Principles of Programming Languages
Lecture 4

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgment: parts of these slides are based on material by Diane Horton & Eric Joanis @ UoT

References: Scheme by Dybvig
Concepts of PL by Sebesta
Prog. in Prolog by Clocksin and Mellish
PL Concepts and Constructs by Sethi
ML for the Working Prog. By Paulson
PL Pragmatics by Scott

Prolog: example 1 – cont'd

Facts

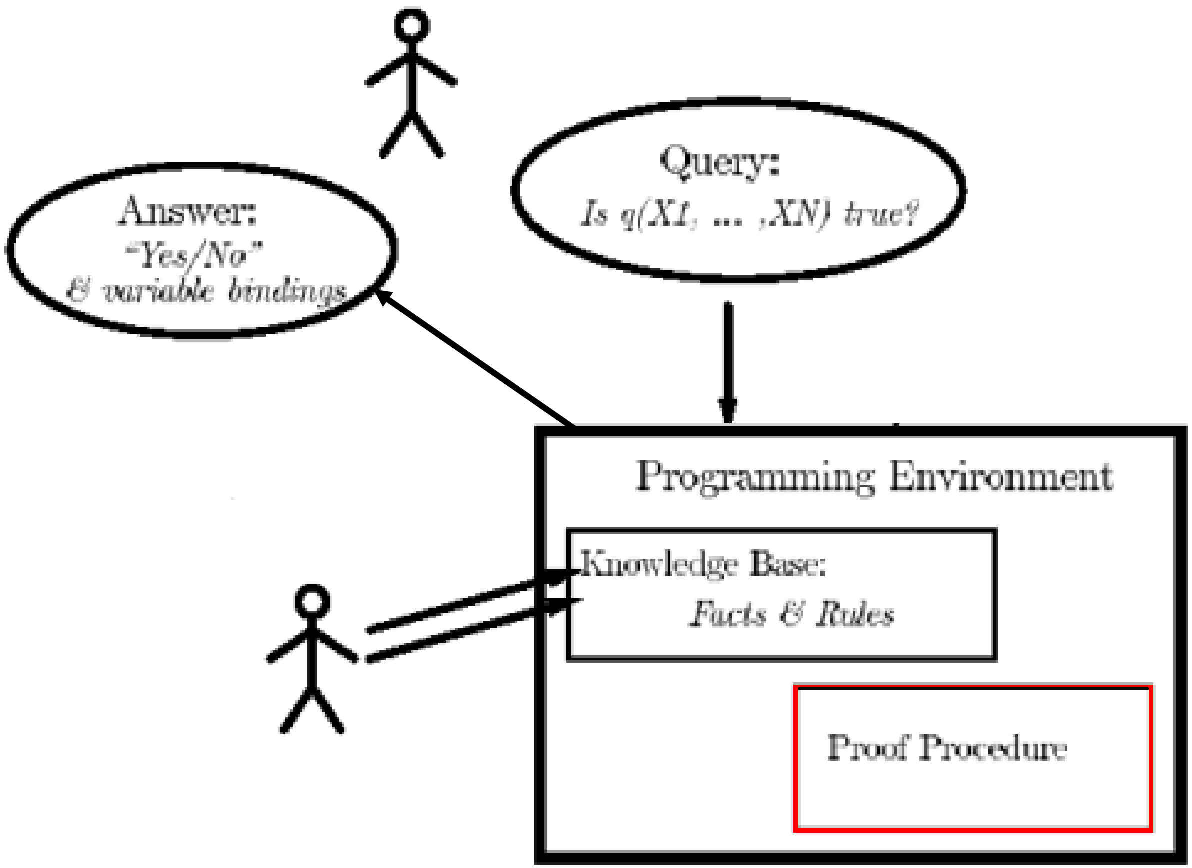
```
likes(eve, pie).      food(pie).
likes(al, eve).       food(apple).
likes(eve, tom).      person(tom).
likes(eve, eve).
```

```
?-likes(A,B).
A=eve,B=pie ; A=al,B=eve ; …
?-likes(D,D).
D=eve ; no
?-likes(eve,W), person(W).
W=tom
?-likes(al,V), likes(eve,V).
V=eve ;  no
```

and

Prolog: proof procedure

- **Two main processes:**
  - Unification
  - Top-down reasoning



Prolog: unification

- **First step in proof procedure**

- **Prolog tries to satisfy a query by *unifying* it with some conclusion and see if it is true!**

