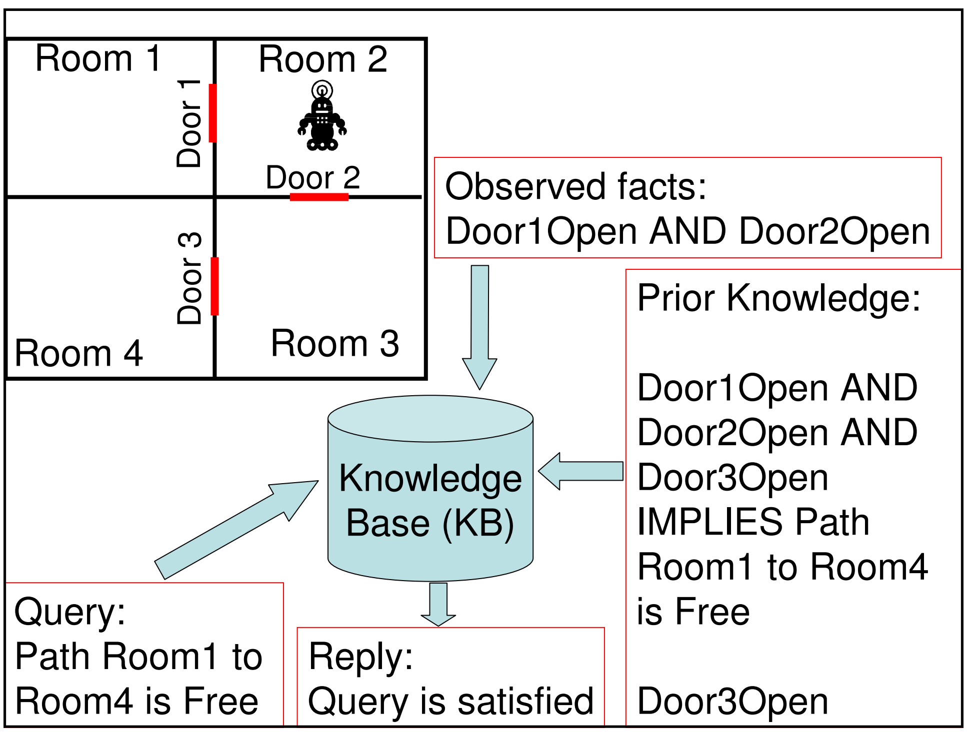
Logic and Reasoning

R&N 7-9



Representing Knowledge: Propositional Logic

- Symbols
- *True*, *False*
- Implication: =>
- Equivalence: <=>
- And (conjunction): ^
- Or (disjunction): V
- Negation: ¬
- Sentences = combination of the symbols, truth values, and operators
- Literals = Symbols and negated symbols (*A* and ¬*A* are both literals)

- Raining => Wet
- ¬(Busy ^ Sleeping)
- (A ^ B) V ¬C

Knowledge Base (KB)

- *Knowledge Base (KB):* a collection of sentences.
- *Model:* an assignment of (*True*/*False*) values to each of the symbols. If the knowledge base is built from n symbols, there are 2^n possible models.
- *Evaluation:* A sentence s is evaluated on a model m by setting each symbol to its corresponding value in m. The result of the evaluation is a value in {*True*,*False*}
- *KB Evaluation:* The result of the KB evaluation is the conjunction of the results of the evaluations of all the sentences in KB

Example KB

KB:

$A \vee B$

$\neg C \vee A$

A	*B*	*C*	*KB*
F	*F*	*F*	*F*
F	*F*	*T*	*F*
F	*T*	*F*	*T*
F	*T*	*T*	*F*
T	*F*	*F*	*T*
T	*F*	*T*	*T*
T	*T*	*F*	*T*
T	*T*	*T*	*T*

Logical Entailment

- "KB logically entails *S*" if all the models that evaluate KB to *True* also evaluate *S* to *True*.
- Denoted by: KB |= *S*
- Note: We do not care about those models that evaluate KB to *False*. The result of evaluating *S* for these models is irrelevant.

