COMPUTATIONAL LOGIC

TOPICS 2013

Krysia Broda

Deduction as Query Answering

Prolog - A Logic Programming language

Prolog lets us describe properties about things

eg whether a sentence conforms to a grammar
 whether an element is a member of a list
 whether a list is a sublist of another
 whether a placing of pieces in a game is valid

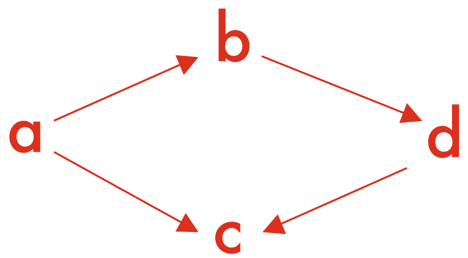
We can use a sentence checking program to check a given sentence or to derive correct sentences

Haskell is functional, whereas Prolog is relational

Rules describing relations (1)

We might have some *facts* about a graph

```
eg node(a).  node(b).  node(c).  node(d).
   arc(a,b). arc(a,c). arc(b,d). arc(d,c).
```

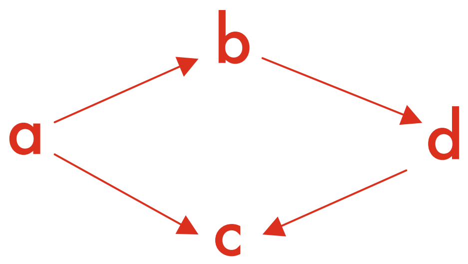


We call this a *theory*, or a *program* and we can ask it a *query*

Which nodes are connected by an arc to c?

`? arc(W,c).` "find W s.t. arc(W,c) is true"

Rules describing relations (2)



We may also have *rules* describing what a path is

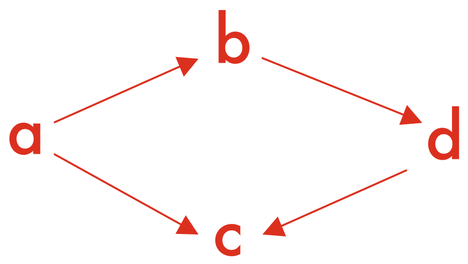
```
eg path(X,Y) :- arc(X,Y).
```

"path from X to Y is true if arc from X to Y is true"

```
path(X,Y) :- arc(X,Z), path(Z,Y).
```

"for any X,Y and Z, path(X,Y) holds
if arc(X,Z) holds and path(Z,Y)holds"

Querying a Program



Is there a path from a to c?

