CSE 473

Lecture 11

Chapter 7

Inference in Propositional Logic

Logic: another thing that penguins aren't very good at.

Recall: Propositional Logic Terminology

Literal

= proposition symbol or its negation

E.g., A, $\neg A$, B, $\neg B$, etc. (positive vs. negative)

Clause

= disjunction of literals

E.g., $(B \vee \neg C \vee \neg D)$

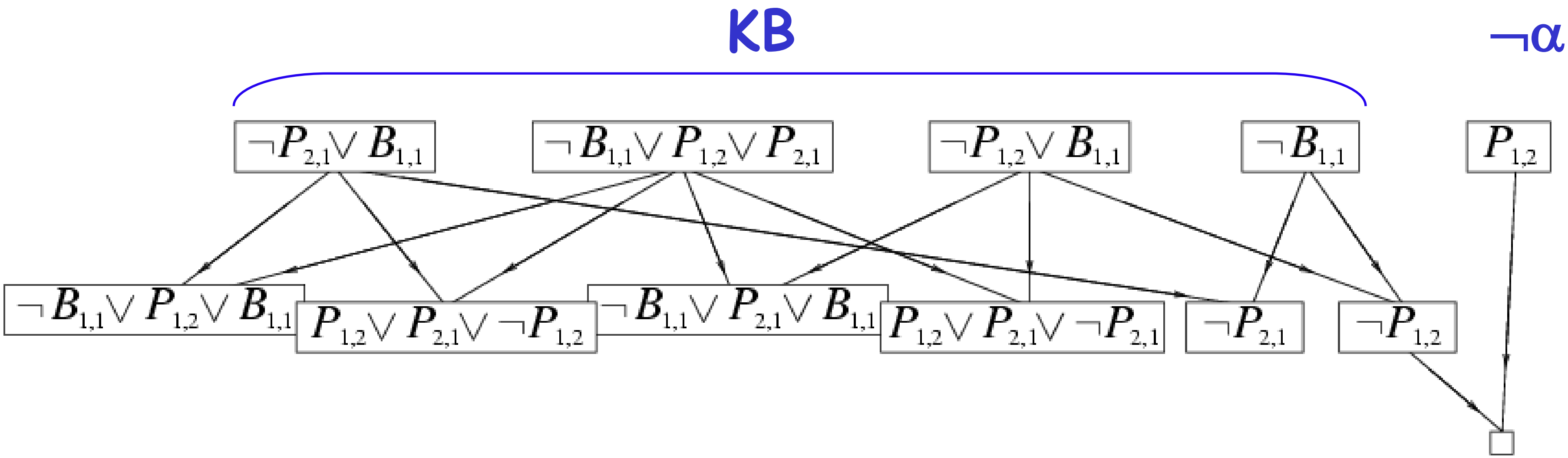
Conjunctive Normal Form (CNF):

sentence = conjunction of clauses

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Can think of **KB** as a **conjunction of clauses**, i.e. one long sentence

Review: Inference Technique I: Resolution



You got a literal and its negation

Empty clause

What does this (empty clause) mean?

Recall that KB is a *conjunction* of all these clauses

Is $P_{1,2} \wedge \neg P_{1,2}$ satisfiable? No!

Therefore, KB $\wedge \neg \alpha$ is unsatisfiable, i.e., KB $\models \alpha$

Inference Technique II: Forward/Backward Chaining

- Requirement: Sentences need to be in **Horn Form**:

  KB = conjunction of Horn clauses

  $\neg P_{2,1} \vee B_{1,1}$

  Horn clause =

  - proposition symbol or
  - "(conjunction of symbols) $\Rightarrow$ symbol"

    (i.e. clause with at most 1 positive literal)

  E.g., KB = $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

- F/B chaining based on "Modus Ponens" rule:

$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

  Complete for Horn clauses

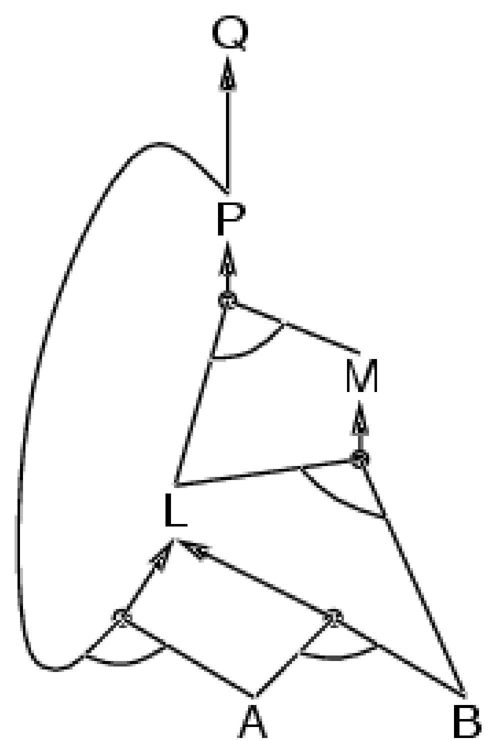
- Very natural and linear time complexity in size of KB

Forward chaining

- Idea: fire any rule whose premises are satisfied in *KB*, add its conclusion to *KB*, until query q is found

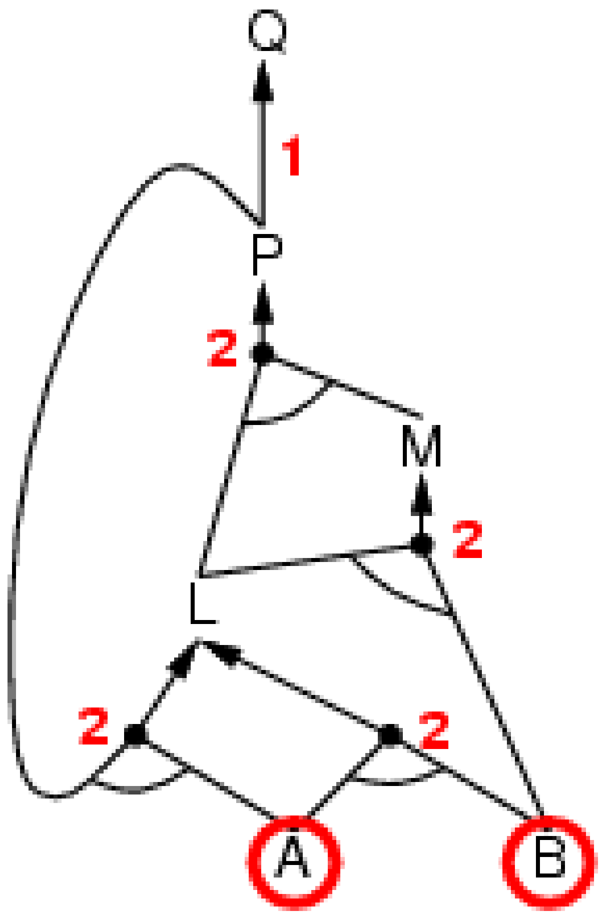
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Query = "Is Q true?"