- **Process of finding these suitable "assignments" of values to variables is called *unification***
  – It is really a process of pattern matching to make statements identical
  – How does it compare to variable bindings in imperative world (C/C++/Java/python) ?

Prolog: unification – cont’d

- **Rules of unification:**

Object 1	Object 2	example		result
constant	free var.	4	X	X=4
bound variable	free variable	X	Y	Y gets the value of X
free variable	bound variable	X	Y	X gets the value of Y
bound variable	constant	X	b	fails if X has a value different then “b”
compound object with Variables	compound object with constants	f(X,Y)	f(2,3)	X=2, Y=3
compound object with nested compound object	compound object	f(q(2,X),3)	f(P,3)	succeds if P is free, and P=q(2,X) . (.. more posibilities)
compund object	compound object	f(3,X)	q(3,X)	fails, due to different functors (p is not q)

Prolog: unification – cont'd

- **Rules of unification:**

  - A constant unifies only with itself, it cannot unify with any other constant.

  - Two structures unify iff they have the same name, number of arguments and all the arguments unify.

  - Unification requires all instances of the same variable in a rule to get the same value

Prolog: unification – cont'd

- **Examples:**

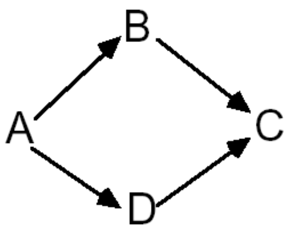
`a(b,C,d,E)` with `x( ... )`	doesn't unify: `a` and `x` differ
`a(b,C,d,E)` `a(_,_,_)`	no: different # of args
`a(b,C,d,E)` `a(j,f,G,H)`	no: $\mathtt{b} \neq \mathtt{j}$
`a(b,C,d,E)` `a(b,f,G,H)`	yes: by either $\{\mathtt{C} \mapsto \mathtt{f}, \mathtt{G} \mapsto \mathtt{d}, \mathtt{H} \mapsto \mathtt{E}\}$ or $\{\mathtt{C} \mapsto \mathtt{f}, \mathtt{G} \mapsto \mathtt{d}, \mathtt{E} \mapsto \mathtt{H}\}$
`a(pred(X,j))` `a(pred(k,j))`	yes: $\{\mathtt{X} \mapsto \mathtt{k}\}$
`a(pred(X,j))` `a(B)`	yes: $\{\mathtt{B} \mapsto \mathtt{pred(X,j)}\}$

Prolog: unification – cont'd

- **Examples:**
  - Does p(X,X) unify with p(b,b) ?
  - Does p(X,X) unify with p(b,c) ?
  - Does p(X,b) unify with p(Y,Y) ?
  - Does p(X,Z,Z) unify with p(Y,Y,b) ?
  - Does p(X,b,X) unify with p(Y,Y,c) ?
    - To make the third arguments equal, we must unify X with c
    - To make the second argument equal, we must unify Y with b.
    - So, p(X,b,X) becomes p(c,b,c), and p(Y,Y,c) becomes p(b,b,c).
    - However, p(c,b,c) and p(b,b,c) are not identical → different atoms → different semantics

Prolog: example 2

- **Facts & rules:**



link(a,b), link(b,c), link(a,d), link(d,c).
path(N, N).
path(L, M) :- link(L, X), path(X, M).