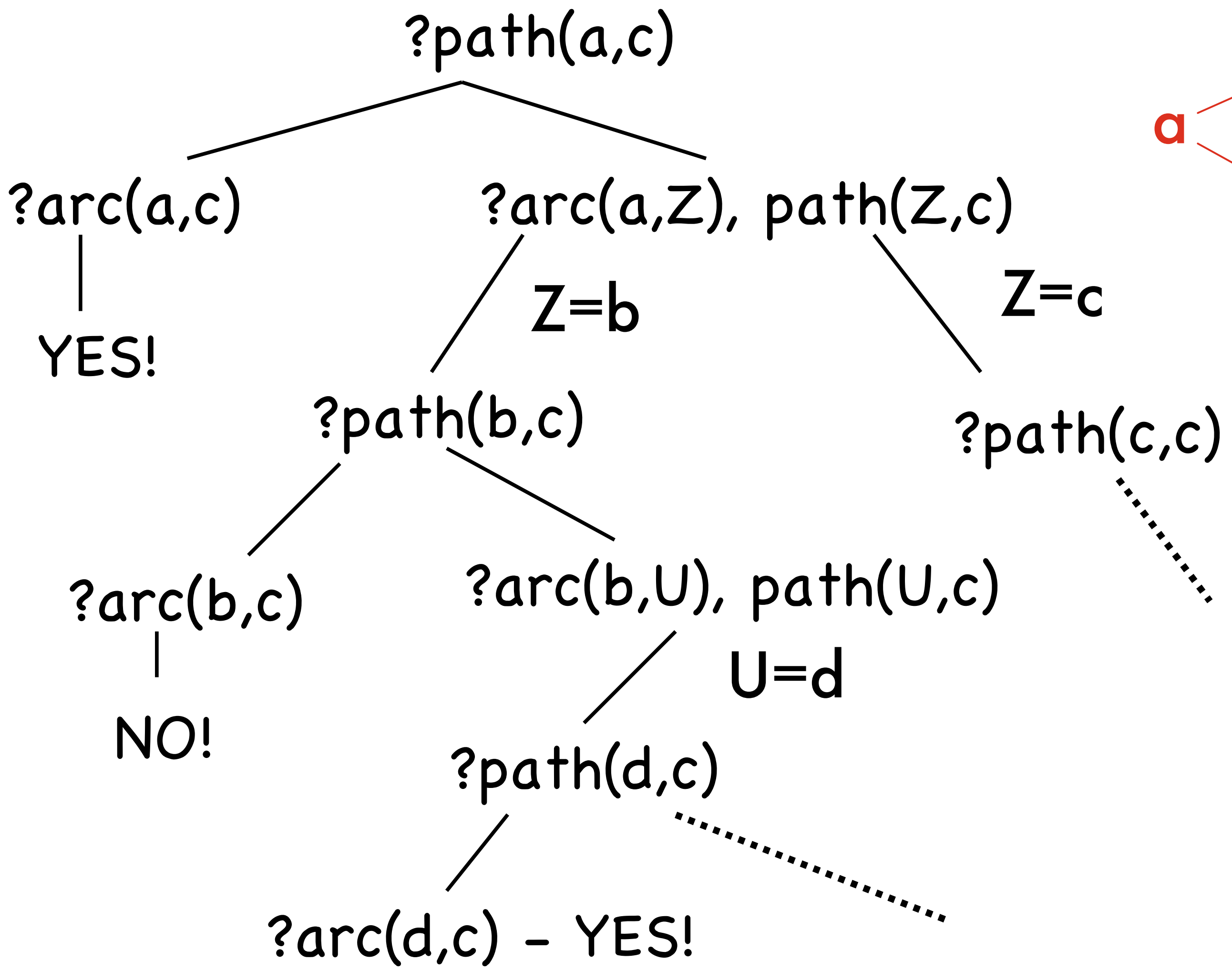
```
? path(a,c).
```

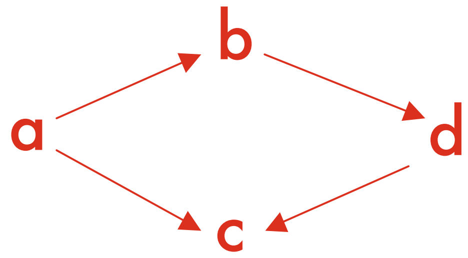
For which nodes is there not a path to b?

```
? node(W), \+path(W,b).
```

```
  "find nodes W s.t. path(W,b) is not true"
```

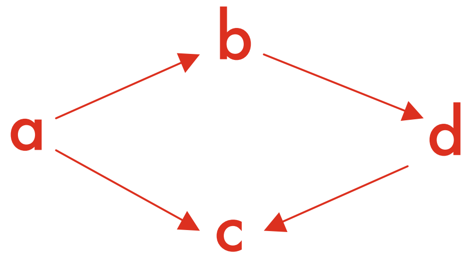
How Prolog deduces the answers (1)





How Prolog deduces the answers (2)

to find W: `node(W), \+path(W,b)`



Prolog assumes that path(a,b) is false if it cannot be deduced - *negation by failure*

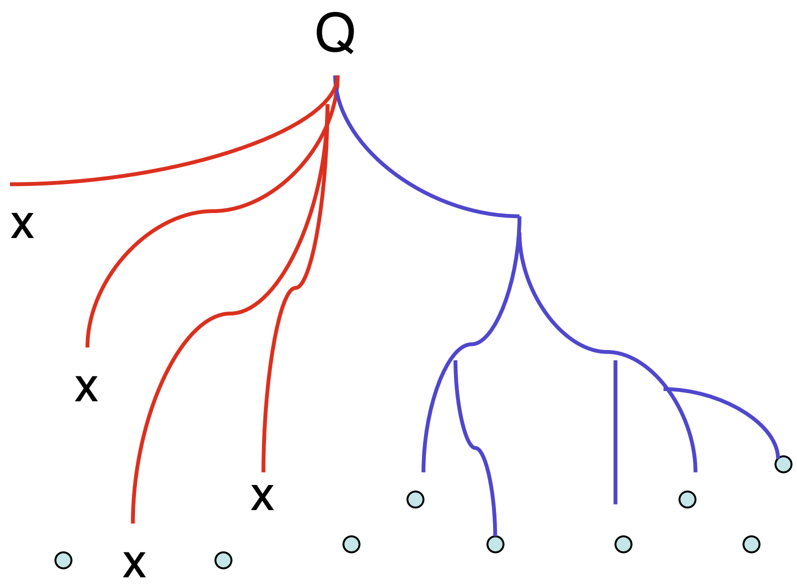
node(a) is true - is path(a,b) false?
path(a,b) can be deduced, so it is <u>not</u> false.

node(c) is true - is path(c,b) false?
path(c,b) cannot be deduced:
    to do so requires arc(c,b) or arc(c,Z) for some Z;
    neither is true - so path(c,b) is false and c is an answer.

node(b) is true - is path(b,b) false?
path(b,b) cannot be deduced ......

