PURDUE CS47100

INTRODUCTION TO AI

ANNOUNCEMENTS

- Midterm exam is coming up in about two weeks!
  - October 20, Thursday, 8-10pm, PHYS 112
  - Content to be included: Systematic search, local search, adversarial search, CSP, propositional logic, first-order logic
  - Closed-book, closed-notes
  - If you have accessibility needs, please contact DRC to reserve a testing center!

RECAP: PROPOSITIONAL LOGIC

- **Logics** are formal languages for representing information such that conclusions can be drawn
- **Atomic sentences** are composed of proposition symbols, **complex sentences** are built up from atomic sentences using negation, conjunction, disjunction, implications, and biconditional
- **Knowledge base** is a set of sentences in a representation language
- **Entailment**: KB $\models \alpha$ means for every possible world that KB is true, the sentence α is also true
- **Inference** is the procedure/algorithm to derive new conclusions from KB
- Desirable properties of inference algorithms
  - **Soundness**: Only entailed sentences are derived
  - **Completeness**: Any entailed sentence can be derived

HOW TO DETERMINE ENTAILMENT?

- Methods can be divided into (roughly) two kinds:
  - **Theorem proving: Application of inference rules**
    - Legitimate (sound) generation of new sentences from old
    - Proof = a sequence of inference rule applications
      Can use inference rules as operators in a standard search algorithm
    - Typically require transformation of sentences into a normal form
  - **Model checking**
    - Truth table enumeration (always exponential in n)
    - Improved backtracking, e.g., Davis-Putnam-Logemann-Loveland (DPLL)
    - Heuristic search in model space (sound but incomplete), e.g., min-conflicts-like hill-climbing algorithms

HOW TO DETERMINE ENTAILMENT?

- Methods can be divided into (roughly) two kinds:
  - **Theorem proving: Application of inference rules**
    - Legitimate (sound) generation of new sentences from old
    - Proof = a sequence of inference rule applications
      Can use inference rules as operators in a standard search algorithm
    - Typically require transformation of sentences into a normal form
  - **Model checking**
    - Truth table enumeration (always exponential in n)
    - Improved backtracking, e.g., Davis-Putnam-Logemann-Loveland (DPLL)
    - Heuristic search in model space (sound but incomplete), e.g., min-conflicts-like hill-climbing algorithms

LOGICAL EQUIVALENCE

- Two sentences are **logically equivalent** iff they are true in the same set of models: $\alpha \equiv \beta$ iff $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

VALIDITY AND SATISFIABILITY

- A sentence is **valid** if it is true in **all** models (aka tautologies)
  - Example: True, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$
- A sentence is **satisfiable** if it is true in **some** model, e.g., Wumpus World example
- A sentence is **unsatisfiable** if it is true in **no** models, e.g., $A \wedge \neg A$
- Validity is connected to inference via the **Deduction Theorem**:
  - $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- Satisfiability is connected to inference via the following:
  - $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable

PROOF BY INFERENCE RULES

- Proof by applying inference rule to show (KB $\Rightarrow \alpha$) is valid
- Rule 1 (Modus Ponens): $$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$
- Rule 2 (And-Elimination): $$\frac{\alpha \wedge \beta}{\alpha}$$
- All logical equivalences can be used as inference rules
- **Monotonicity**: set of entailed sentences can only increase as information is added to the KB

$$\text{if } KB \models \alpha \text{ then } KB \wedge \beta \models \alpha$$

PROOF BY INFERENCE RULES: EXAMPLE

- KB = $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$
- Proof:
  - And elimination: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$, $\neg B_{1,1}$
  - Biconditional elimination: $B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})$, $P_{1,2} \vee P_{2,1} \Rightarrow B_{1,1}$
  - Contraposition: $\neg B_{1,1} \Rightarrow \neg (P_{1,2} \vee P_{2,1})$
  - Modus Ponens: $\neg (P_{1,2} \vee P_{2,1})$
  - De Morgan: $\neg P_{1,2} \wedge \neg P_{2,1}$
  - And elimination: $\neg P_{1,2}$

This is a search problem!

