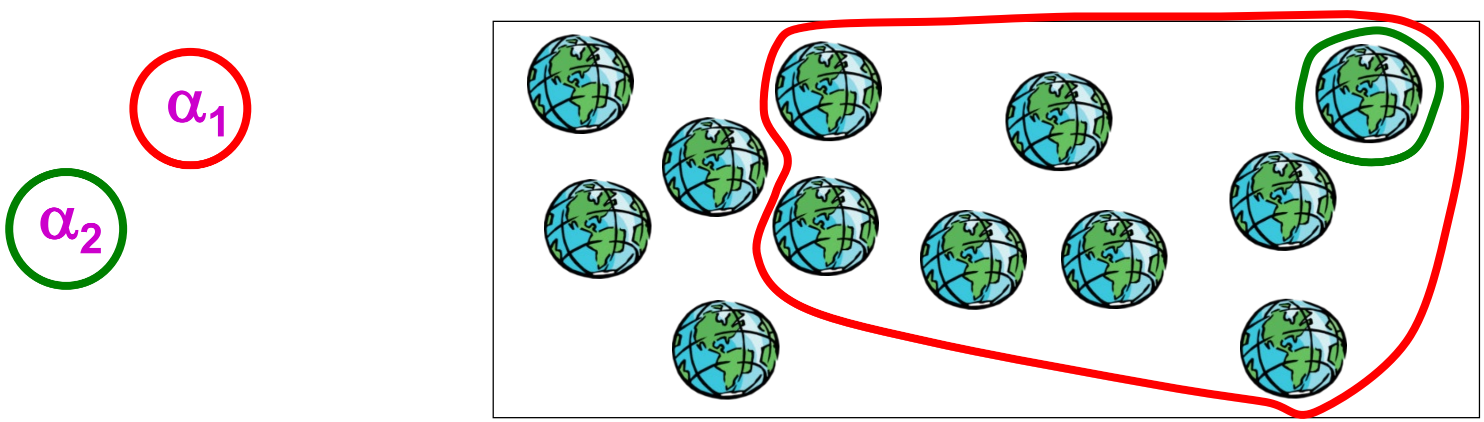
Inference: entailment

- Entailment: $\alpha \models \beta$ ("α entails β" or "β follows from α") means in every world where α is true, β is also true
  - i.e., the α-worlds are a subset of the β-worlds [models(α) $\subseteq$ models(β)]
- In the example, $\alpha 2 \models \alpha 1$



Inference: proof

- A proof (α |- β) is a demonstration of entailment from α to β
  - Method 1: model checking
    - Truth table enumeration to check if models(α) $\subseteq$ models(β)
    - Time complexity always exponential in n ☹

P1	**P2**	**…**	**Pn**	α	β
F	F	…	F	F	T
F	F	…	T	T	T
		……			
T	T	…	F	T	T
T	T	…	T	F	F

Inference: proof

- A proof (α |- β) is a demonstration of entailment from α to β
  - Method 2: application of inference rules
    - Search for a finite sequence of sentences each of which is an axiom or follows from the preceding sentences by a rule of inference
    - Axiom: a sentence known to be true
    - Rule of inference: a function that takes one or more sentences (premises) and returns a sentence (conclusion)

Inference: soundness & completeness

- Sound inference
  - everything that can be proved is in fact entailed
- Complete inference
  - everything that is entailed can be proved

- Method 1 (enumeration) is obviously sound and complete
- For method 2 (applying inference rules), it is much less obvious
  - Example: arithmetic is found to be not complete! (Gödel's theorem, 1931)

Quiz

- What's the connection between complete inference algorithms and complete search algorithms?
- Answer 1: they both have the words "complete…algorithm"
- Answer 2: Formulate inference α |- β as a search problem
  - Initial state: KB contains α
  - Actions: apply any inference rule that matches KB, add conclusion
  - Goal test: KB contains β

  Hence any complete search algorithm can be used to produce a complete inference algorithm

Resolution: an inference rule in PL

- Conjunctive Normal Form (CNF)
  - conjunction of <u>disjunctions of literals</u> (clauses)
  - Ex
    - $(A \lor \neg B) \land (B \lor \neg C \lor \neg D)$
    - $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Apply distributivity law ($\wedge$ over $\vee$) and flatten:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution: an inference rule in PL

- Resolution inference rule (for CNF):

  Suppose l_i is $\neg m_j$

$$\frac{l_1 \vee \ldots \vee l_k, \qquad\qquad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

  Examples:

$$\frac{P_{1,3} \vee P_{2,2}, \qquad P_{2,3} \vee \neg P_{2,2}}{P_{1,3} \vee P_{2,3}} \qquad\qquad \frac{P_1, \neg P_1}{\{\}}$$

- Resolution is sound and complete for propositional logic

Resolution algorithm

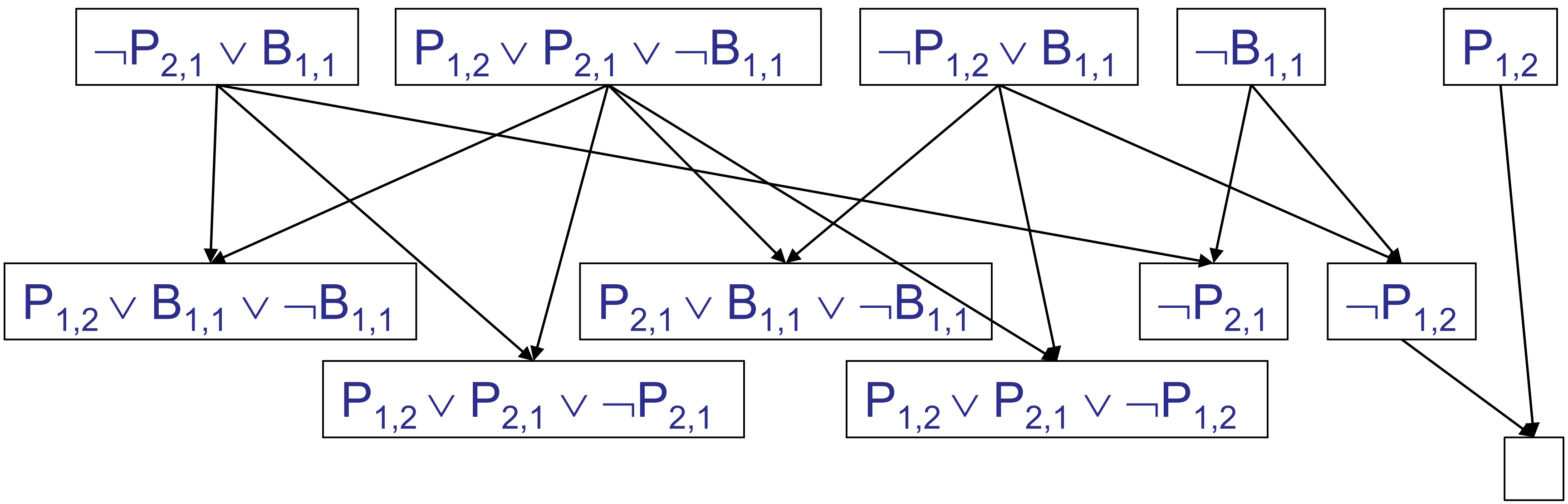
- The best way to prove $KB \models \alpha$?
  - Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ is unsatisfiable