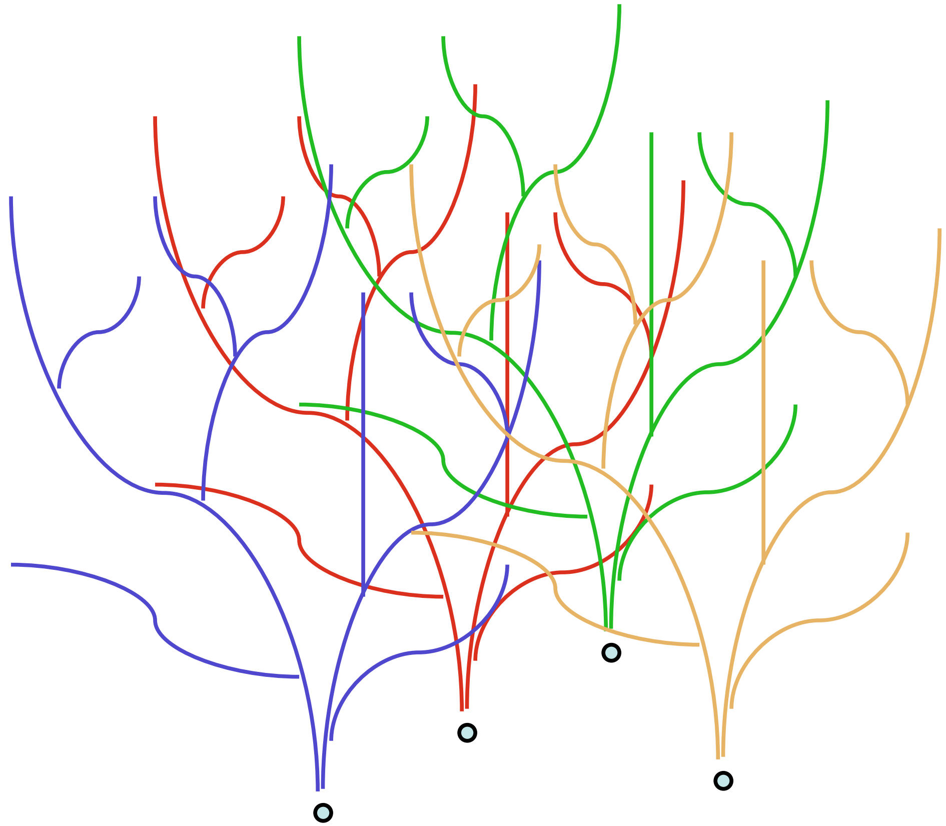
Different Computation Strategies

Prolog



Prolog reduces a query Q to sub-queries .... until it reaches known facts

ASP



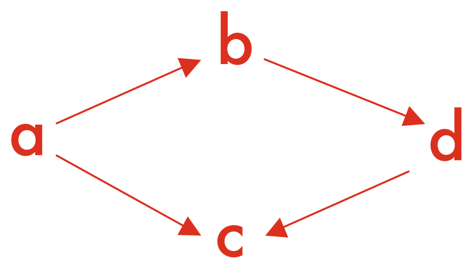
From the known facts ASP finds all consequences at once ....

ASP - another Logic Programming Language

ASP - finds all facts implied by the program at once

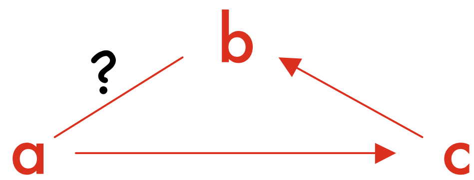
```
node(a).   node(b).   node(c).    node(d).
arc(a,b). arc(a,c). arc(b,d). arc(d,c).
source(X) :- node(X), not arcTo(X).
arcTo(X) :- arc(Y,X).
```

node(a), node(b), node(c), node(d),
arc(a,b), arc(a,c), arc(b,d), arc(d,c),
arcTo(b), arcTo(c), arcTo(d),
source(a)



ASP - another Logic Programming Language

```
node(a).  node(b).  node(c).
arc(a,c).  arc(c,b).
1{arc(a,b),arc(b,a)}1.
path(X,Y) :- arc(X,Y).
path(X,Y) :- path(Z,Y),arc(X,Z).
:-path(X,X).
```



node(a), node(b),
node(c),
arc(a,b), arc(c,b),
arc(a,c),
path(a,b), path(a,c),
path(c,b)

node(a), node(b), node(c),
arc(b,a), arc(a,c), arc(c,b),
path(b,a), path(a,c), path(c,b),
path(c,a), path(b,c), path(a,b),
path(b,b), path(a,a), path(c,c)

ASP - Answer Set programming

OTTER - A GENERAL THEOREM PROVER

Otter uses *clausal form* (disjunctions of literals)

1. Write data in logic and convert to clauses
2. Write conclusion in logic
3. Negate conclusion and convert to clauses
4. Derive a contradiction

Otter uses the deduction rule of *resolution*

c(d) v c(e) and ¬c(e) v h(e) ==> c(d) v h(e)

c(d) v c(e) and ¬c(X) v h(X) ==> c(d) v h(e)

both by *resolution*

An Example: Two naughty children

Either Dolly and Ellen was the culprit but only one.