PROOF BY RESOLUTION

- Proof by showing (KB $\wedge \neg \alpha$) is unsatisfiable
- Sketch:
  - Negate the conclusion and and it to the KB, i.e., add $\neg \alpha$ to KB
  - Convert sentences in KB to **conjunctive normal form**
  - Repeatedly apply the **resolution inference rule** until contradiction is found

PROOF BY RESOLUTION: CONVERSION TO CNF

- Every sentence in propositional logic is logically equivalent to a conjunction of clauses (which are each disjunctions of literals)
  - e.g., $(\text{Literal}_1 \vee \text{Literal}_2 \vee \text{Literal}_3) \wedge (\text{Literal}_4 \vee \text{Literal}_5)$
  - Literal can be any symbol or its negation
  - Clauses: $\text{Literal}_1 \vee \text{Literal}_2 \vee \text{Literal}_3$, $\text{Literal}_4 \vee \text{Literal}_5$

CONVERSION TO CNF: EXAMPLE

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

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

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

CONVERSION TO CNF: EXAMPLE

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

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 $\neg\alpha \vee \beta$.

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

CONVERSION TO CNF: EXAMPLE

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

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:

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

CONVERSION TO CNF: EXAMPLE

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

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:

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

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

- **Resolution** inference rule (for CNF):

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

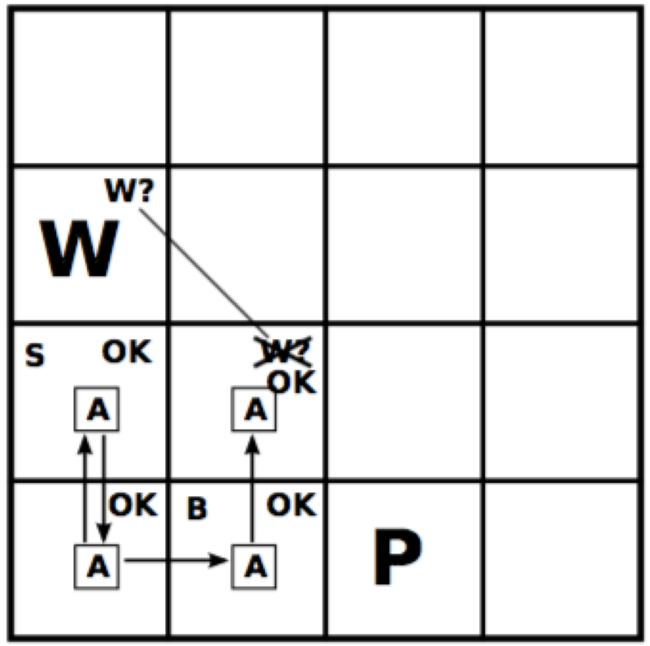
  where ℓ_i and m_j are ***complementary*** literals (i.e., one is the negation of the other)

- For example:

$$\frac{W_{1,3} \vee W_{2,2}, \quad \neg W_{2,2}}{W_{1,3}}$$

- Main strategy: Apply resolution and find contradiction, i.e., arrive at an empty clause

- Resolution is sound and complete for propositional logic



RESOLUTION EXAMPLE

- KB = $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$
- $\alpha = \neg P_{1,2}$

Prove by contradiction, show that KB ∧ ¬α is unsatisfiable

Resolve pairs
in first row

RESOLUTION ALGORITHM

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  $new \leftarrow \{\,\}$
  loop do
    for each pair of clauses C_i, C_j **in** $clauses$ **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if $resolvents$ contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

FORWARD AND BACKWARD CHAINING

- **Horn Form** (restricted): KB = *conjunction* of **Horn clauses**
  - Horn clause = proposition symbol; or (**conjunction** of symbols) $\Rightarrow$ symbol
    - E.g., C $\wedge$ (B $\Rightarrow$ A) $\wedge$ (C $\wedge$ D $\Rightarrow$ B)
- **Modus Ponens** (for Horn Form): complete for Horn KBs