- **Posing queries:**

Based on our logical encoding of the graph, we can then write queries:

?- path(a,c)
yes

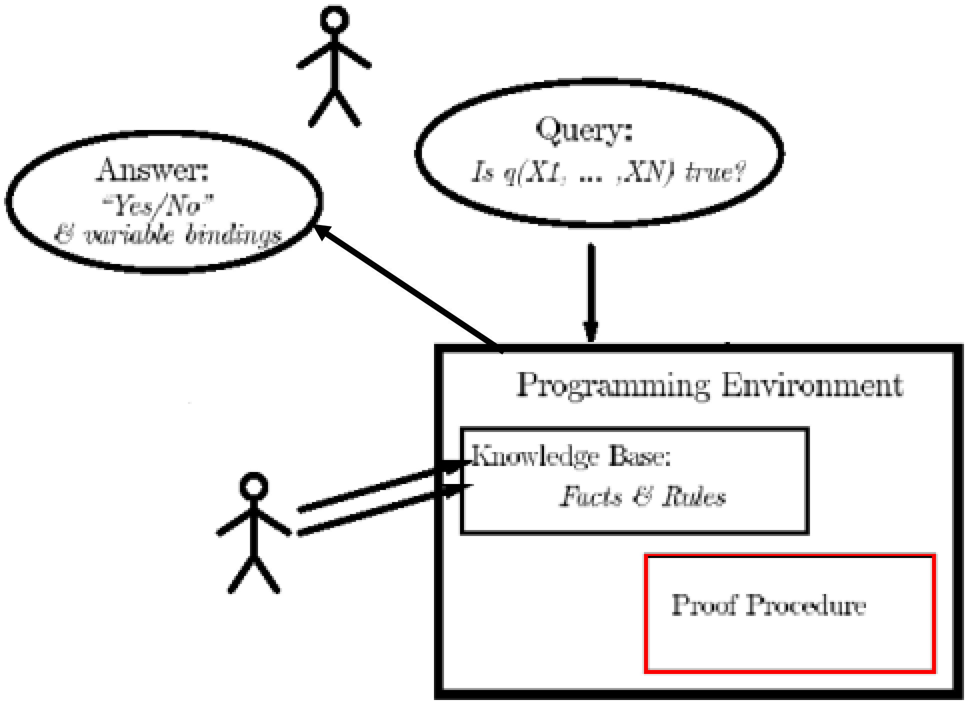
?- path(c,a)
no

?- path(a,X), path(X,c)
X = a
X = b
X = c
X = d

Notice that we didn't write a graph traversal algorithm, and we didn't hard code the set of questions we can ask in advance. We just define what a graph is…

Prolog: proof procedure - revisited

- **Two main processes:**
  - ✓ Unification
  - – Top-down reasoning



Prolog: reasoning

- **Given a set of facts and rules, we need a mechanism to deduce new facts and/or prove that a given rule is true or false or has no answer**

- **There are two techniques to do this:**
  - Bottom-up reasoning
  - Top-down reasoning

Prolog: bottom-up reasoning

- Bottom-up (or forward) reasoning: starting from the given facts, apply rules to infer everything that is true.

  e.g., Suppose the fact B and the rule $A \leftarrow B$ are given. Then infer that A is true.

A rule base:

```
A <- B        (1)
B <- C        (2)
C             (3)
```

A bottom-up proof:

infer A
↑ rule (1)
infer B
↑ rule (2)
infer C
↑ rule (3)
start

So, A is proved

Example

Rule base:

```
p(X,Y,Z) <- q(X),q(Y),q(Z).
q(a1).
q(a2).
...
q(an).
```

Bottom-up inference derives n^3 facts of the form p(a_i,a_j,a_k):

```
p(a1, a1, a1)
p(a1, a1, a2)
p(a1, a2, a3)
...
```

Prolog: top-down reasoning

- Top-down (or backward) reasoning: starting from the query, apply the rules in reverse, attempting only those lines of inference that are relevant to the query.

  e.g., Suppose the query is A, and the rule $A \leftarrow B$ is given. Then to prove A, try to prove B.