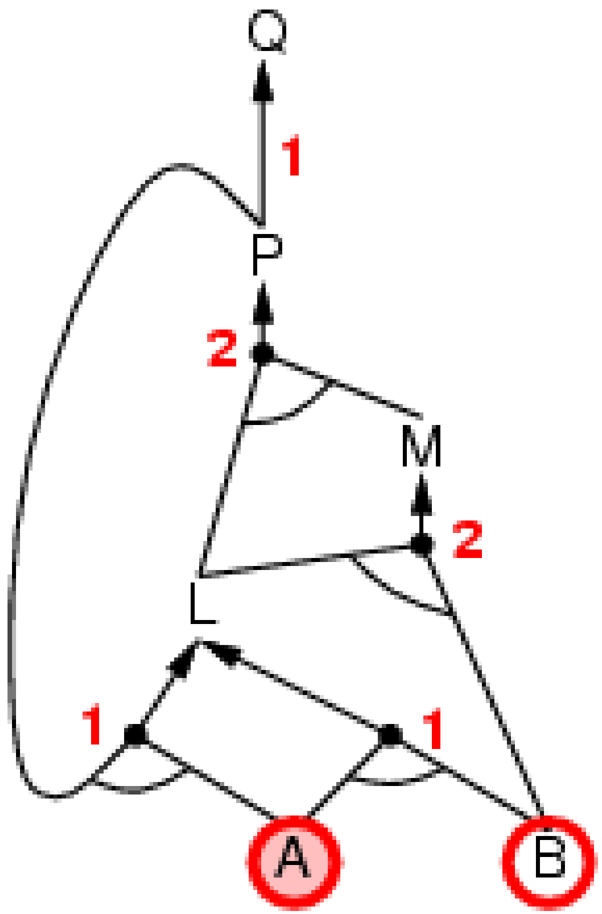
AND-OR Graph

Forward chaining example

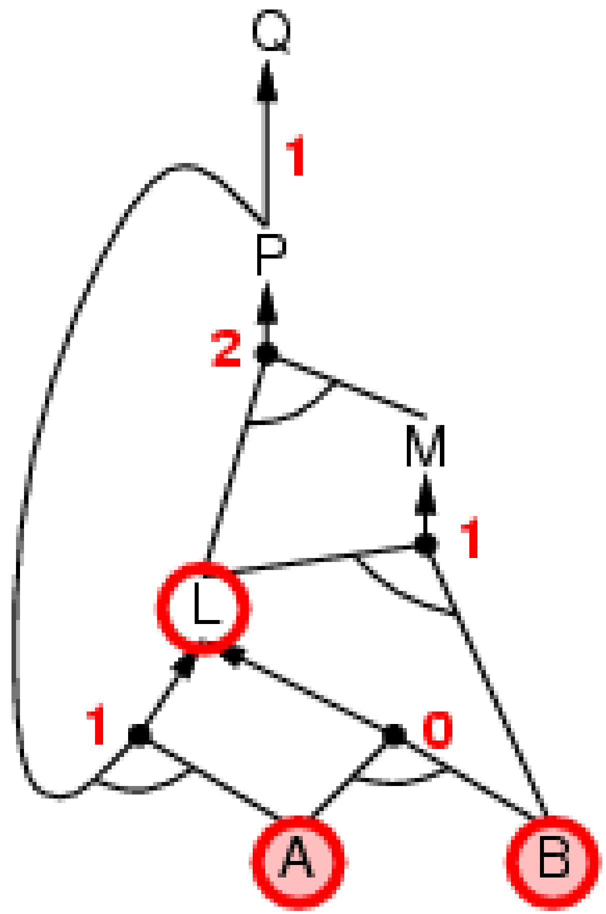


Query = Q
(i.e. "Is Q true?")