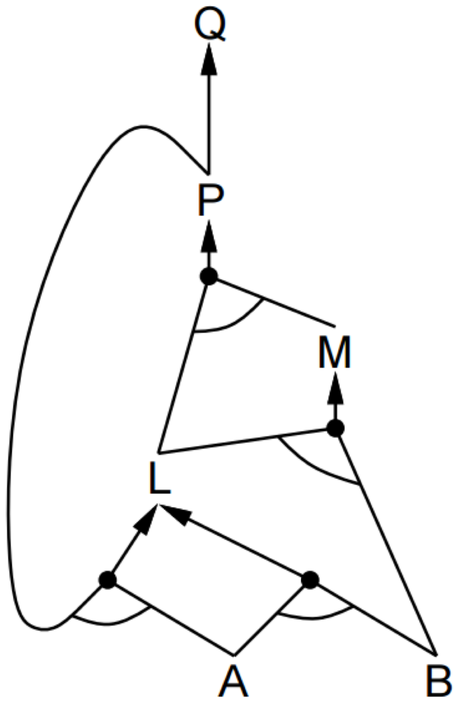
$$\frac{\alpha_1, \ldots \alpha_n, \quad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Proof can be done with **forward chaining** or **backward chaining**
- These algorithms are very natural and run in **linear** time

FORWARD CHAINING

- Idea: Consider each rule whose premises are satisfied in KB,
  - add its conclusion to the KB, until query is found

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



FORWARD CHAINING ALGORITHM

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
      p ← POP(agenda)
      if p = q then return true
      if inferred[p] = false then
          inferred[p] ← true
          for each clause c in KB where p is in c.PREMISE do
              decrement count[c]
              if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

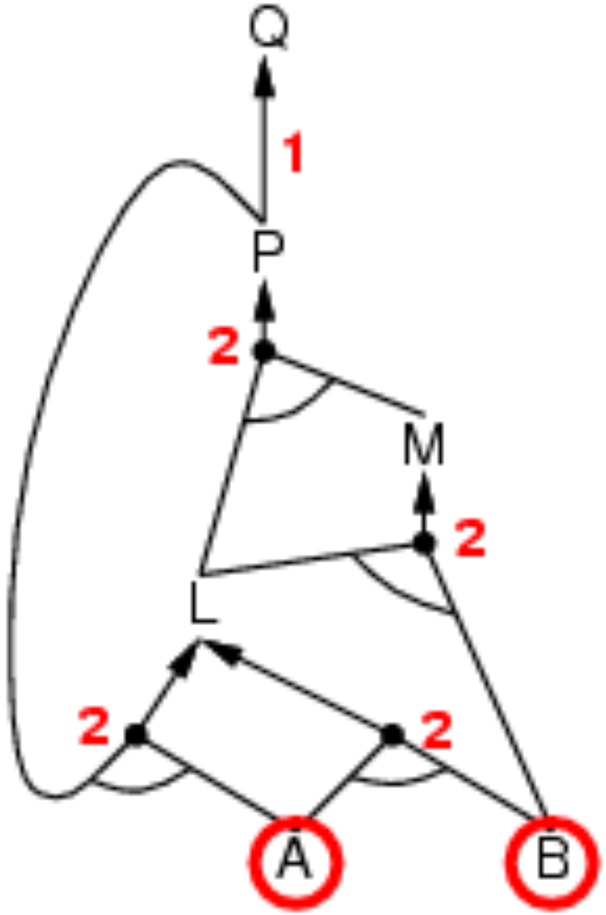
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