1. Convert $KB \wedge \neg\alpha$ to CNF
2. Repeatedly apply the resolution rule to add new clauses, until one of the two things happens
   a) Two clauses resolve to yield the empty clause, in which case KB entails α
   b) There is no new clause that can be added, in which case KB does not entail α

Resolution example

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$

$\alpha = \neg P_{1,2}$



Horn Logic

Horn logic

- Inference in propositional logic is in general NP-complete!
- Solution: a subset of propositional logic that supports efficient inference

Expressiveness vs. Inference difficulty!!

- Horn logic: only (strict) Horn clauses are allowed
  - A Horn clause has the form:

    $P1 \wedge P2 \wedge P3 \ldots \wedge Pn \Rightarrow Q$

    or alternatively

    $\neg P1 \vee \neg P2 \vee \neg P3 \ldots \vee \neg Pn \vee Q$

    where Ps and Q are non-negated proposition symbols (atoms)
  - n can be zero, i.e., the clause contains a single atom

Inference in Horn logic

- Modus Ponens

$$\frac{\alpha 1, \ldots, \alpha n, \quad \overbrace{\alpha 1 \wedge \ldots \wedge \alpha n}^{\text{premises}} \Rightarrow \overset{\text{conclusion}}{\beta}}{\beta}$$

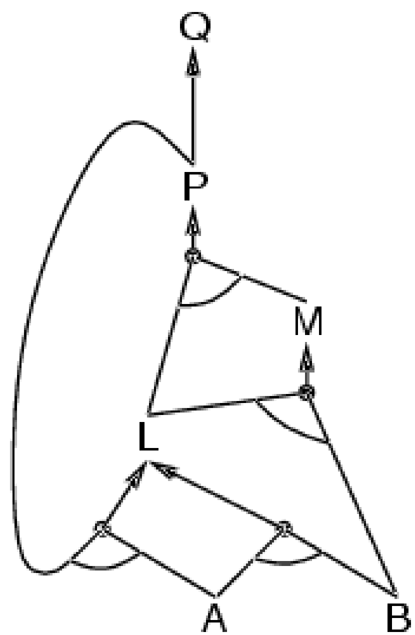
  - Modus Ponens is sound and complete for Horn logic

- Inference algorithms (for Horn logic)
  - Forward chaining, backward chaining
  - These algorithms are very natural and run in linear time

Forward chaining

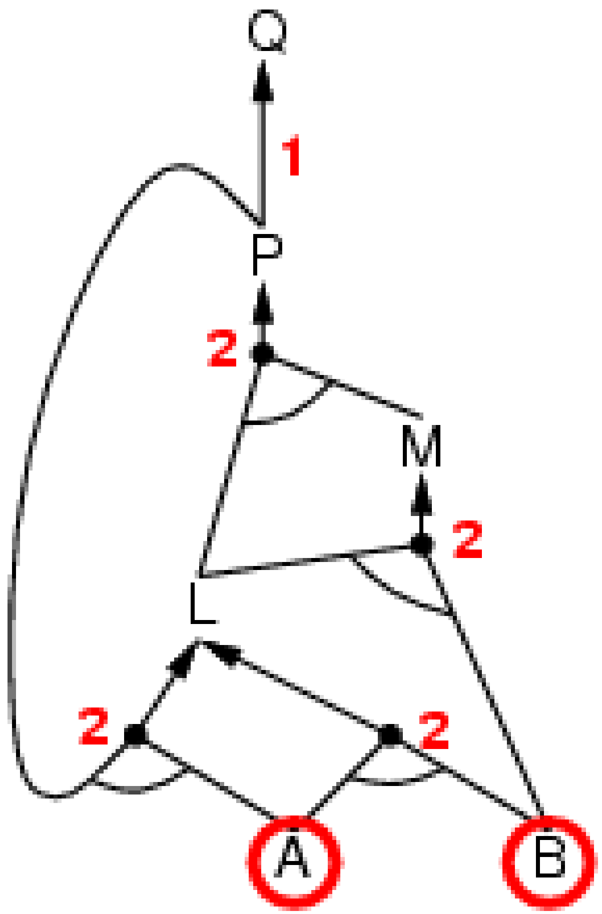
- Idea: to prove KB |= Q
  - Add new clauses into the KB by applying Modus Ponens, until Q is added

$$
\begin{aligned}
P &\Rightarrow Q \\
L \wedge M &\Rightarrow P \\
B \wedge L &\Rightarrow M \\
A \wedge P &\Rightarrow L \\
A \wedge B &\Rightarrow L \\
A & \\
B &
\end{aligned}
$$



