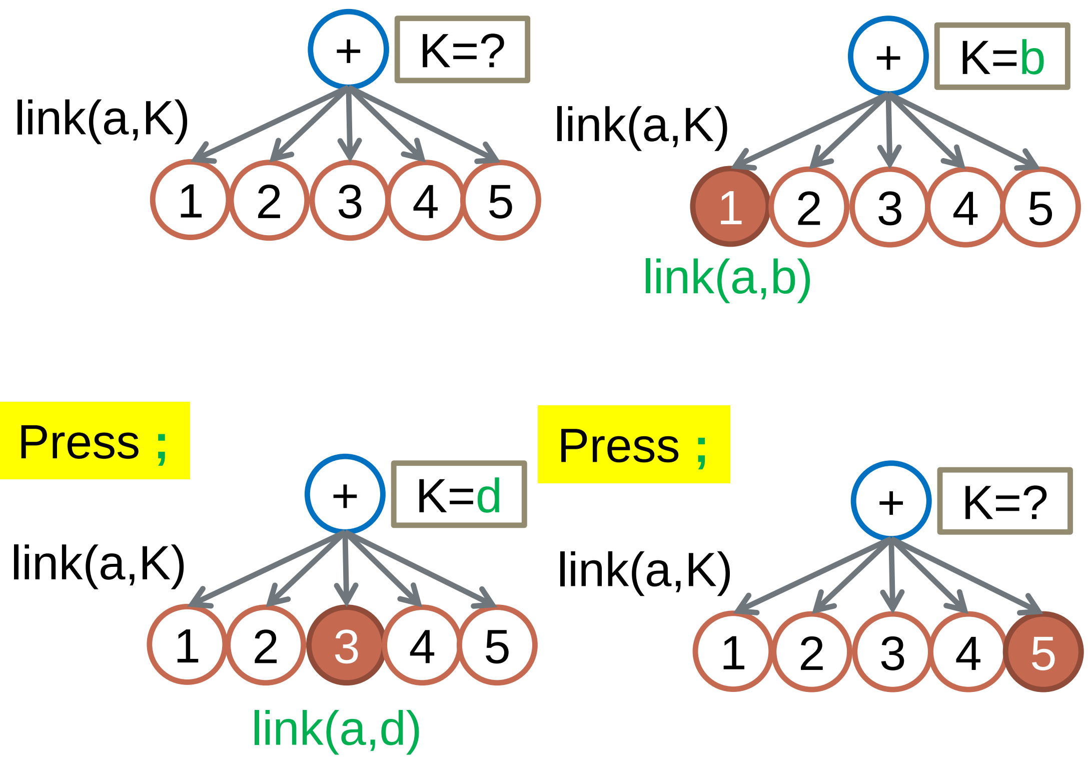
Queries - Example



a → K

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



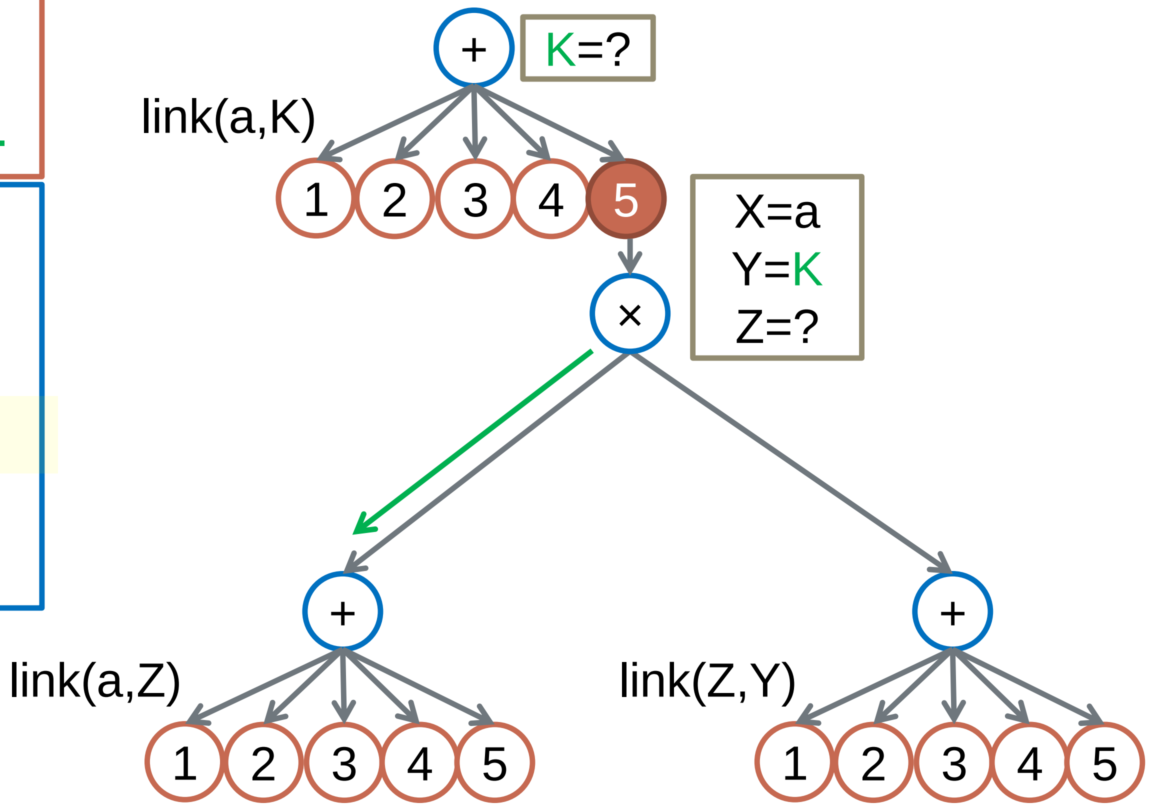