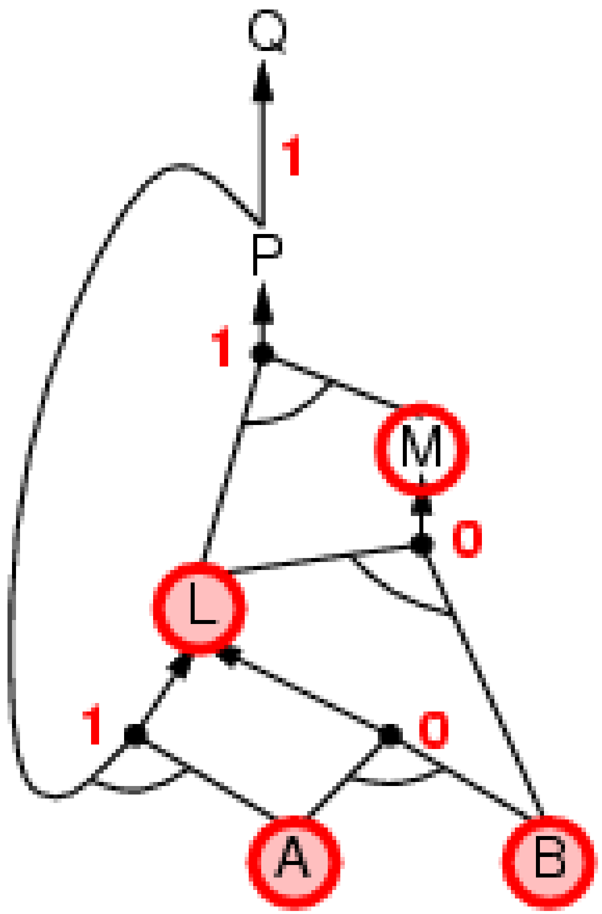
Forward chaining example



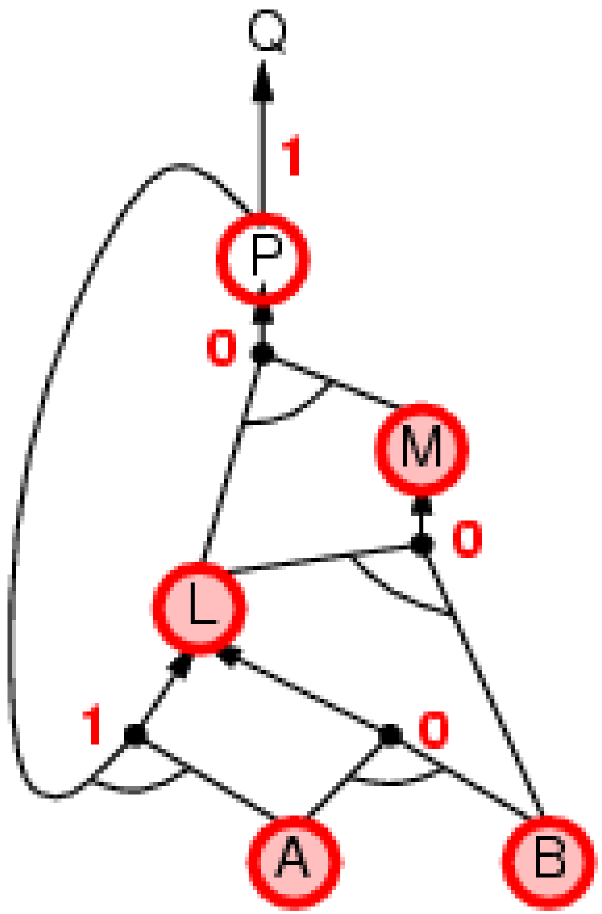
Forward chaining example



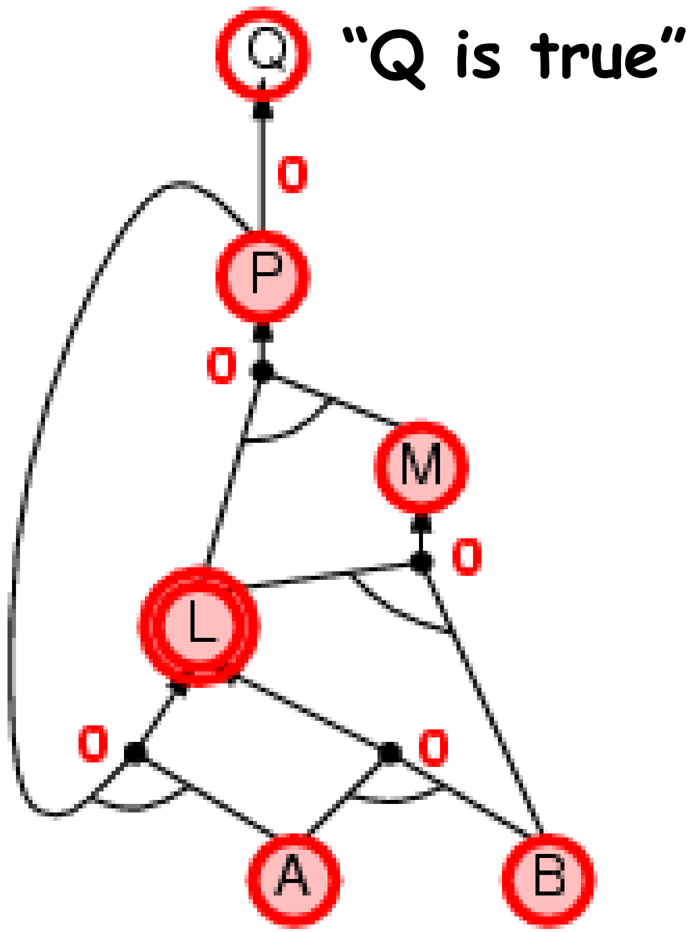
Forward chaining example



Forward chaining example



Forward chaining example



Forward chaining algorithm

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known to be true

   while agenda is not empty do
       p ← POP(agenda)
       unless inferred[p] do
           inferred[p] ← true
           for each Horn clause c in whose premise p appears do
               decrement count[c]
               if count[c] = 0 then do
                   if HEAD[c] = q then return true
                   PUSH(HEAD[c], agenda)
   return false
```

Forward chaining is sound & complete for Horn KB

Backward Chaining (BC)

Idea: work backwards from the query q:

To prove q by BC,

check if q is known to be true already, or

prove by BC all premises of some rule concluding q

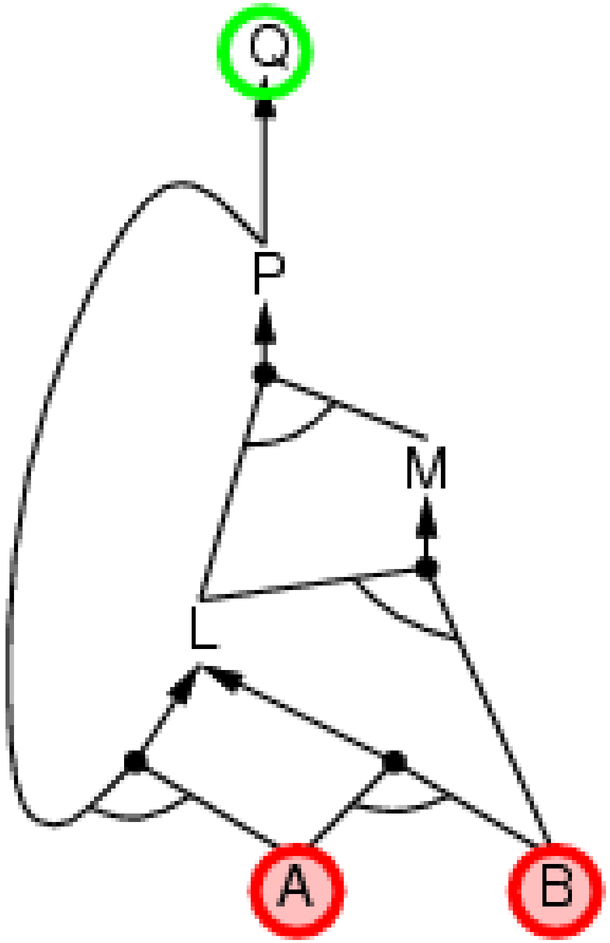