A rule base:

```
A <- B        (1)
B <- C        (2)
C             (3)
```

A top-down proof:

goal A

↓ rule (1)

goal B

↓ rule (2)

goal C

↓ rule (3)

success

So, A is proved

Prolog: top-down reasoning – cont'd

- **Multiple rules and multiple premises:**
  - A fact may be inferred by many rules
    - E.g.
      ```
      E <- B
      E <- C
      E <- D
      ```
  - A rule may have many premises
    - E.g.
      ```
      E <- B /\ C /\ D
      ```
- **In top-down inference, such rules give rise to**
  - Inference trees
  - Backtracking

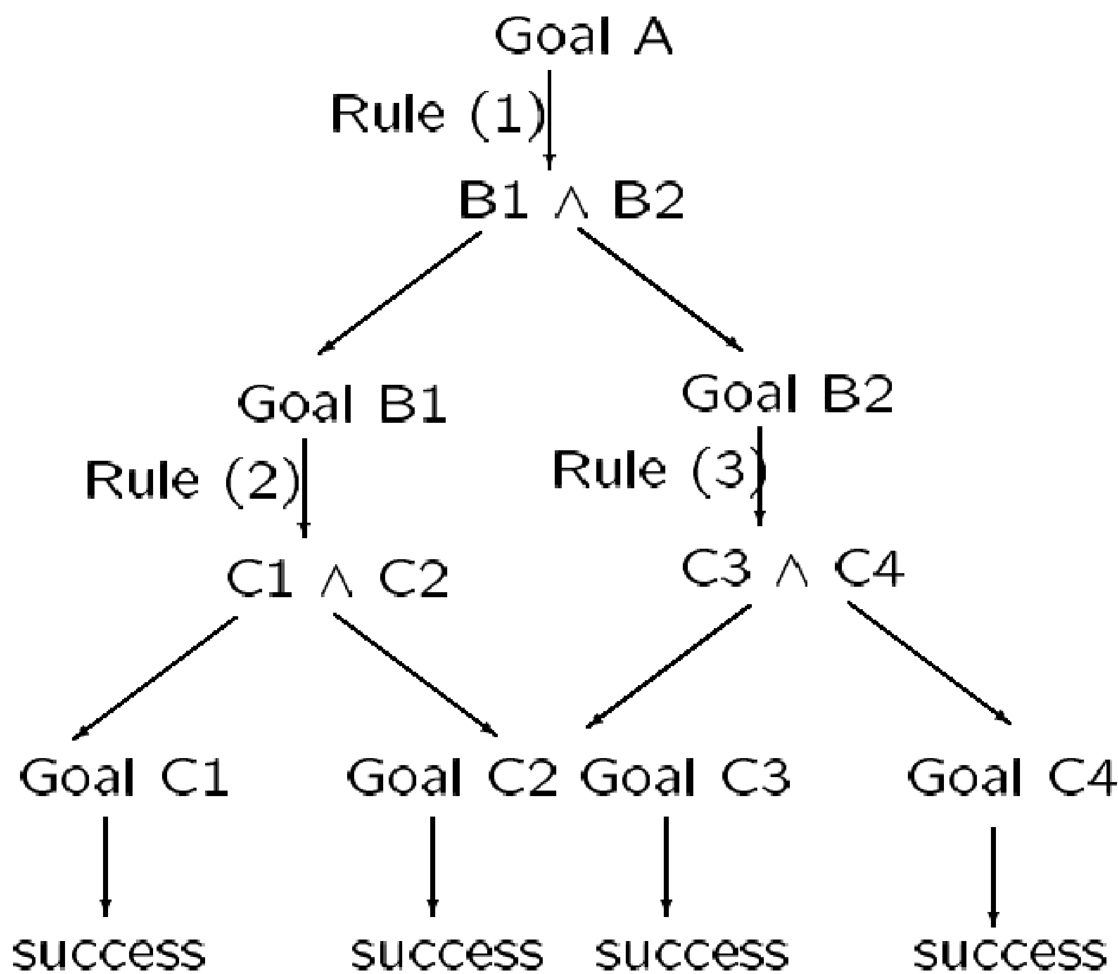
Prolog: top-down reasoning – cont'd

- **Example:** *multiple premises*

Rule base:

```
(1)    A  <- B1 /\ B2
(2)    B1 <- C1 /\ C2
(3)    B2 <- C3 /\ C4
       C1    C2    C3    C4
```

Query: Is A true?