× AND   + OR   1 Fact/Rule   1 Unified fact/rule

Node Expansion

a → ? → K

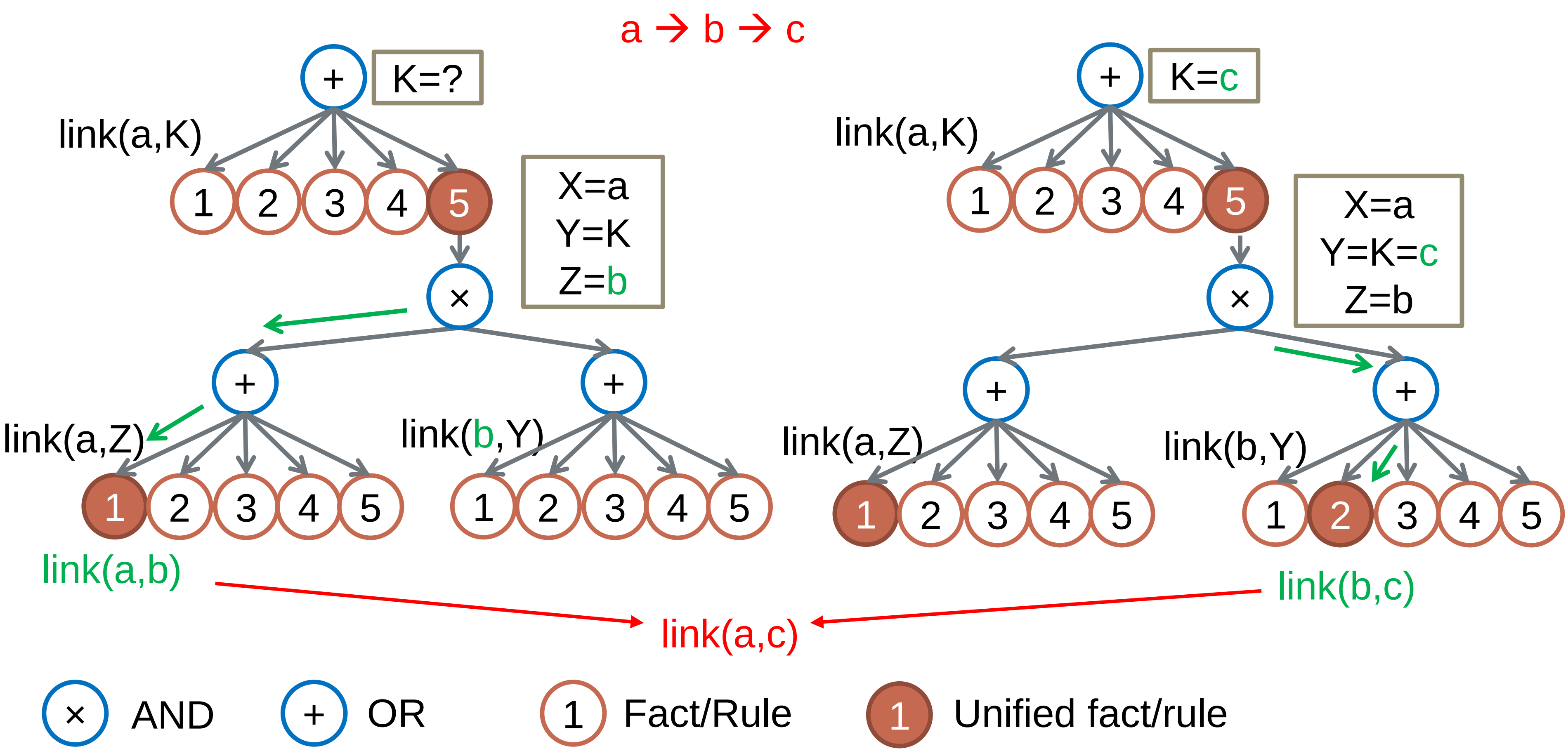
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