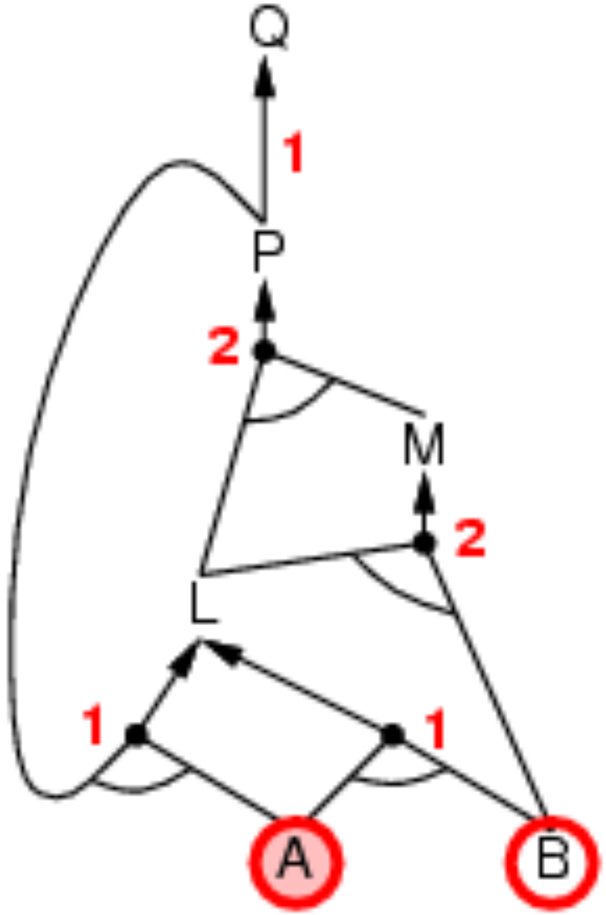
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

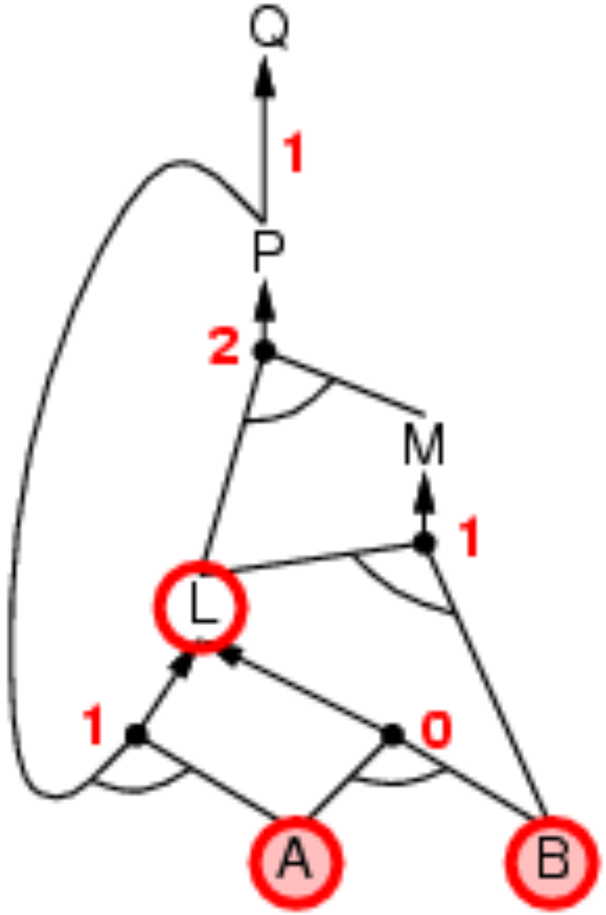
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

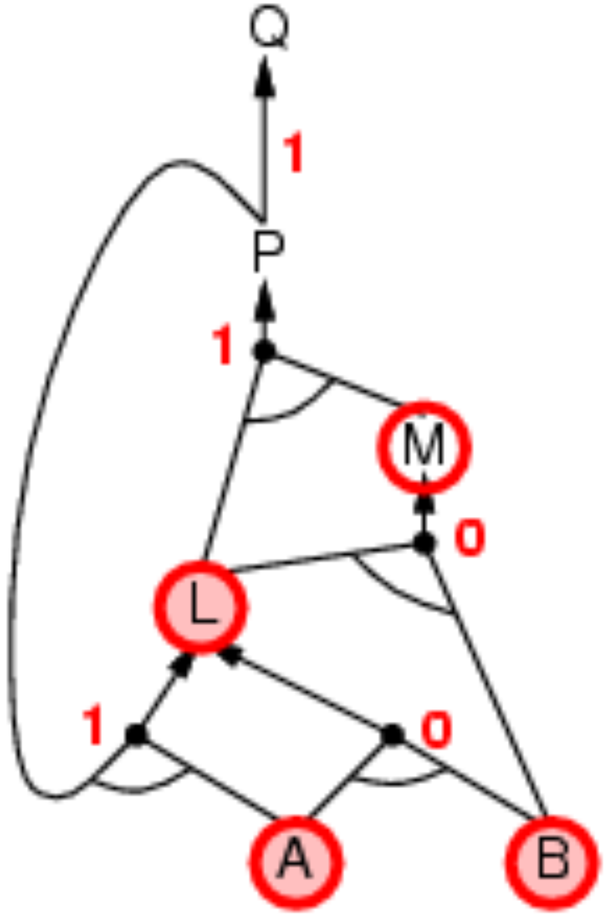
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