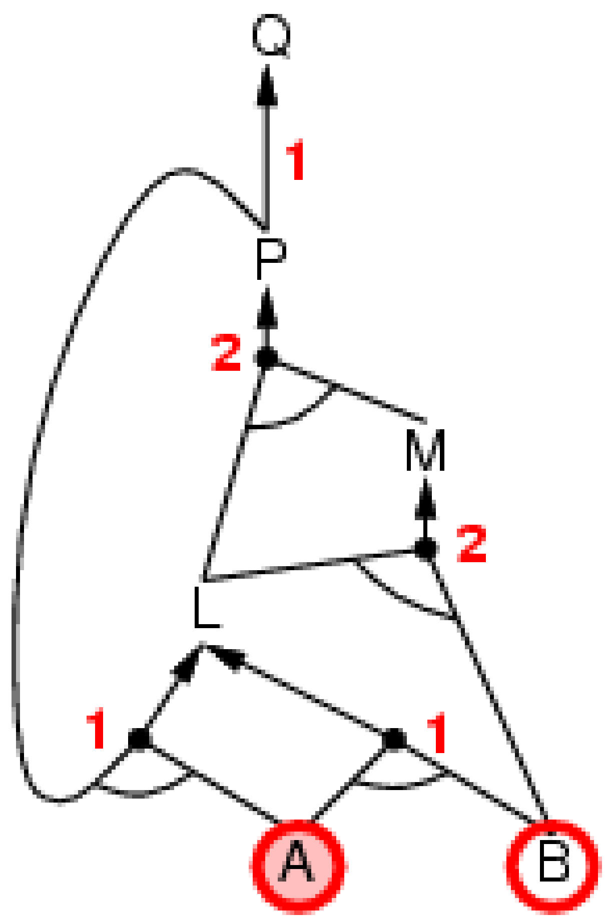
Forward chaining example

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



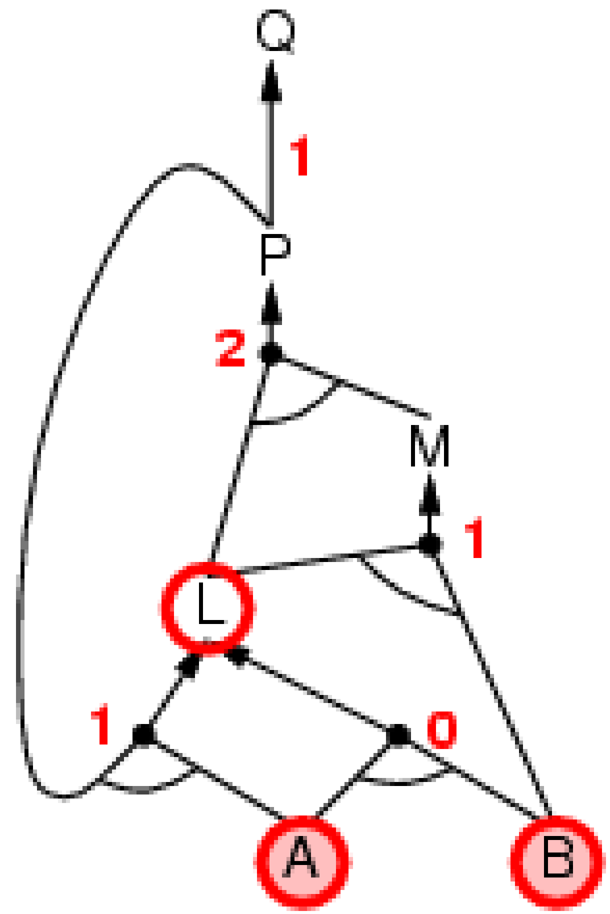
Forward chaining example

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



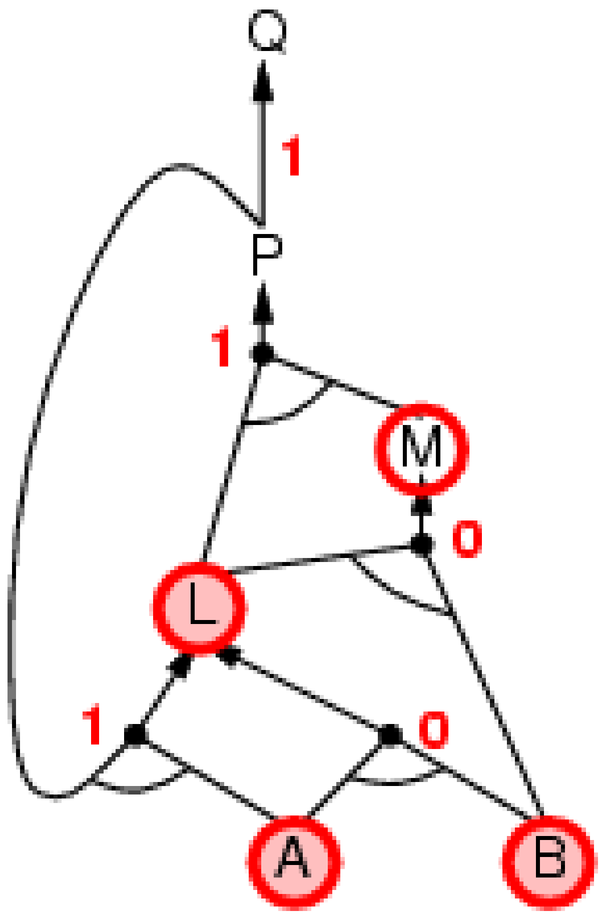
Forward chaining example

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



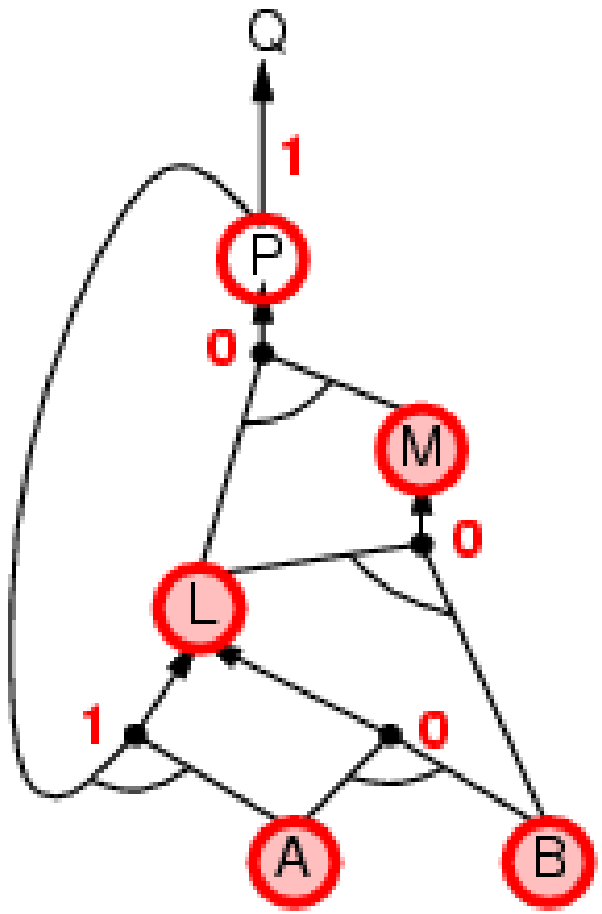
Forward chaining example