Example KB

KB:

$A \vee B$

$\neg C \vee A$

S:

$A \wedge C$

A	*B*	*C*	*KB*	*S*
F	F	F	F	F
F	F	T	F	F
F	T	F	T	F
F	T	T	F	F
T	F	F	T	F
T	F	T	T	T
T	T	F	T	F
T	T	T	T	T

KB $\not\models$ S because KB is true but *S* is false

Example KB

KB:

$A \vee B$

$\neg C \vee A$

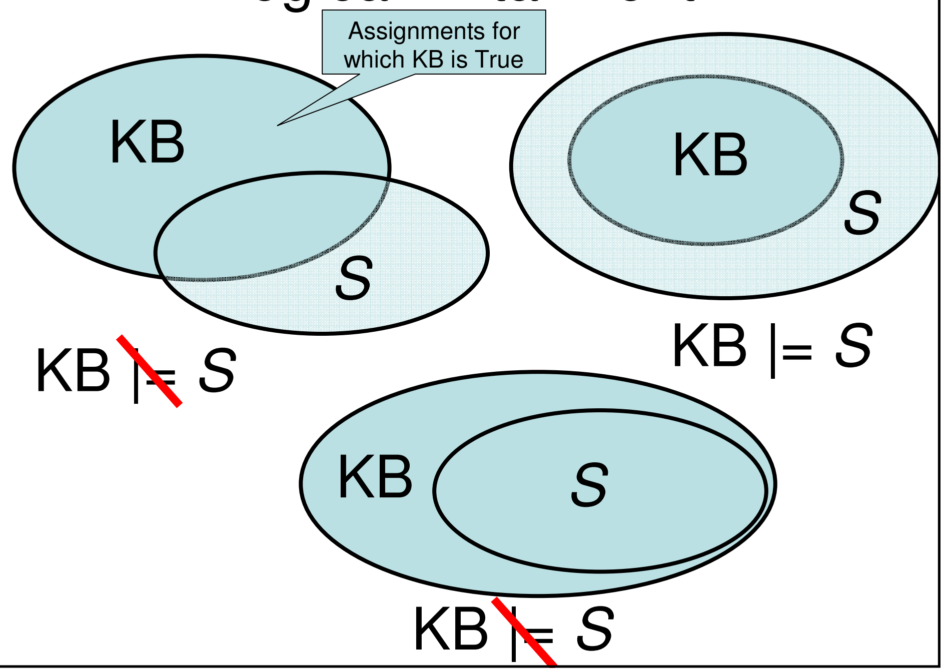
S:

$A \vee B \vee C$

A	*B*	*C*	*KB*	*S*
F	F	F	F	F
F	F	T	F	T
F	T	F	T	T
F	T	T	F	T
T	F	F	T	T
T	F	T	T	T
T	T	F	T	T
T	T	T	T	T

KB |= *S* because *S* is true for all the assignments for which KB is true

Logical Entailment



Inference

- An inference algorithm is a procedure for deriving a sentence from the KB
- KB $|-_i$ *S* means that *S* is inferred from KB using algorithm *i*.
- The inference algorithm is *sound* if it derives only sentences that are entailed by KB.
- The inference algorithm is *complete* if it can derive any sentence that is entailed by KB.

Examples

- Examples of sound inference rules

Premise

$$\frac{\alpha \wedge \beta}{\alpha}$$

Conclusion

And-Elimination. In words: if two things must be true, then either of them must be true.

$$\frac{\alpha \Rightarrow \beta \quad \alpha}{\beta}$$

Modus Ponens. In words: if α implies β and α is in the KB, then β must be entailed.

$$\frac{\alpha \ , \beta}{\alpha \ \wedge \ \beta}$$

And-Introduction.