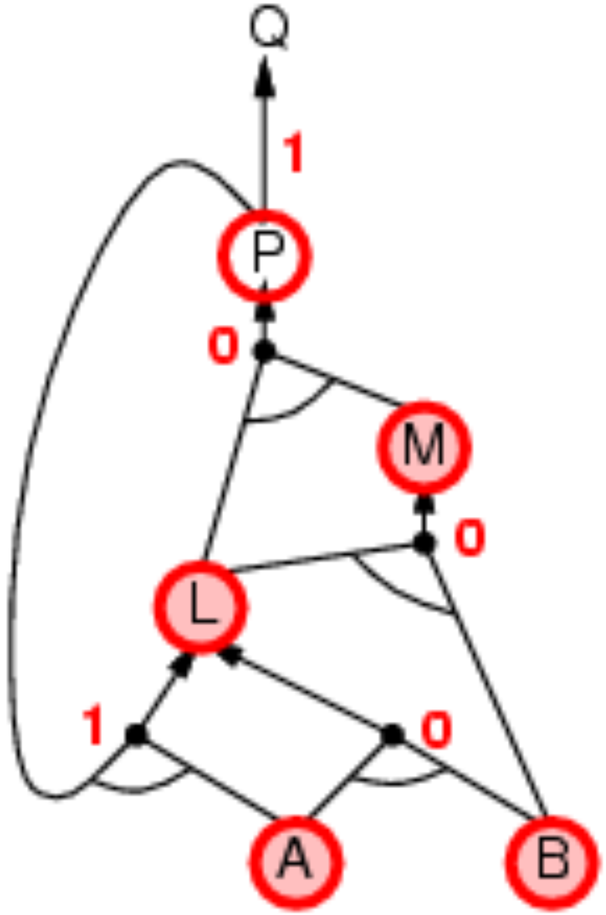
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

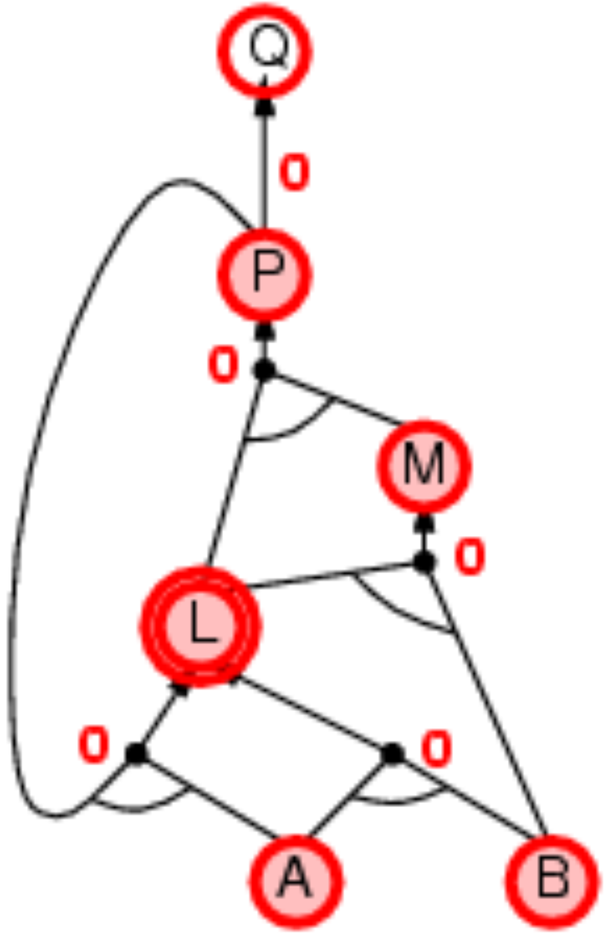
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