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



Forward chaining example

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



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

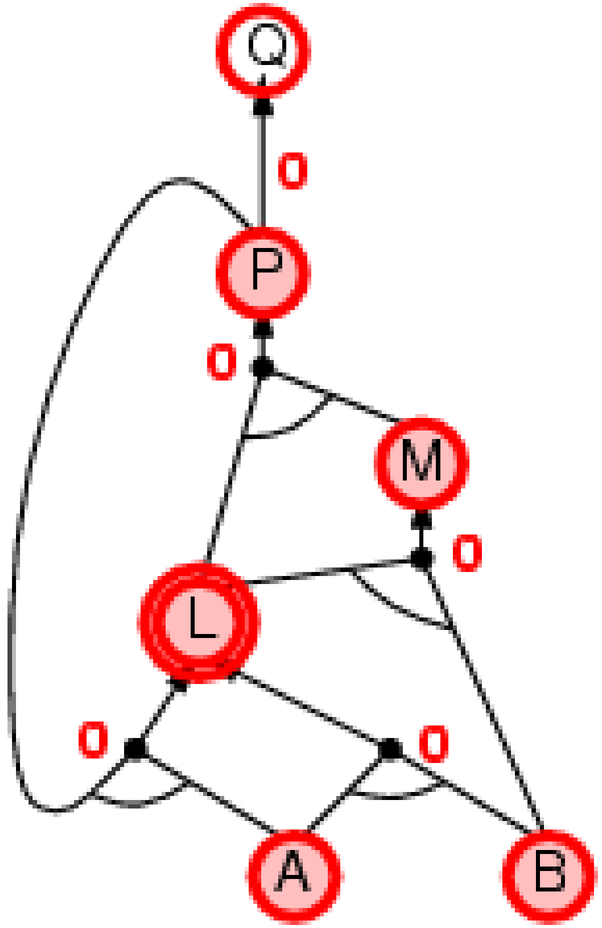
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

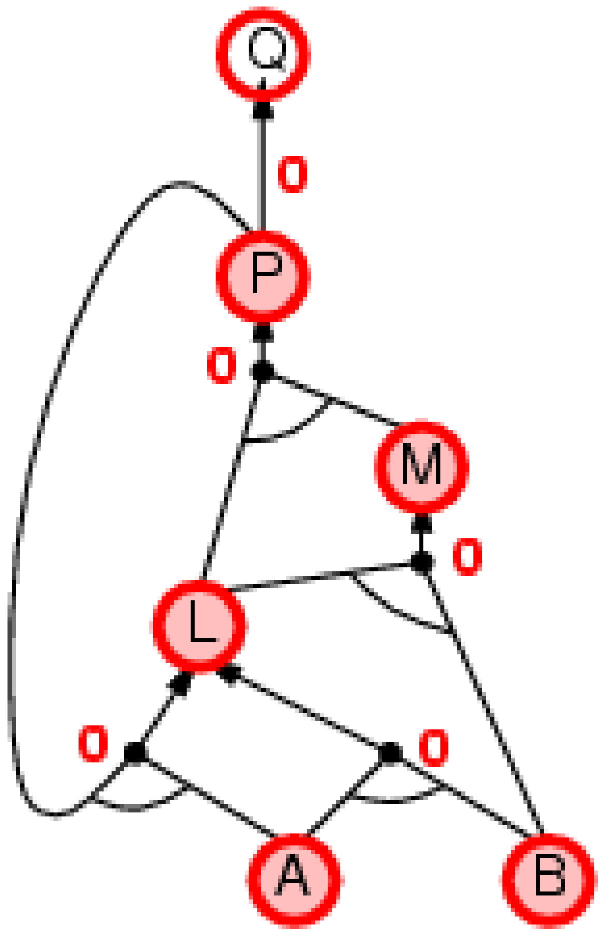
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

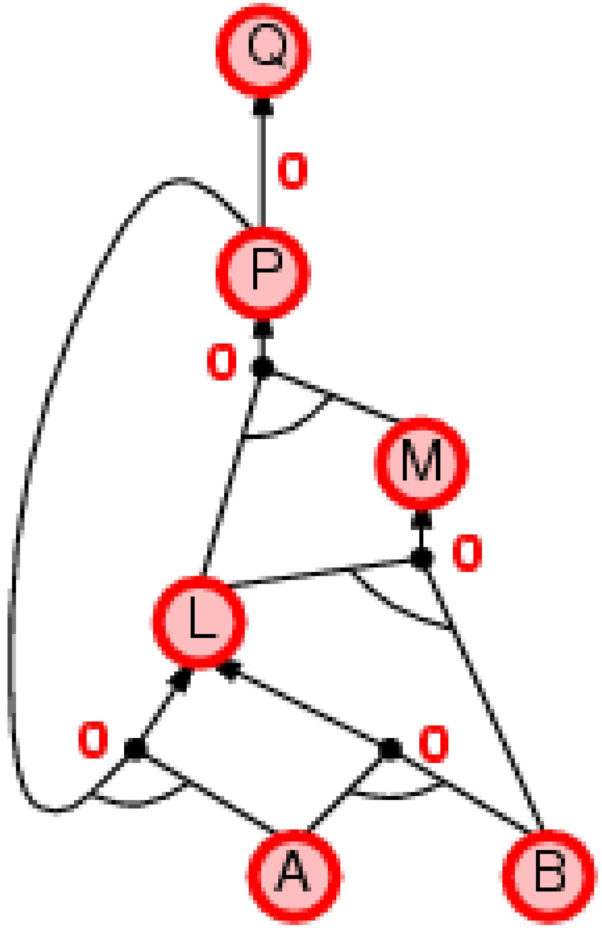
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