Inference

- Basic problem:
  - We have a KB
  - We have a sentence *S* (the "query")
  - We want to check KB |= *S*
- Informally, "prove" *S* from KB
- Simplest approach: Model checking = evaluate all possible settings of the symbols
- Sound and complete (if the space of models is finite), but 2^n

Definitions

- *Valid:* A sentence is *valid* if it is true for all models.

$$\boxed{\alpha \vee \neg \alpha}$$

- *Satisfiable:* A sentence is *satisfiable* if it is true for some models.
- *Unsatisfiable:* A sentence is *unsatisfiable* if it is true for no models.

$$\boxed{\alpha \wedge \neg \alpha}$$

- *Proof by contradiction:* Given KB and a sentence S, establishing entailment is equivalent to proving that no model exists that satisfies KB and ¬*S.*

  KB *|=* *S* equivalent to (KB ^ ¬*S*) unsatisfiable

Proof as Search: Model Checking

- Enumerate values in the KB's truth table
- By default, exponential in the size of the KB
- All of the CSP techniques described earlier can be applied, in particular:
  - Backtracking search
  - Local search (hill-climbing, min-conflicts, → WalkSAT)

KB:
$A \vee B$
$\neg C \vee A$

A	B	C	KB
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

Proof as Search: Inference Rules

- Proof can be viewed as a search problem → The basic search algorithms that we saw before can be used here
  - State: KB
  - Successor: Apply inference to KB to obtain new sentences
  - Solution: Sequence of inferences to goal sentence. If the inference algorithm is sound, then this is guaranteed to establish entailment
- Questions: Is there an inference algorithm that guarantees efficient search? Will the search be complete?

Resolution

- A sentence is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses, each clause being a disjunction of literals
- Examples:

$$\underbrace{(A \vee B)}_{\text{Clause}} \wedge \underbrace{(C \vee D \vee J)}_{\text{Clause}} \wedge \underbrace{(E \vee G)}_{\text{Clause}}$$

- Key fact: It is always possible to convert any KB to CNF

CNF Conversion

1. $\alpha \Leftrightarrow \beta$ $\quad (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
2. $\alpha \Rightarrow \beta$ $\quad \neg\alpha \vee \beta$
3. $\neg(\alpha \wedge \beta)$ $\quad \neg\alpha \vee \neg\beta$
4. $\neg(\alpha \vee \beta)$ $\quad \neg\alpha \wedge \neg\beta$

To CNF

$$(A \wedge B) \Rightarrow C$$

1.

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

3.

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

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

Resolution

$$\frac{A_1 \vee \ldots \vee A_i \vee \ldots \vee A_n \qquad \neg A_i}{A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \vee \ldots \vee A_n}$$

- In words: If a literal appears in one clause and its negation in the other one, the two clauses can be merged and that literal can be discarded.