So, goal A is proved. (all paths must succeed)

Prolog: top-down reasoning – cont'd

- **Example:** *multiple rules*

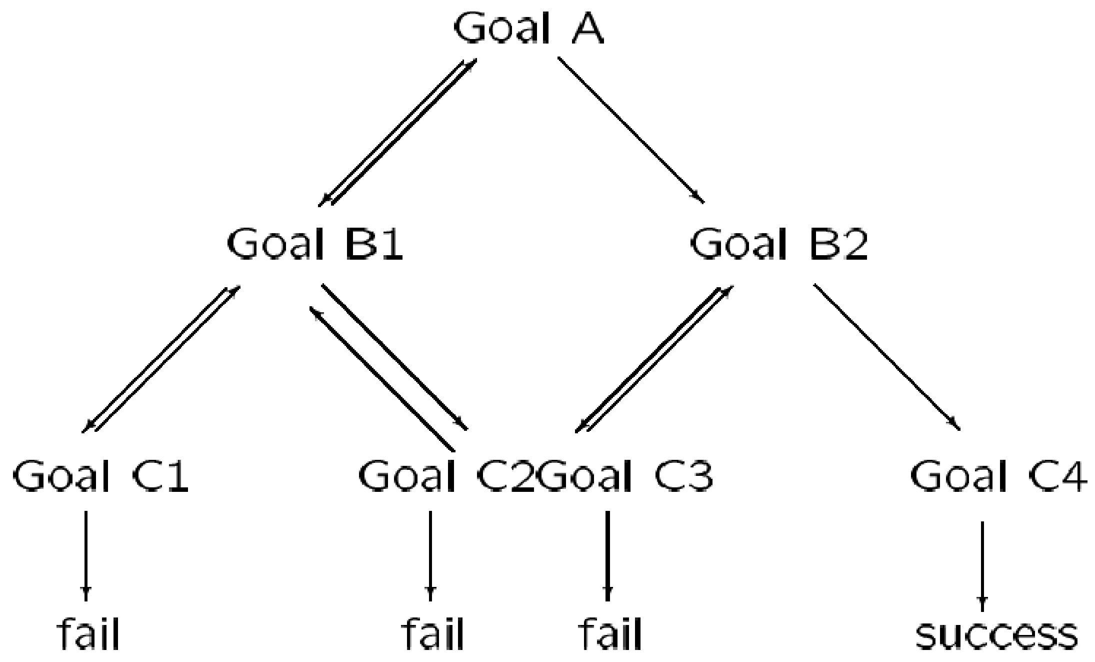
Rule base:

A <- B1   B1 <- C1   B2 <- C3

A <- B2   B1 <- C2   B2 <- C4

C4

Query: Is A true?



So, goal A is proved. (only one path must succeed

Prolog: backtracking

- **Prolog uses this algorithm for proving a goal by recursively breaking goal down into sub-goals and try to prove these sub-goals until facts are reached.**

- **To satisfy a goal:**
  - Try to unify with conclusion of first rule in database
  - If successful, apply substitution to first premise, try to satisfy resulting sub-goals
  - Then apply both substitutions to next sub-goal (premise), and so on...
  - If not successful, go on to the next rule in database
  - If all rules fail,try again (backtrack) to a previous sub-goal

Prolog: backtracking example 1

<u>Rule base</u>:

```
p(X) :- q(X),r(X).
q(d).   q(e).   q(f).   q(g).
r(e).   r(g).
```

<u>Query</u>: Find `X` such that `p(X)` is true.