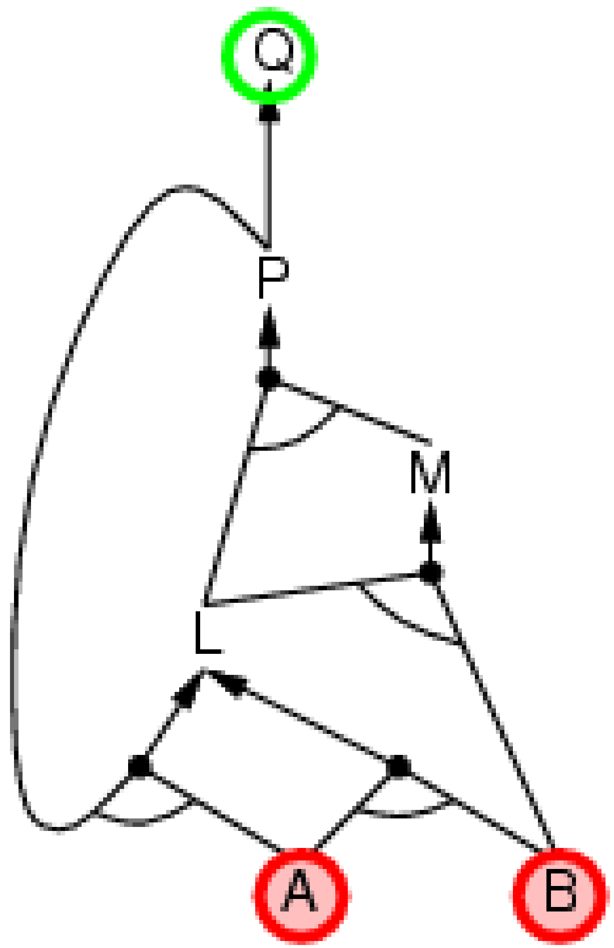
B



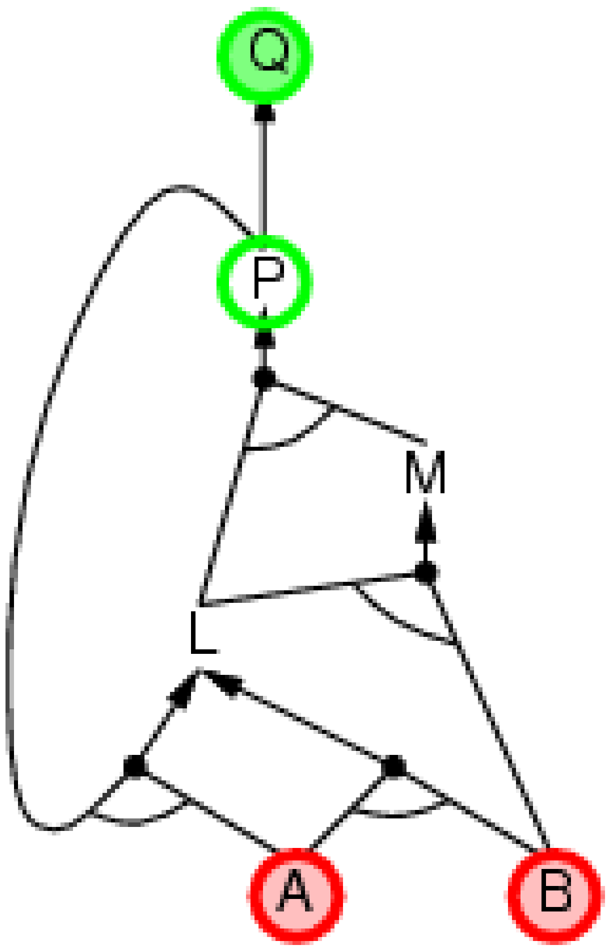
Backward chaining

- 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
- Avoid loops: check if new subgoal is already on the goal stack
- Avoid repeated work: check if new subgoal
  1. has already been proved true, or
  2. has already failed