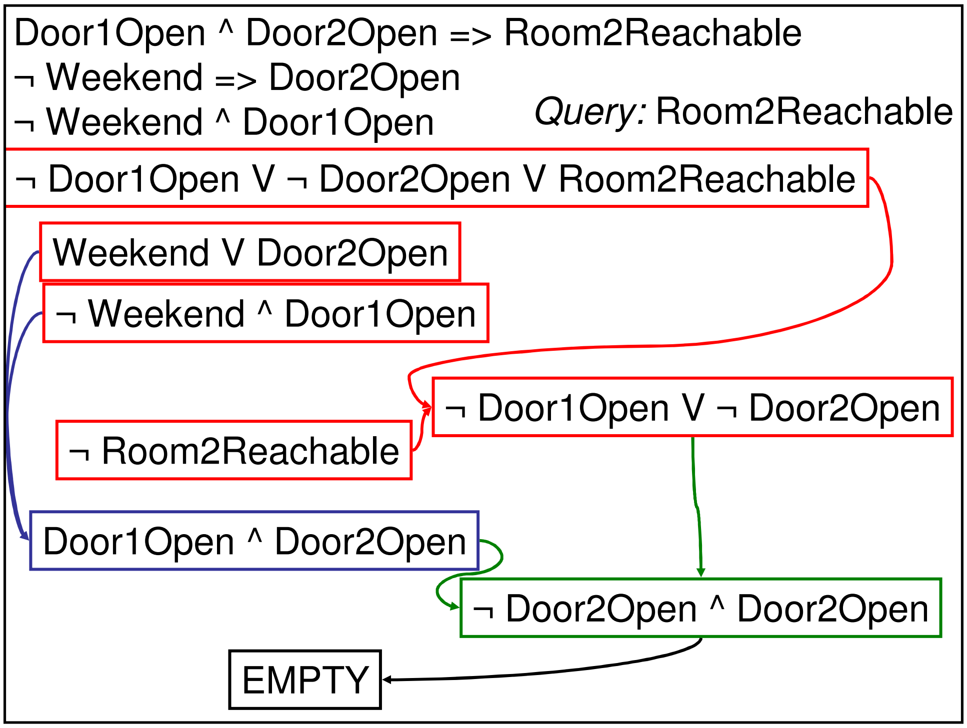
Resolution

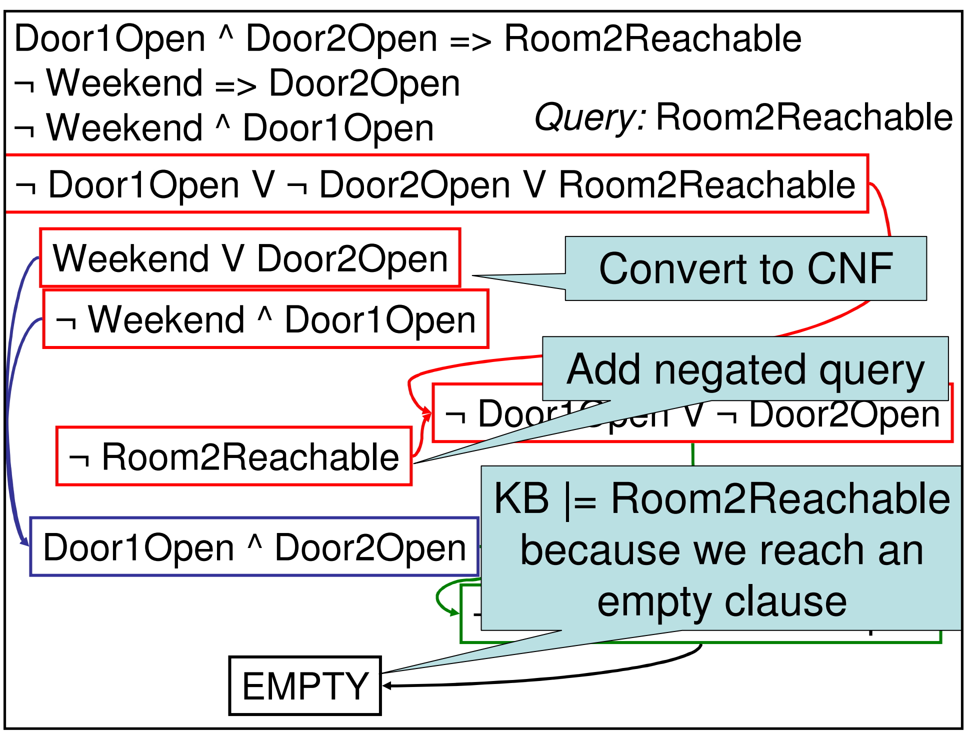
$$\frac{A_1 \vee \ldots \vee A_i \vee \ldots \vee A_n \quad B_1 \vee \ldots \vee \neg A_i \vee \ldots \vee B_m}{A_1 \vee \ldots \vee A_{i-1} \vee A_{i+1} \vee \ldots \vee A_n \vee B_1 \vee \ldots \vee B_{l-1} \vee B_{l+1} \vee \ldots \vee B_m}$$

- In words: If a symbol appears in one clause and its negation in the other one, the two clause can be merged and that symbol can be discarded.