(each premise to be proved is a subgoal)

Avoid loops: check if new subgoal is already on goal stack

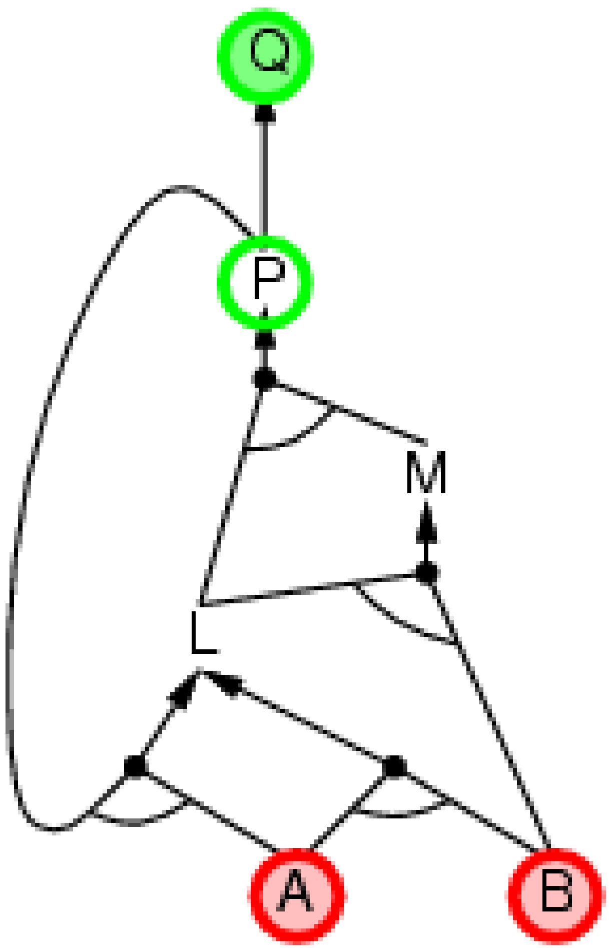
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

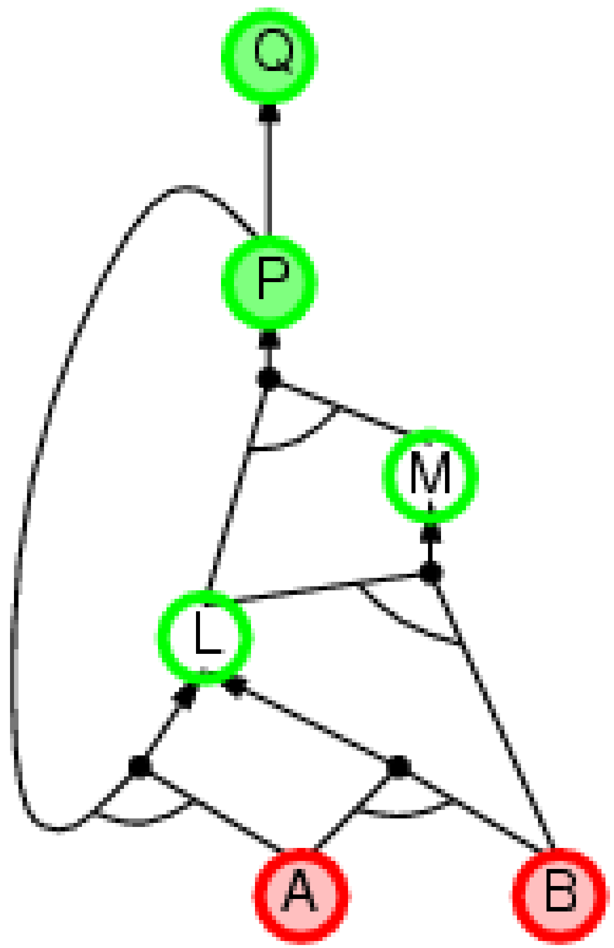
Backward chaining example



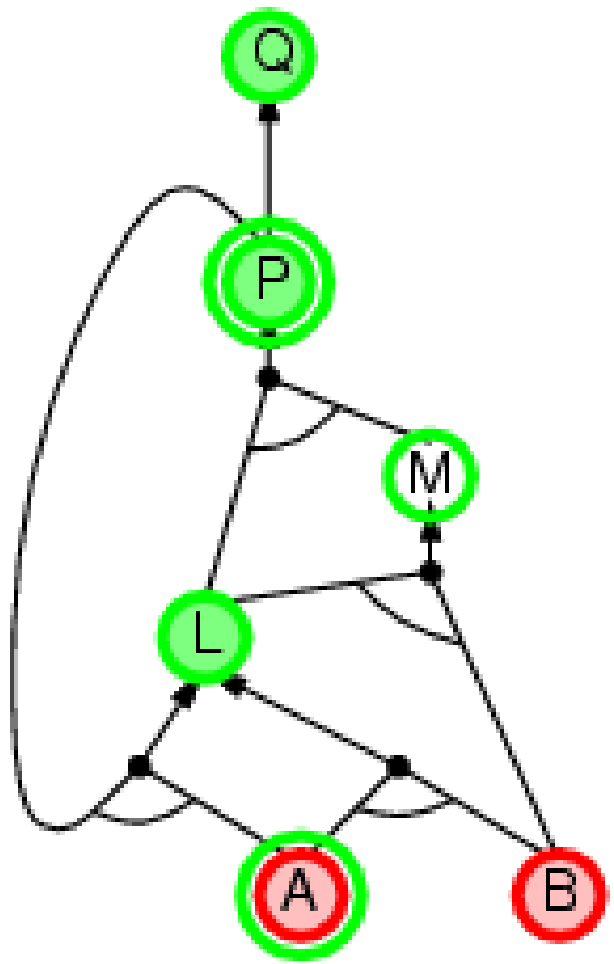
Backward chaining example



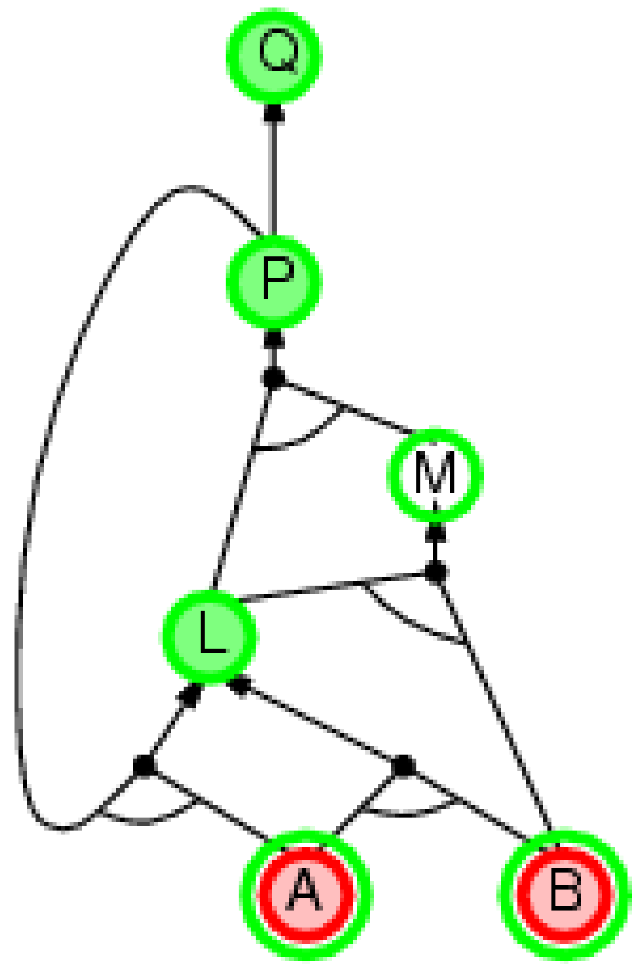
Backward chaining example



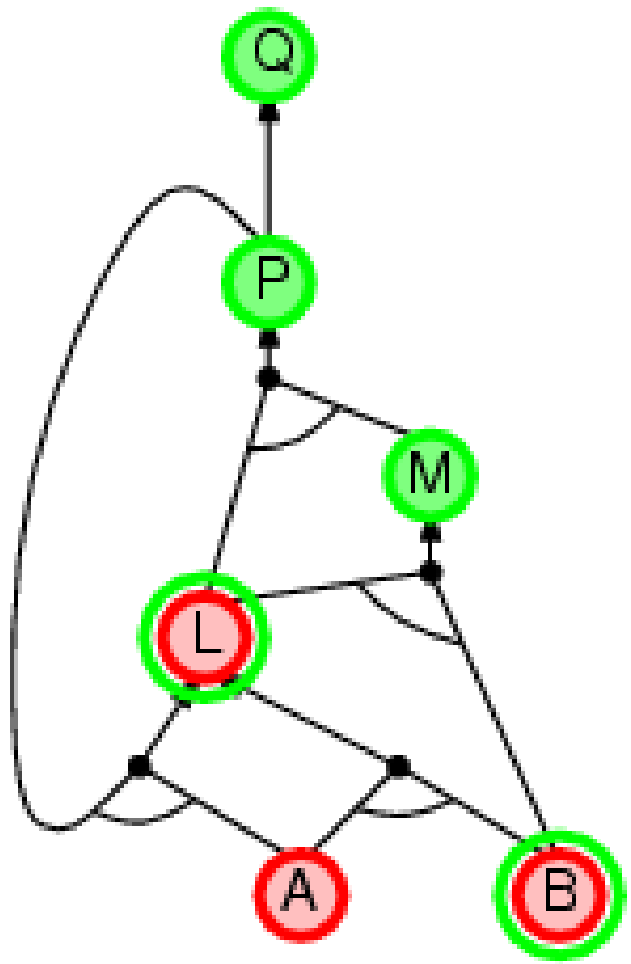
Backward chaining example



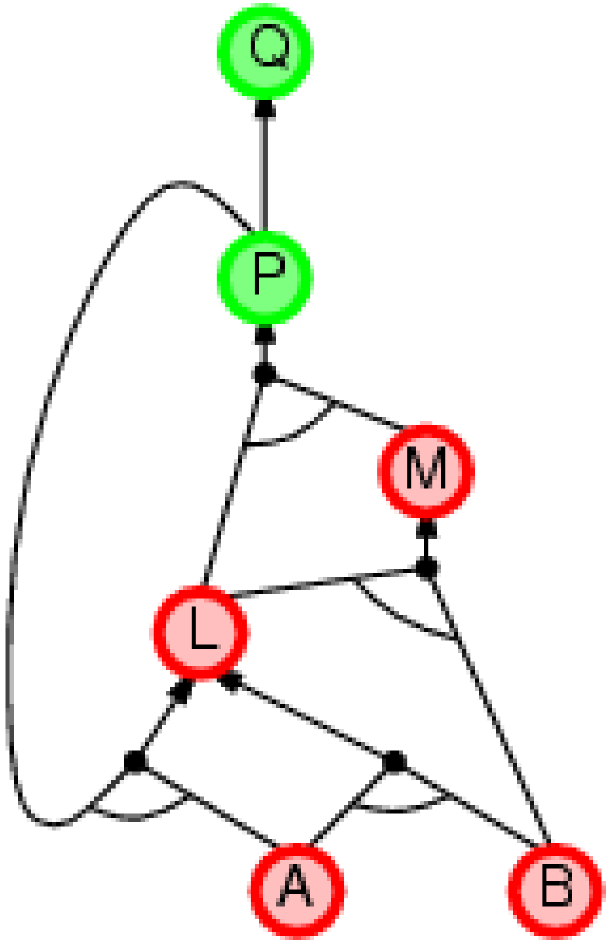
Backward chaining example



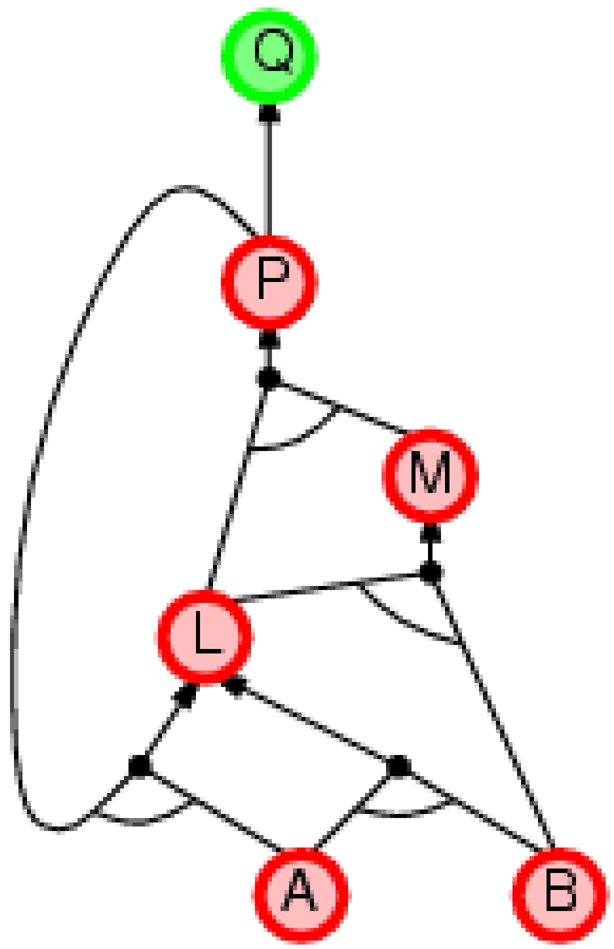
Backward chaining example



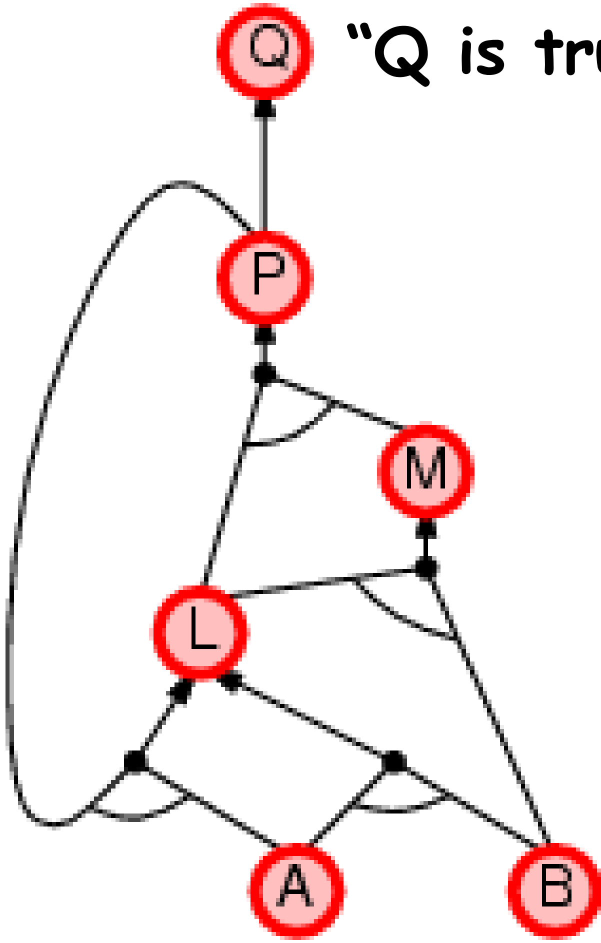
Backward chaining example



Backward chaining example



Backward chaining example



"Q is true"

Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  e.g., object recognition, routine decisions

- FC may do lots of work that is irrelevant to the goal

- BC is goal-driven, appropriate for problem-solving,
  e.g., How do I get an A in this class?