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



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



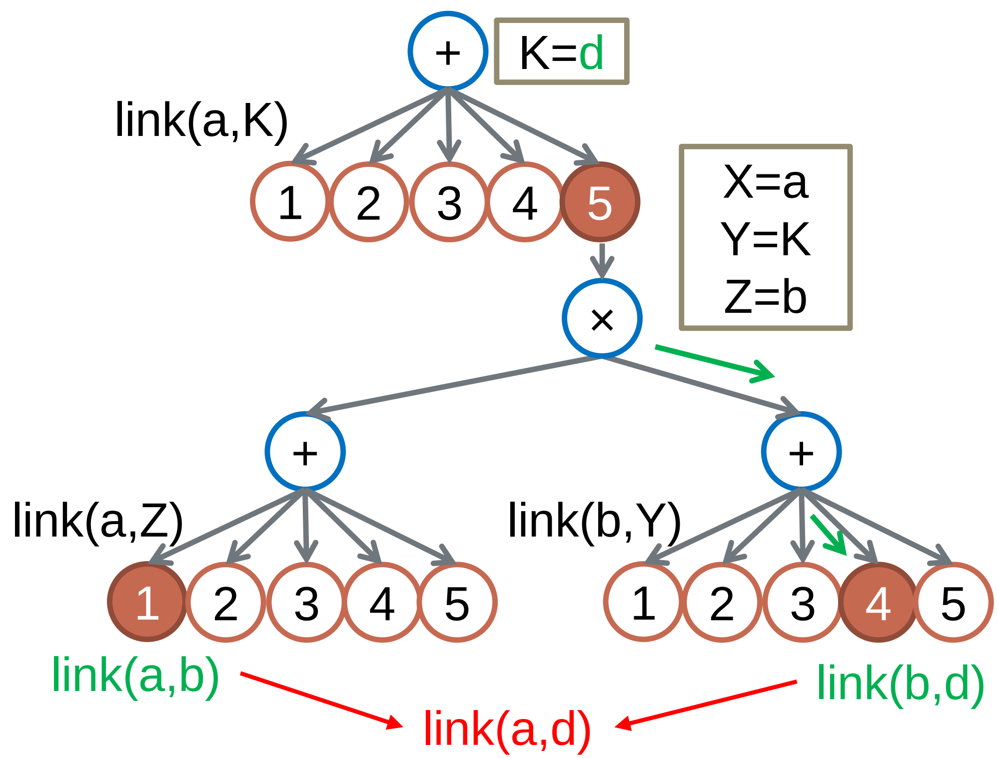
1. link(a,b).
2. link(b,c).
3. link(a,d).
4. link(b,d).
5. link(X,Y):- link(X,Z),link(Z,Y).