Resolution Algorithm (In words)

- Suppose that KB ^ ¬*S* is in normal form.
- If KB entails *S*, then there should be a sequence of inferences through resolution that will lead to at least one clause that cannot be satisfied by any model
- Idea: Keep apply resolution to all the pairs of clauses in KB ^ ¬*S* until:
  - We can't find anymore clauses to resolve → KB does not entail *S*
  - We found an empty clause (which cannot be satisfied by any model) → KB does entail *S*


Door1Open ^ Door2Open => Room2Reachable
¬ Weekend => Door2Open
¬ Weekend ^ Door1Open
Query: Room2Reachable
¬ Door1Open V ¬ Door2Open V Room2Reachable
Weekend V Door2Open
¬ Weekend ^ Door1Open
¬ Door1Open V ¬ Door2Open
¬ Room2Reachable
Door1Open ^ Door2Open
¬ Door2Open ^ Door2Open
EMPTY


Door1Open ^ Door2Open => Room2Reachable
¬ Weekend => Door2Open
¬ Weekend ^ Door1Open
Query: Room2Reachable
¬ Door1Open V ¬ Door2Open V Room2Reachable
Weekend V Door2Open
Convert to CNF
¬ Weekend ^ Door1Open
Add negated query
¬ Room2Reachable
KB |= Room2Reachable because we reach an empty clause
Door1Open ^ Door2Open
EMPTY

Resolution Algorithm

- Input: KB and *S*
- Output: *True* if KB entails *S*, False otherwise
- Initialize: *Clauses* ← CNF(KB ^ ¬*S*)
  - Repeat:
    - For each pair of clauses C_i and C_j in *Clauses*
      - *R* ← Resolution(C_i, C_j)
      - If *R* contains the empty clause: Return True
      - *new* ← *new* U *R*
    - If *Clauses* contains *new*: Return False
    - *Clauses* ← *Clauses* U *new*

Resolution: Property

- *Resolution is sound:* Always produces sentences that are entailed by their premise
- *Resolution is complete:* It is guarantee to establish entailment of the query in finite time
- Completeness based on the key theorem: If a set of clauses is unsatisfiable, then the set of all clauses that can be obtained by resolution contains the empty clause
- So, conversely, if we cannot find the empty clause, the query must be satisfiable

Resolution can be Used to Check Consistency of a KB

- Repeat: Resolve pairs of sentences from the KB until
  - No more sentences can be produced → KB is consistent (satisfiable)
  - A unsatisfiable sentence is produced → KB is inconsistent

KB

Human => Mortal
Peter => Human

¬ Human V Mortal
¬ Peter V Human

CNF

Resolution ⬇

¬ Peter V Mortal

Human => Mortal
Peter => Human

Peter ^ Mortal => *False*
Peter

¬Human V Mortal

¬Peter V Human

¬Peter V ¬ Mortal V *False*

Peter

¬Peter V ¬Human V *False*

¬Peter V *False*

$\boxed{\textit{False}}$

Chaining

- "Special" case: The KB contains only two types of sentences:
  - Symbols
  - Sentences of the form:

    (*conjunction*) => *symbol*

$$\boxed{(A_1 \wedge \cdots \wedge A_n) \Rightarrow B}$$

Sentences of this type are "Horn clauses"

Chaining

- Basic inference mechanism ("Modus Ponens"):

$$\boxed{\frac{A_1 \wedge \ldots \wedge A_n \Rightarrow B \quad A_1 \wedge \ldots \wedge A_n}{B}}$$

- Basic idea: Given KB and a symbol *S*
  - *Forward chaining:* Repeatedly apply the inference rule to KB until we get to *S*
  - *Backward chaining:* Start from *S* and find implications whose conclusions are *S*

Sentences of this type are "Horn clauses"

Forward Chaining

$$\underbrace{A_1 \wedge \ldots \wedge A_n}_{} \Rightarrow B$$

Counter = number of symbols on left hand-side

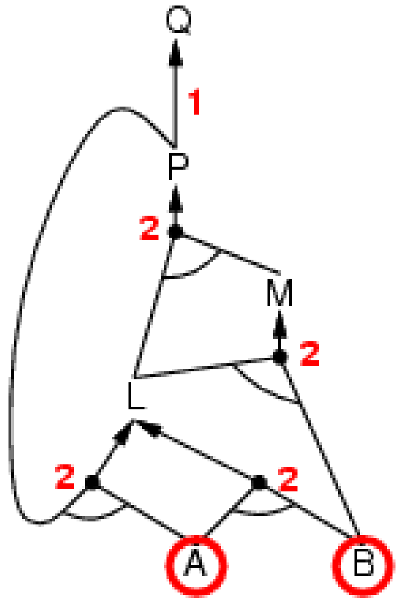
If Counter = 0→Infer symbol on right-hand side, *B*