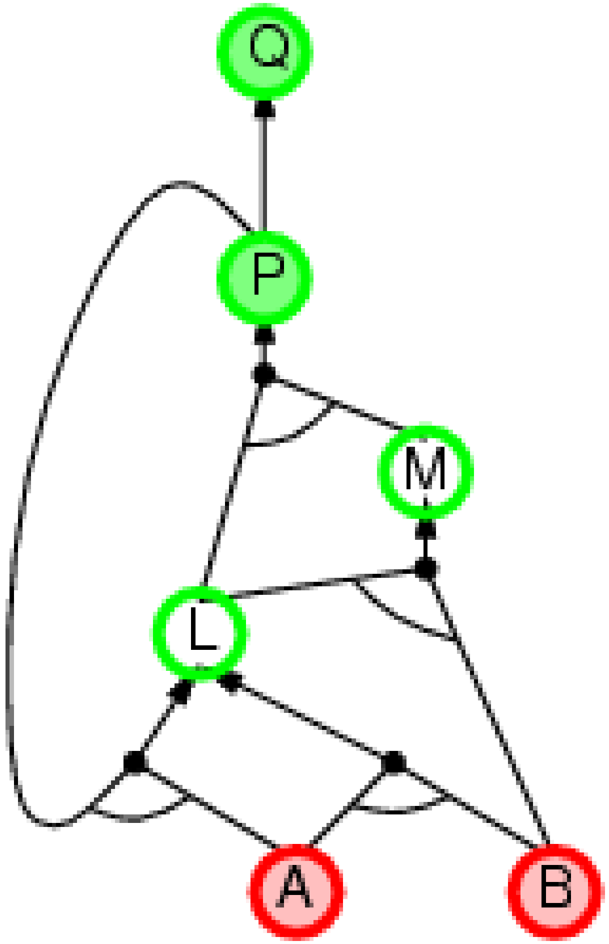
Backward chaining example



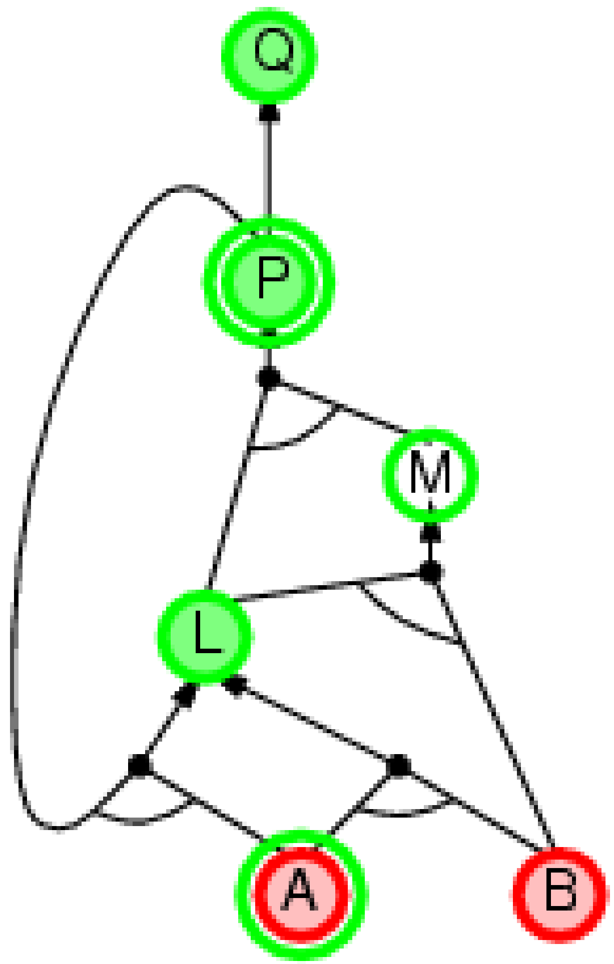
Backward chaining example



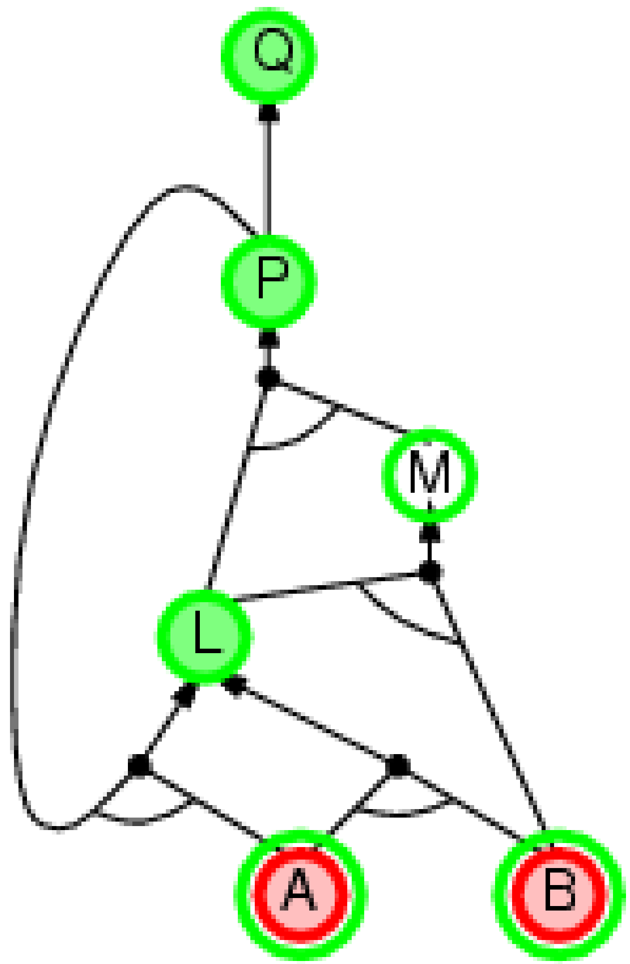
Backward chaining example



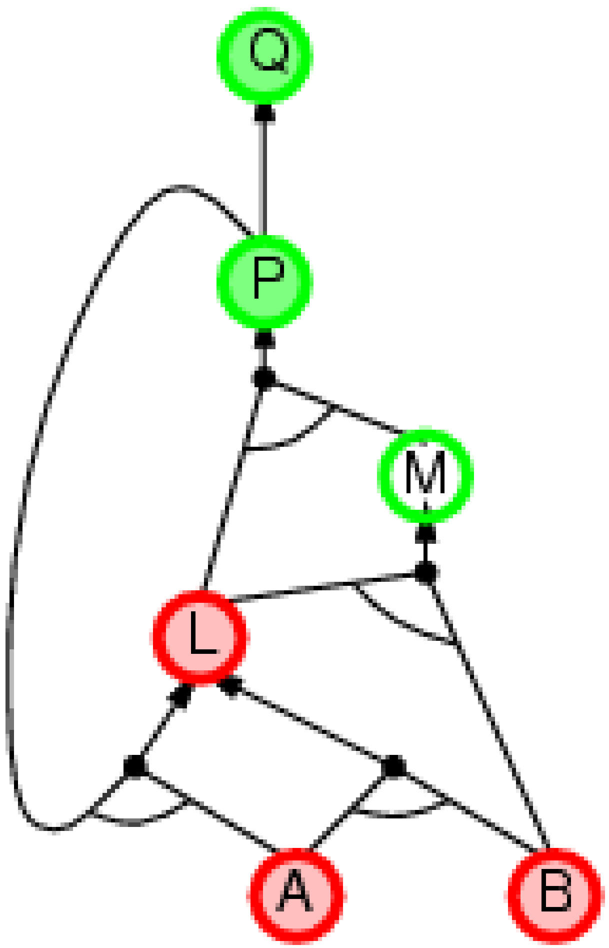
Backward chaining example



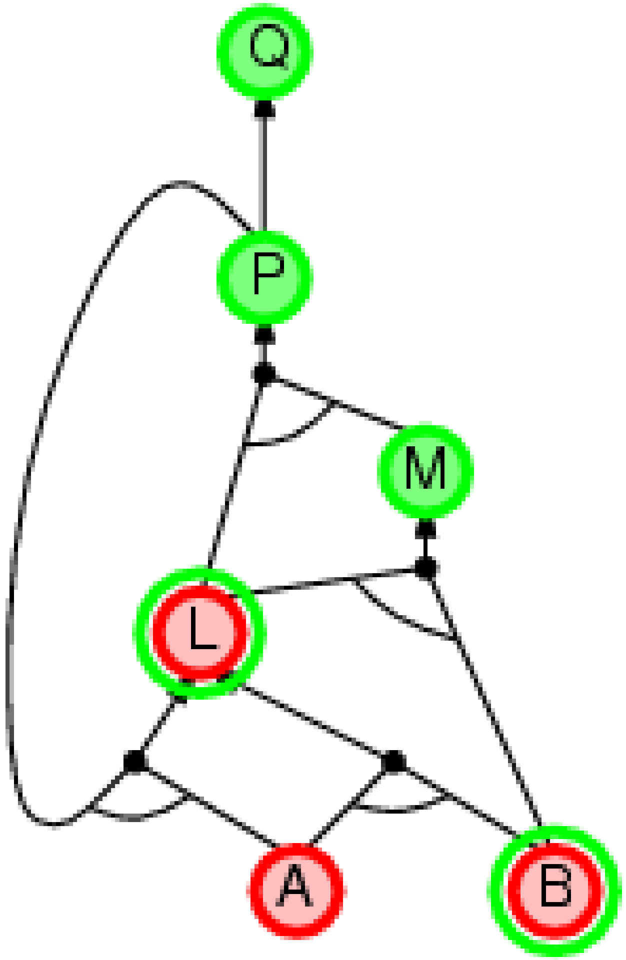
Backward chaining example



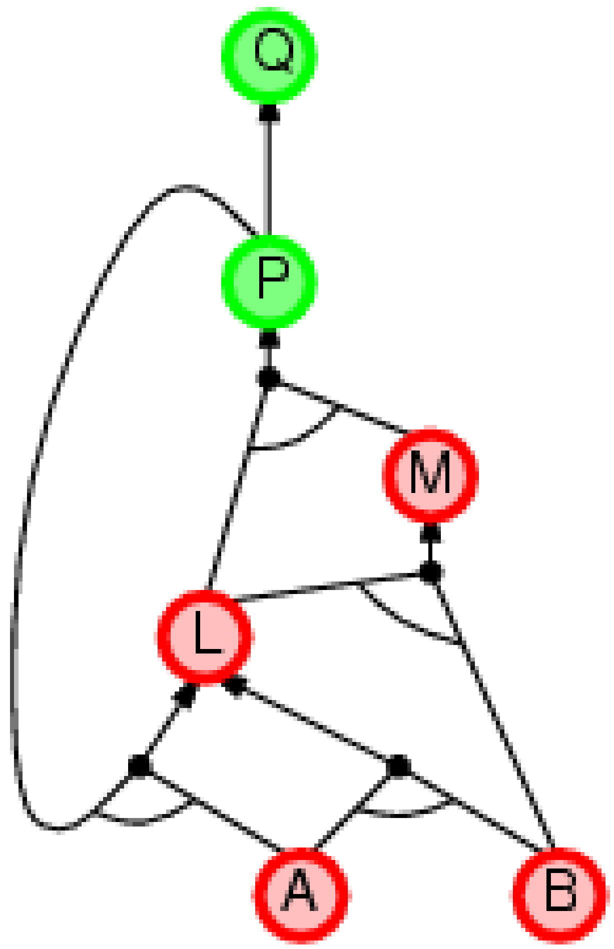
Backward chaining example



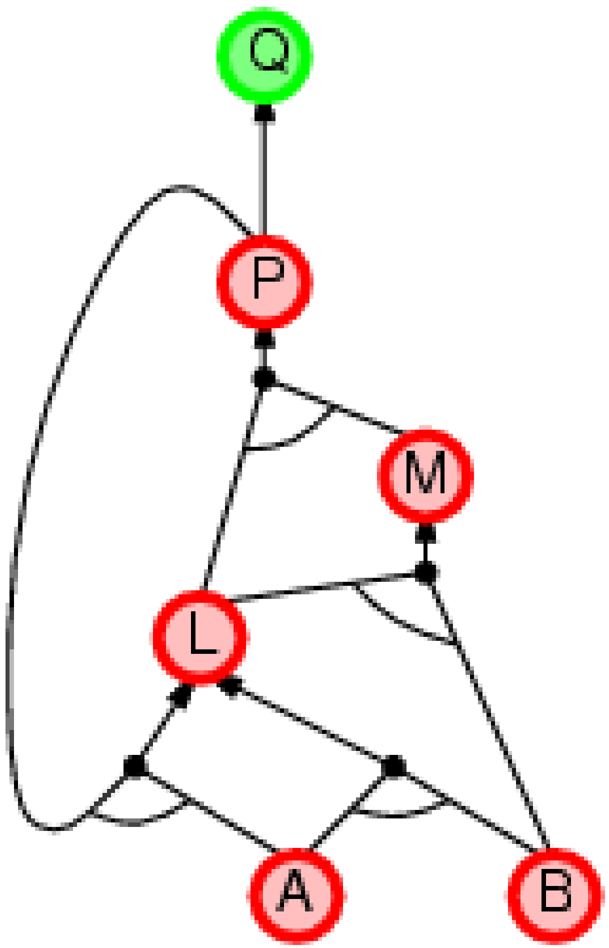
Backward chaining example



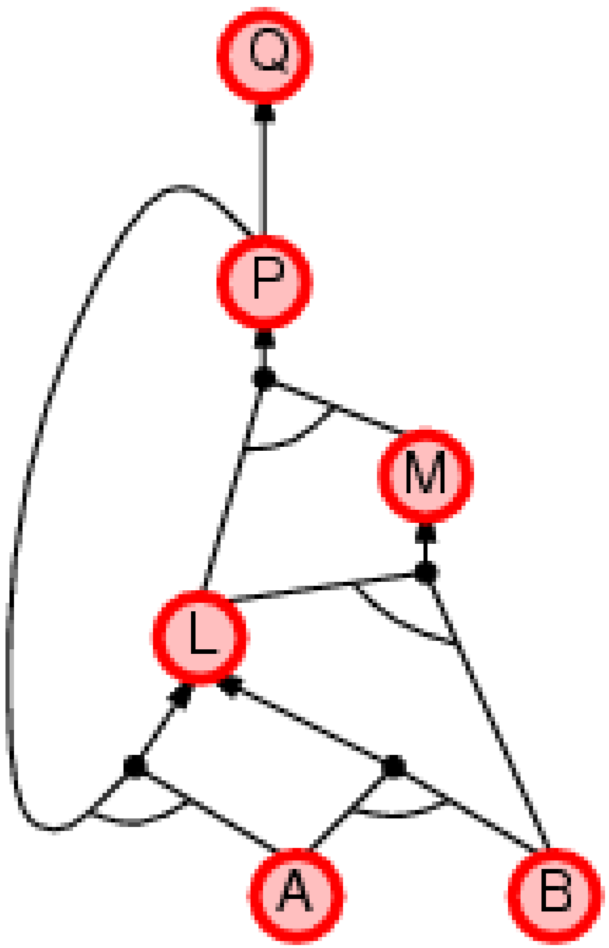
Backward chaining example



Backward chaining example



Backward chaining example

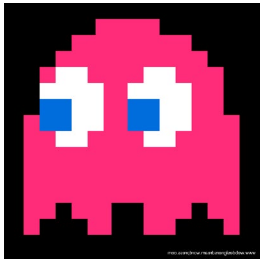


Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing,
  - e.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal

- BC is goal-driven, appropriate for problem-solving,
  - e.g., Where are my keys? How do I get into a PhD program?
  - Complexity of BC can be much less than linear in size of KB

A Logical Pacman



Partially observable Pacman

- Pacman perceives just the local walls/gaps
- Formulation: what proposition symbols do we need?
  - Pacman's location
    - At_1,1_0 (Pacman is at [1,1] at time 0)   At_3,3_1   etc
  - Wall locations
    - Wall_0,0   Wall_0,1   etc
  - Percepts
    - Blocked_W_0 (blocked by wall to my West at time 0) etc.
  - Actions
    - W_0 (Pacman moves West at time 0)   E_0   etc.
- NxN world for T time steps => $N^2T + N^2 + 4T + 4T = O(N^2T)$ symbols

Sensor model

- State facts about how Pacman's percepts arise…
- Pacman perceives a wall to the West at time t if he is in x,y and there is a wall at x-1,y

$$((At\_1,1\_0 \wedge Wall\_0,1) \vee$$