  e.g., What is my best exit strategy out of the classroom?
  e.g., How can I impress my date tonight?

- Complexity of BC *can be* much less than linear in size of KB

Recall: Inference by Model Checking

Complete search algorithms

Truth table enumeration: Recursive depth-first enumeration of assignments to all symbols (*TT-entails*)

Heuristic search

DPLL algorithm (Davis, Putnam, Logemann, Loveland): Recursive depth-first enumeration of possible models *with heuristics* (see textbook if interested)

Incomplete *local search* algorithms

`WalkSAT` algorithm for checking satisfiability

Why Satisfiability?

Can't get
¬satisfaction

Why Satisfiability?

- Recall: $KB \models \alpha$ iff $KB \wedge \neg\alpha$ is unsatisfiable
  - Equivalent to proving sentence α by contradiction

- Thus, algorithms for satisfiability can be used for inference (entailment)

- However, determining if a sentence is satisfiable or not (the SAT problem) is NP-complete

  Finding a fast algorithm for SAT automatically yields fast algorithms for hundreds of difficult (NP-complete) problems

Satisfiability Examples

E.g. 2-CNF sentences (2 literals per clause):

$(\neg A \vee \neg B) \wedge (A \vee B) \wedge (A \vee \neg B)$

Satisfiable?

Yes (e.g., A = true, B = false)

$(\neg A \vee \neg B) \wedge (A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$

Satisfiable?

No

The WalkSAT algorithm

- Local search algorithm
  - Incomplete: may not always find a satisfying assignment even if one exists
- Evaluation function? ("fitness" function)
  = Number of satisfied clauses
  - WalkSAT tries to maximize this function
- Balance between greediness and randomness
  - Each iteration:
  - Randomly select a symbol for flipping (T to F or F to T)
  - OR select symbol that maximizes # satisfied clauses

The `WalkSAT` algorithm

function WALKSAT(*clauses*, *p*, *max-flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move