The culprit was in the house.

Dolly: " It wasn't me, I wasn't in the house; Ellen did it."

Ellen: " I didn't do it; Dolly was in the house."

Neither told the truth.

Who did it?

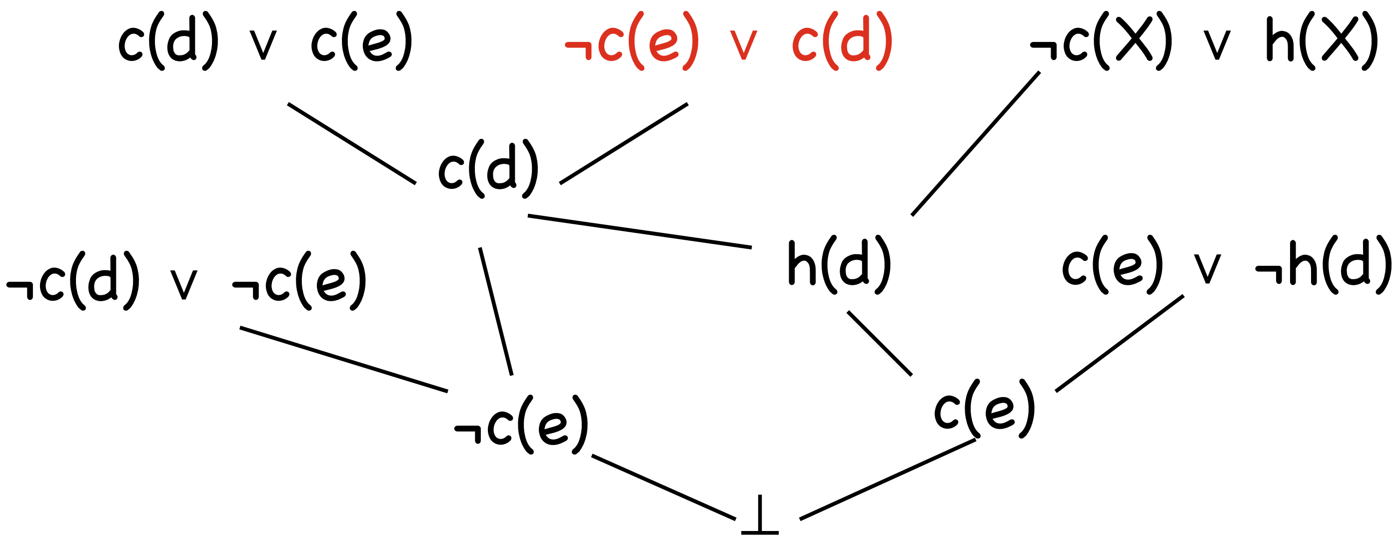
Two naughty children in Logic

1. $c(d) \vee c(e)$	At least one of 2 girls did it
2. $\neg c(d) \vee \neg c(e)$	Exactly one of them was the culprit
3. $\neg c(X) \vee h(X)$	The culprit was in the house
4. $c(d) \vee h(d) \vee \neg c(e)$	Negation of dolly's testimony
5. $c(e) \vee \neg h(d)$	Negation of Ellen's testimony
6. $\neg c(e) \vee c(d)$	Negation of goal

Two naughty children in Logic

1. $c(d) \vee c(e)$ 2. $\neg c(d) \vee \neg c(e)$ 3. $\neg c(X) \vee h(X)$

4. $c(d) \vee h(d) \vee \neg c(e)$ 5. $c(e) \vee \neg h(d)$ 6. $\neg c(e) \vee c(d)$



7. (1+6) $c(d)$ 8. (7+3) $h(d)$ 9. (8+5) $c(e)$

10. (7+2) $\neg c(e)$ 11. (10+9) $\perp$

Using Automated Reasoning in our Course

Reasoning about Programs - year 1

Prolog language - year 2
Artificial Intelligence - year 2

Software verification - year 3
Machine Learning - year 3

Automated Reasoning - year 4
Probabilistic Inference and Data Mining - year 4
Multi-Agent Systems - year 4

plus bits here and there in other courses

and in projects