$$(At\_1,2\_0 \wedge Wall\_0,2) \vee$$
$$(At\_1,3\_0 \wedge Wall\_0,3) \vee \ldots.\ ) \Rightarrow Blocked\_W\_0$$

Sensor model

- State facts about how Pacman's percepts arise…
- Pacman perceives a wall to the West at time t ***if and only if*** he is in x,y and there is a wall at x-1,y

$$\begin{aligned}\text{Blocked\_W\_0} \Leftrightarrow ((&\text{At\_1,1\_0} \wedge \text{Wall\_0,1}) \vee \\ &(\text{At\_1,2\_0} \wedge \text{Wall\_0,2}) \vee \\ &(\text{At\_1,3\_0} \wedge \text{Wall\_0,3}) \vee \ldots.)\end{aligned}$$

Transition model

- How does each state symbol at each time get its value?
  - E.g., should At_3,3_17 be T or F?
- A state symbol gets its value according to a successor-state axiom

$$X_t \Leftrightarrow [X_{t-1} \wedge \neg(\text{some action}_{t-1} \text{ made it false})] \vee$$
$$[\neg X_{t-1} \wedge (\text{some action}_{t-1} \text{ made it true})]$$

- For Pacman location:

$$\text{At\_3,3\_17} \Leftrightarrow$$
$$[\text{At\_3,3\_16} \wedge \neg((\neg\text{Wall\_3,4} \wedge \text{N\_16}) \vee (\neg\text{Wall\_4,3} \wedge \text{E\_16}) \vee \ldots)]$$
$$\vee\ [\neg\text{At\_3,3\_16} \wedge ((\text{At\_3,2\_16} \wedge \neg\text{Wall\_3,3} \wedge \text{N\_16}) \vee$$
$$(\text{At\_2,3\_16} \wedge \neg\text{Wall\_3,3} \wedge \text{E\_16}) \vee \ldots)]$$

Initial state

- The agent may know its initial location:

  At_1,1_0

- Or, it may not:

  $At\_1,1\_0 \vee At\_1,2\_0 \vee At\_1,3\_0 \vee \ldots \vee At\_3,3\_0$

Domain constraint

- Pacman cannot be in two places at once!

  $\neg(At\_1,1\_0 \wedge At\_1,2\_0) \wedge \neg(At\_1,1\_0 \wedge At\_1,3\_0) \wedge \ldots$

  $\neg(At\_1,1\_1 \wedge At\_1,2\_1) \wedge \neg(At\_1,1\_1 \wedge At\_1,3\_1) \wedge \ldots$