    max-flips, number of flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for i = 1 **to** *max-flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability p flip the value in *model* of a randomly selected symbol
      from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

Greed **Randomness**

Hard Satisfiability Problems

Consider random 3-CNF sentences. e.g.,

$(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor A) \land (A \lor \neg D \lor B) \land (B \lor D \lor \neg C)$

Satisfiable?

(Yes, e.g., $A = B = C =$ *true*)

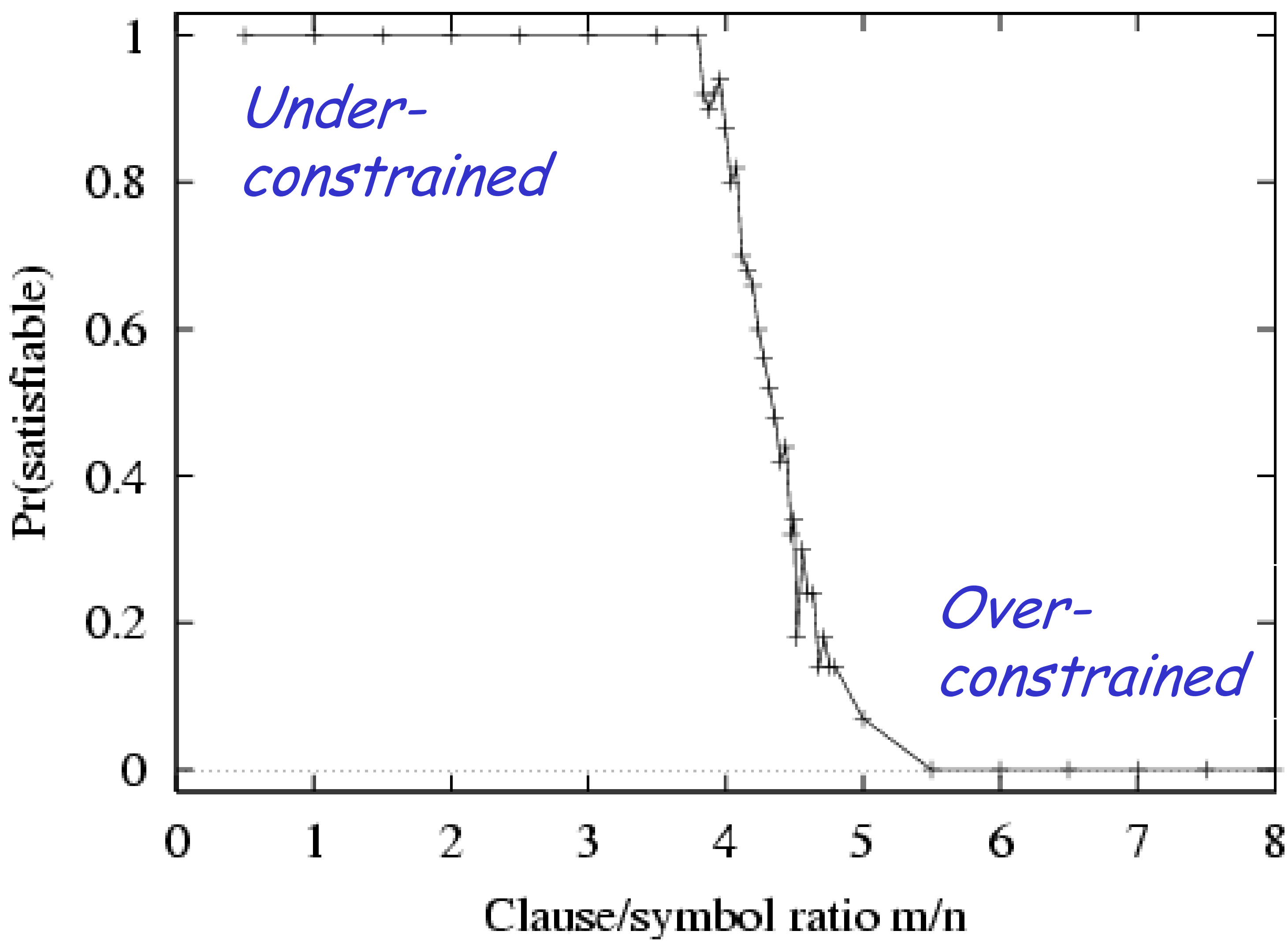
m = number of clauses (Here 5)

n = number of symbols (Here 4 – A, B, C, D)

m/n = 1.25 (enough symbols, usually satisfiable)

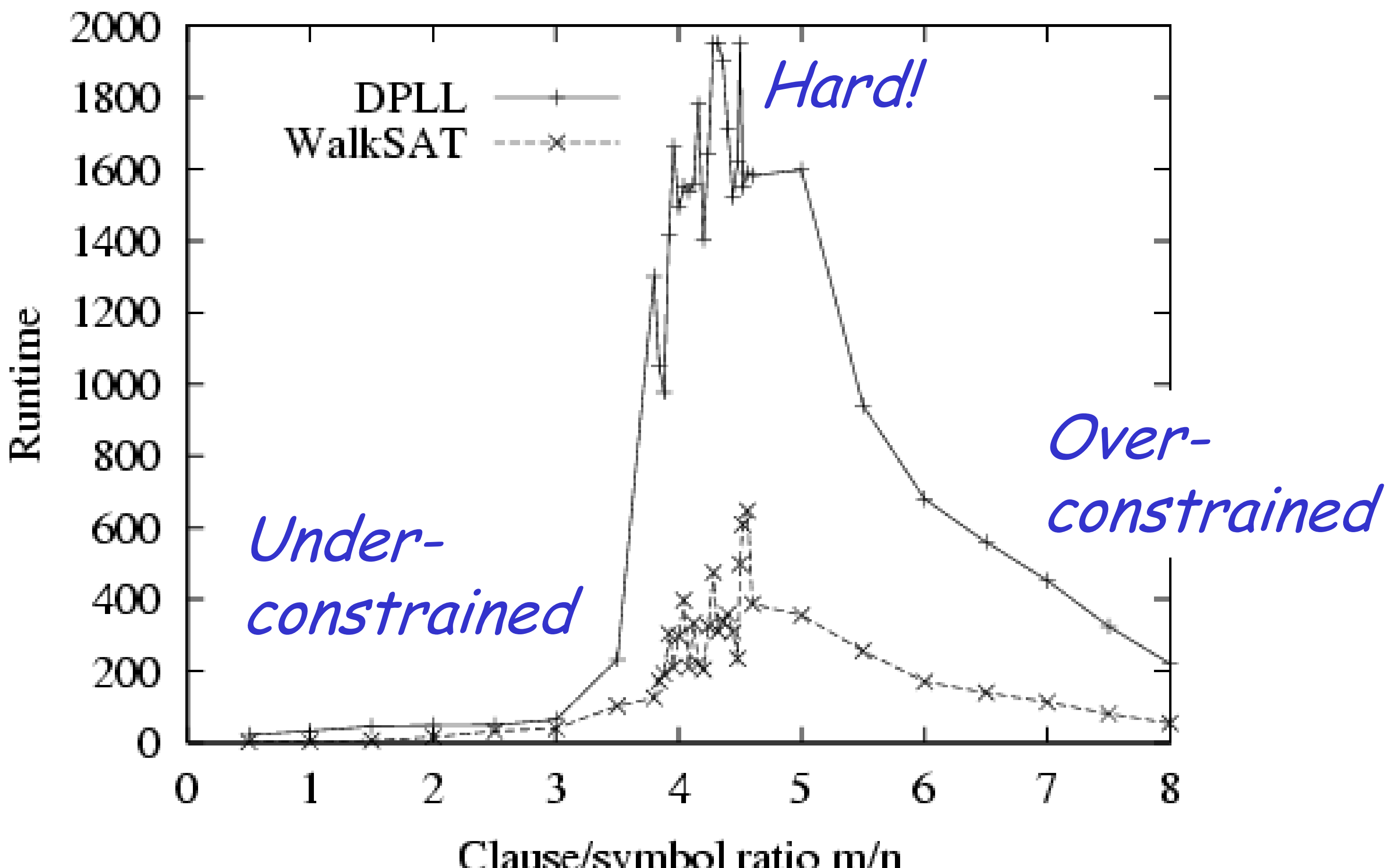
Hard instances of SAT seem to cluster near m/n = 4.3 (critical point)

Hard Satisfiability Problems

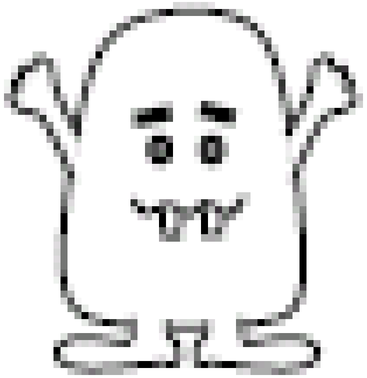


Hard Satisfiability Problems

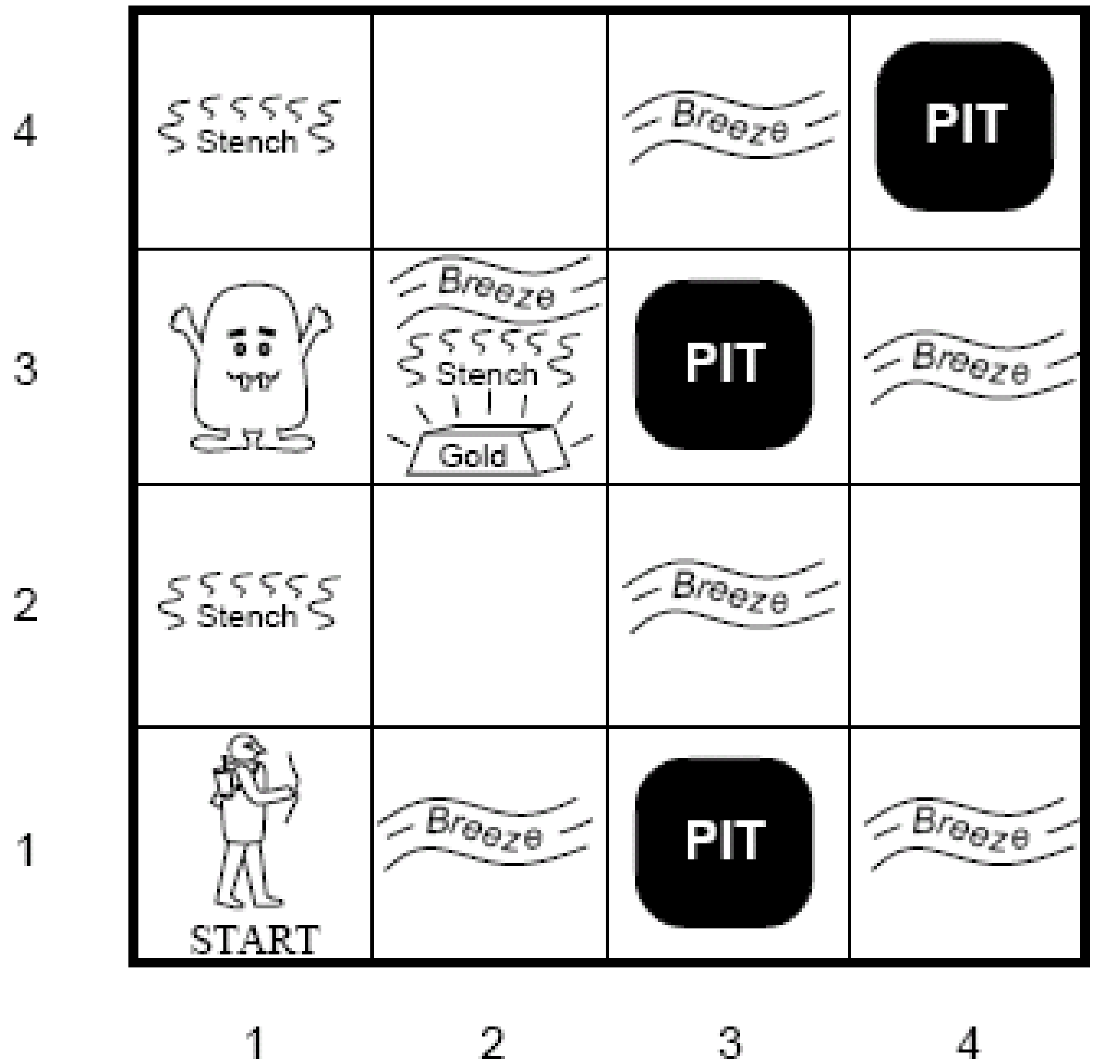
Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$



What about me?



Wumpus World



Putting it all together: Logical Wumpus Agents

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
For x = 1, 2, 3, 4 and y = 1, 2, 3, 4, add (with appropriate boundary conditions):

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$ At least 1 wumpus
$\neg(W_{1,1} \wedge W_{1,2})$
$\neg(W_{1,1} \wedge W_{1,3})$ At most 1 wumpus

...

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences!

Limitations of propositional logic

- KB contains "physics" sentences for every single square

- For every time step *t* and every location [*x*,*y*], we need to add to the KB "physics" rules such as:

$$L^{t}_{x,y} \wedge FacingRight^{t} \wedge Forward^{t} \Rightarrow L^{t+1}_{x+1,y}$$

- Rapid proliferation of sentences…

What we'd like is a way to talk about *objects* and *groups* of objects, and to define relationships between them.

Enter: First-order logic
(aka "predicate logic")

Next Time

- First-Order Logic
- To Do:
  - Project #2
  - Read chapter 8