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

Press **;**

a → b → d

Try alternatives of the second clause



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).

a → b → ? → K

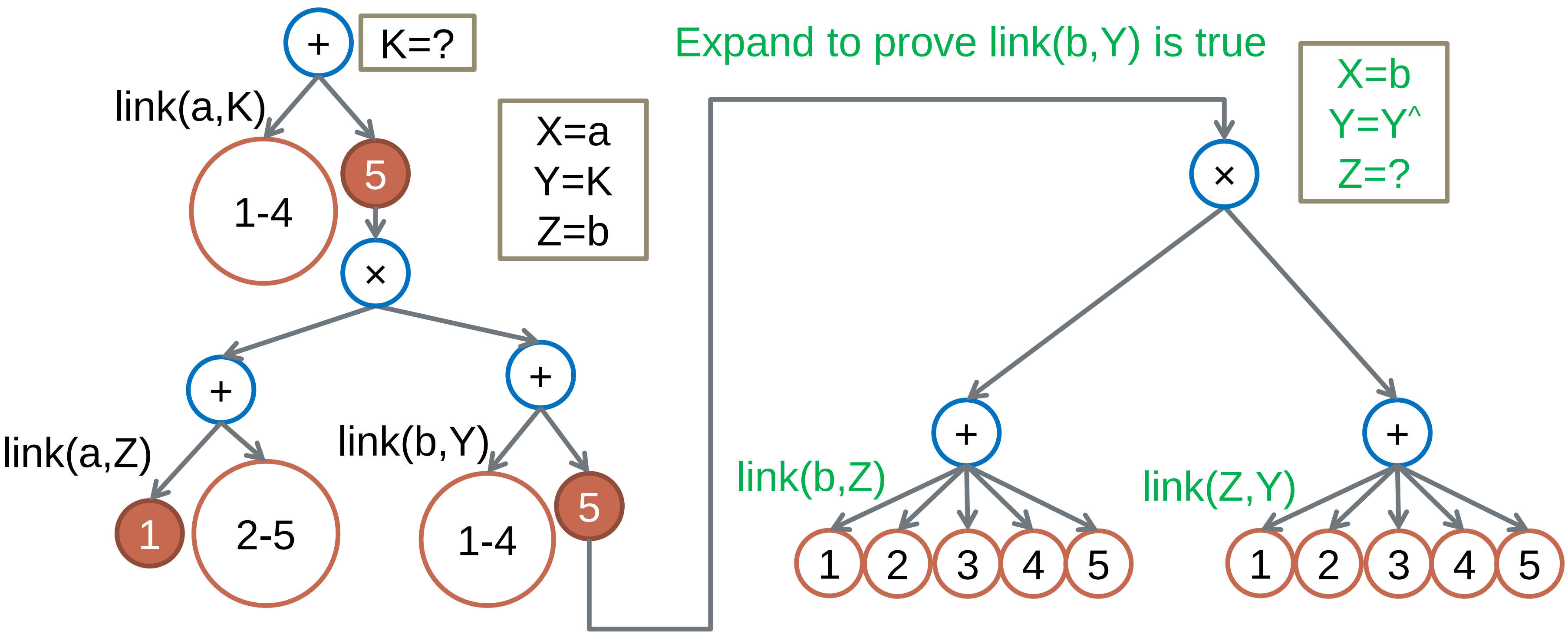
K = c ;

K = d ;

ERROR: Out of local stack

Press ;