Forward Chaining

- Maintain a current list of symbols from KB
- Initialize a counter for each clause in KB = number of symbols on the left-hand side
- Repeat:
  - Get the next symbol *P* from the queue
  - If *P* = *S*
    - We are done, KB |= *S*
  - Else
    - Decrease the counter of each clause in which *P* appear in the left-hand side
    - If the counter is 0: Add the right-hand side of the clause to the list

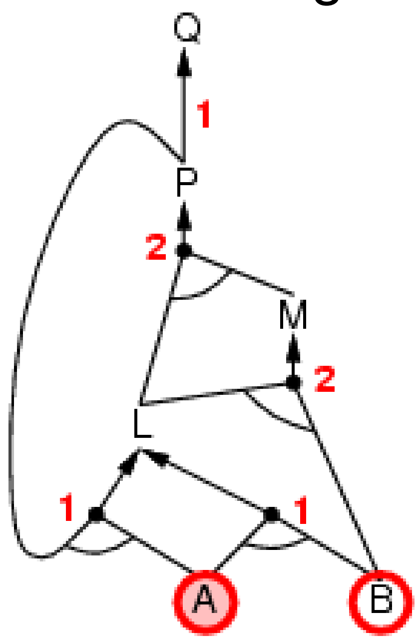
Forward Chaining

P => *Q*
L ^ *M* => *P*
B ^ *L* => *M*
A ^ *P* => *L*
A ^ *B* => *L*
A
B



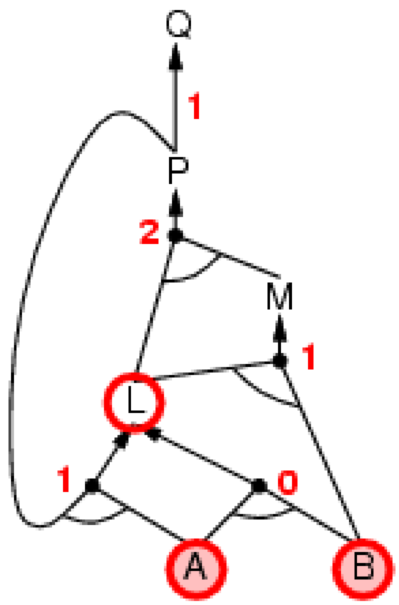
Forward Chaining

P => *Q*
L ^ *M* => *P*
B ^ *L* => *M*
A ^ *P* => *L*
A ^ *B* => *L*
A
B



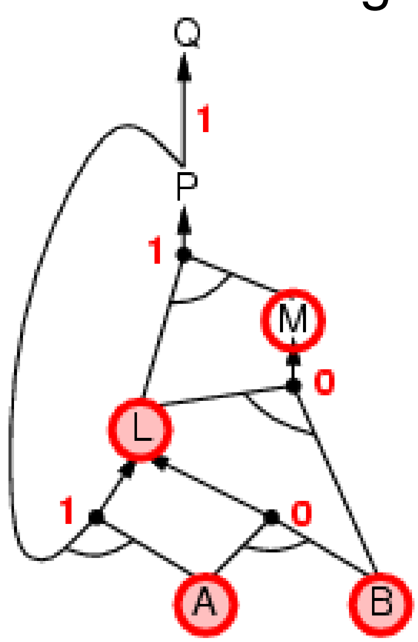
Forward Chaining

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



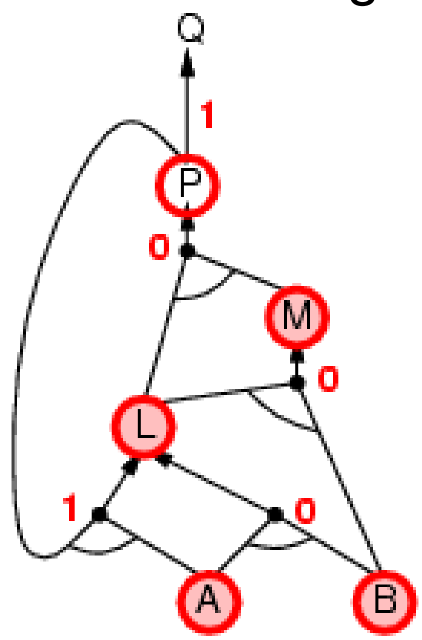
Forward Chaining

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



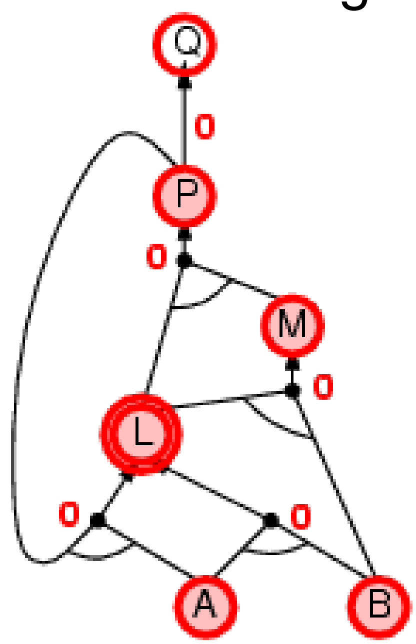
Forward Chaining

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



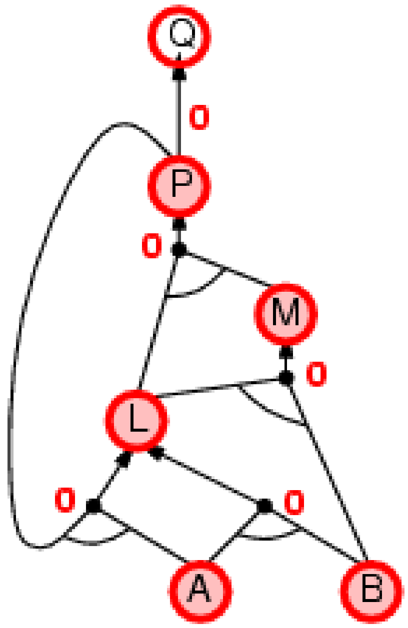
Forward Chaining

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



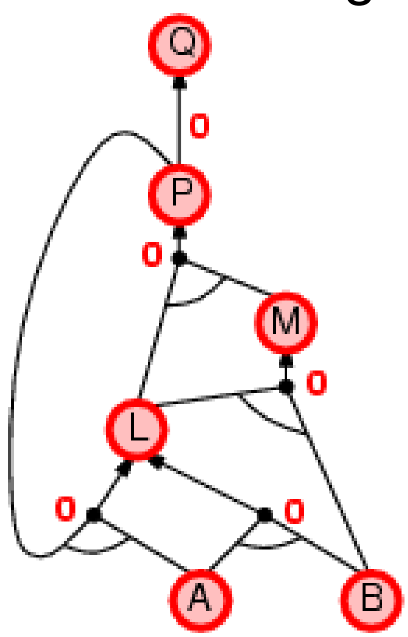
Forward Chaining

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



Forward Chaining

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



Forward/Backward Chaining

- Both algorithms are
  - *Sound* (valid inferences)
  - *Complete* (every entailed symbol can be derived)
- Both algorithms are linear in the size of the knowledge base
- *Forward=data-driven:* Start with the data (KB) and draw conclusions (entailed symbol) through logical inferences
- *Backward=goal-driven:* Start with the goal (entailed symbol) and check backwards if it can be generated by an inference rule

Summary

- Knowledge base (KB) as list of sentences
- Entailment verifies that query sentence is consistent with KB
- Establishing entailment by direct model checking is exponential in the size of the KB, but:
  - If KB is in CNF form (always possible): Resolution is a sound and complete procedure
  - If KB is composed of Horn clauses:
  - Forward and backward checking algorithms are linear, and are sound and complete
- Shown so far using a restricted representation (propositional logic)
- What is the problem with using these tools for reasoning in real-world scenarios?