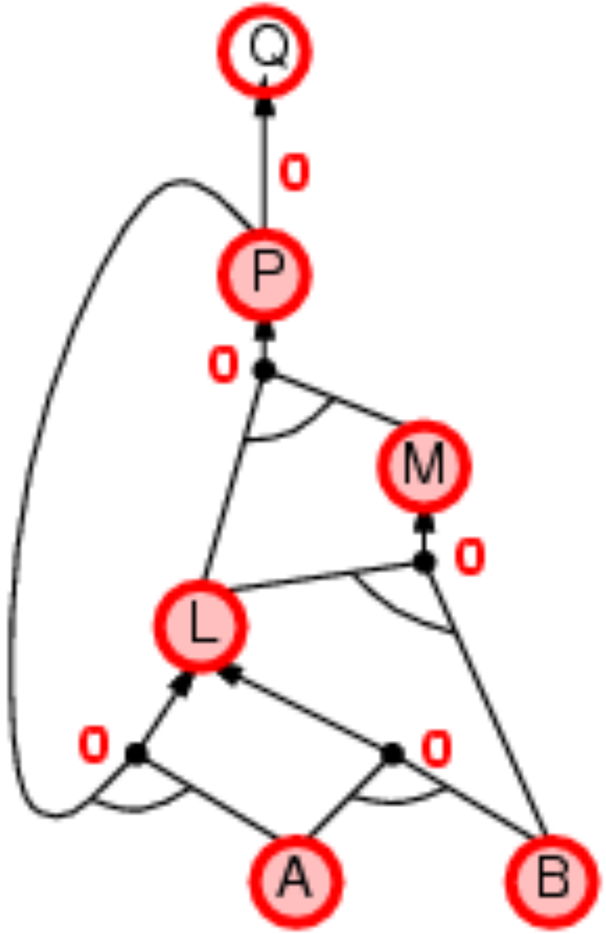
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



FORWARD CHAINING EXAMPLE

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

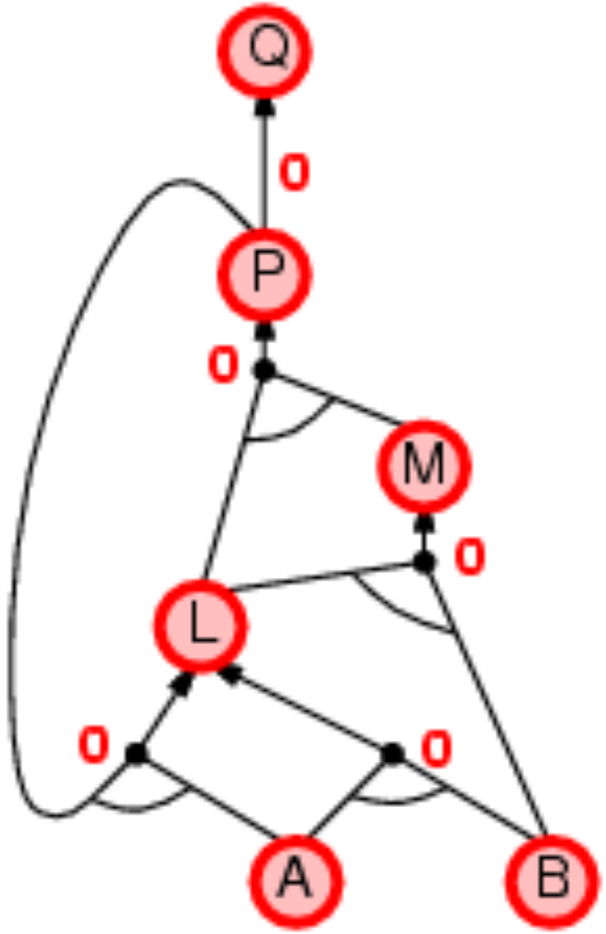
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



BACKWARD CHAINING

- Idea: work backwards from the query q:
  - to prove q by Backward Chaining,
    - check if q is known already, or
    - prove by Backward Chaining 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
  - has already been proved true, or
  - has already failed