Try alternatives of the second clause → 5



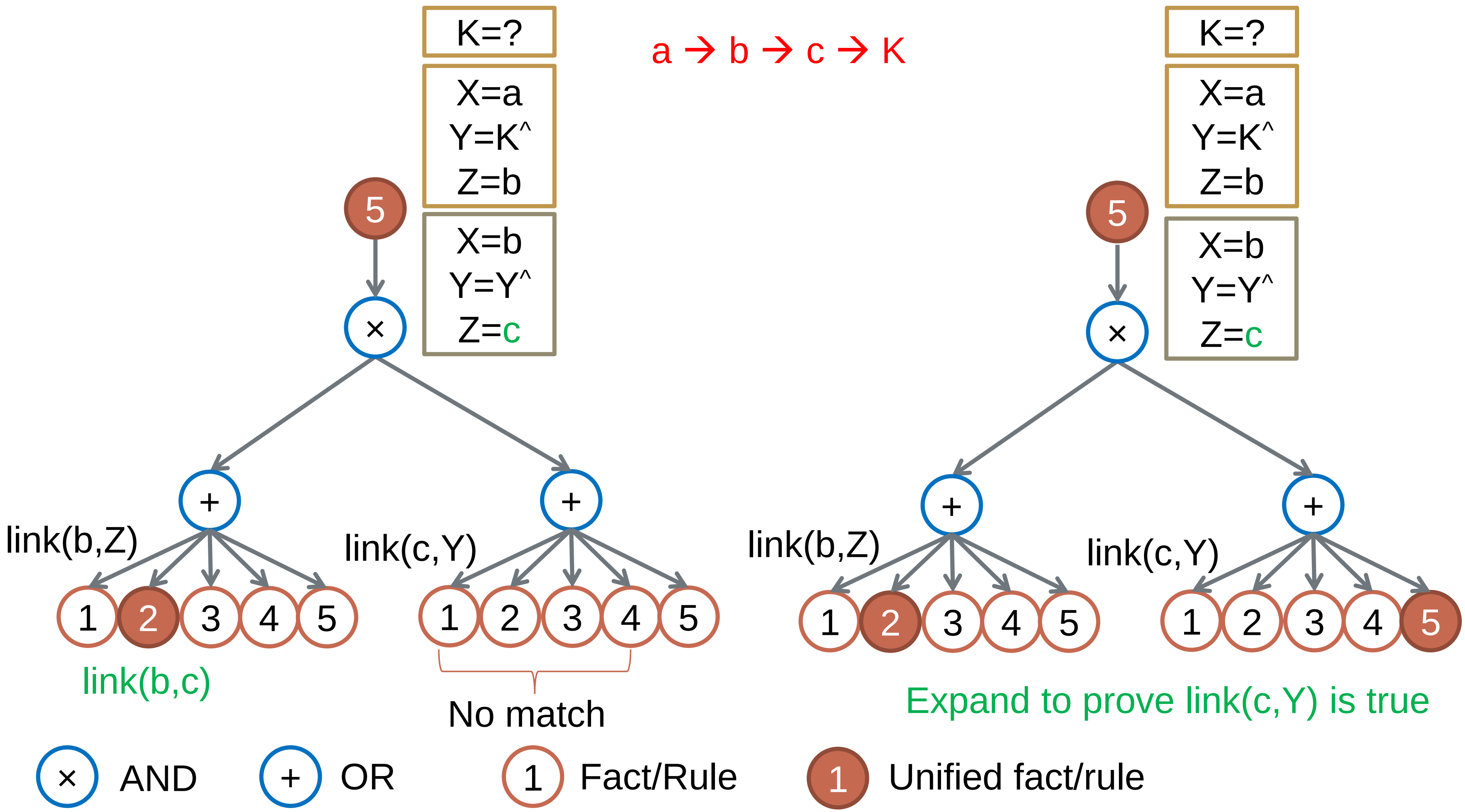
× AND + OR 1 Fact/Rule 1 Unified fact/rule

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).