  …
- Pacman cannot take two actions at the same time!

  $\neg(E\_0 \wedge S\_0) \wedge \neg(E\_0 \wedge W\_0) \wedge \ldots$

  $\neg(E\_1 \wedge S\_1) \wedge \neg(E\_1 \wedge W\_1) \wedge \ldots$

  …
- Pacman cannot go into a wall!

  $At\_1,1\_0 \wedge N\_0 \Rightarrow \neg Wall\_1,2$

  …

Planning as satisfiability

- SAT solver
  - Input: a logic expression
  - Output: a model (true/false assignments to symbols) that satisfies the expression if such a model exists
- Can we use it to make plans for Pacman (e.g., to move to a goal position)?
  - For T = 1 to infinity, set up the KB as follows and run SAT solver:
    - Initial state, domain constraints, sensor & transition model sentences up to time T
    - Goal is true at time T
  - If a model is returned, extract a plan from action assignment

Planning as satisfiability

- Q: Isn’t this a search problem? Any advantage of using logic?
- A: We can use logic to solve not only search problems, but any problems that are representable using the logic.

Logic programming

- Ordinary programming
  - Identify problem
  - Assemble information
  - Figure out solution
  - Encode solution
  - Encode problem instance as data
  - Apply program to data
- Logic programming
  - Identify problem
  - Assemble information
  - <coffee break> ☺
  - Encode info in KB
  - Encode problem instance as facts
  - Ask queries (run SAT solver)

Summary

- Logic
  - Logical AI applies inference to a knowledge base to derive new information
- Propositional logic
  - Syntax
  - Semantics
  - Inference (resolution)
- Horn logic
  - Inference (forward/backward chaining)
- Application of logic to Pacman