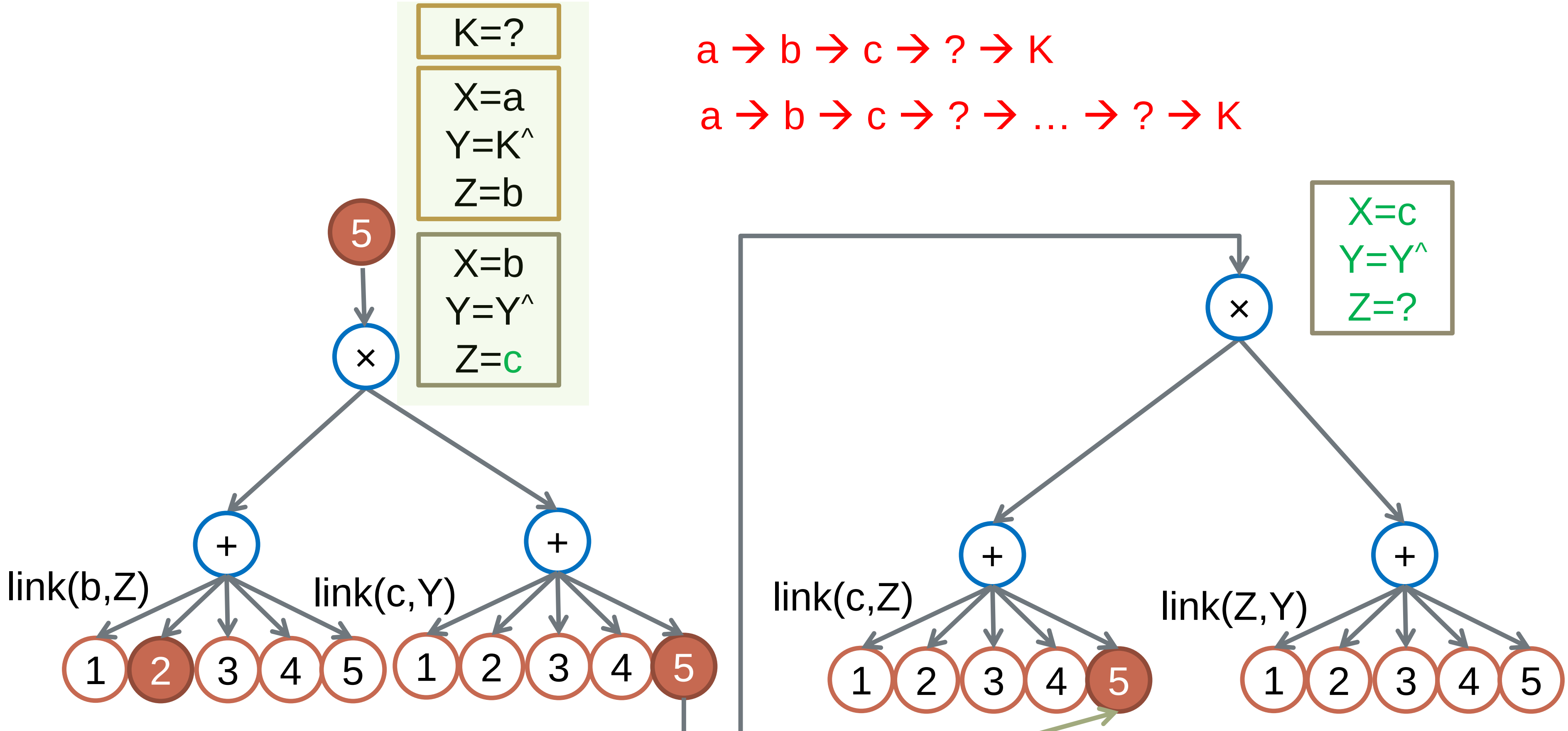
K = c ;

K = d ;

ERROR: Out of local stack

a → b → c → ? → K

a → b → c → ? → … → ? → K

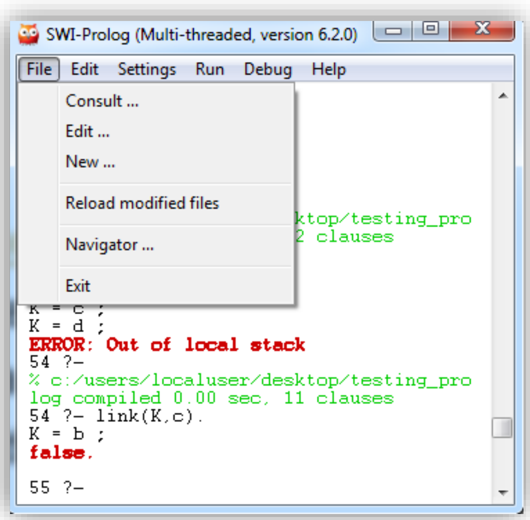


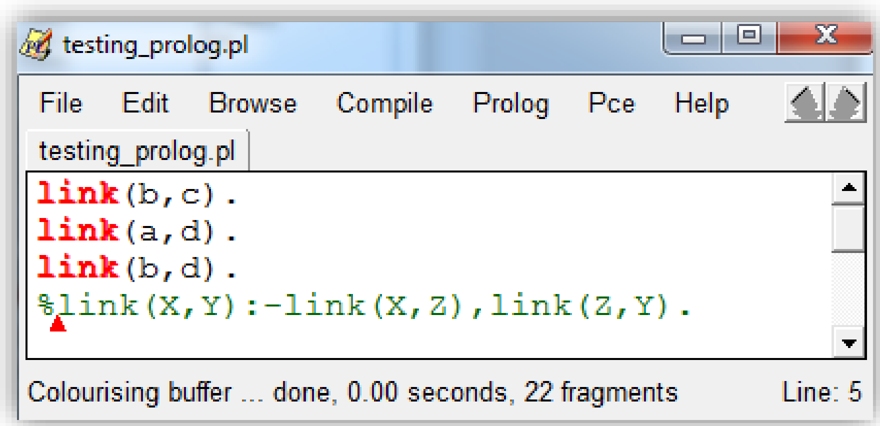
Expanding rule 5 to prove link(c,Z) is true which will repeat this step again!

?

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

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

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