Propositional Logic

- Knowledge-based agents
- Logic in general—models and entailment
- Propositional (Boolean) logic
- Equivalence, validity, satisfiability
- Inference rules and theorem proving
  - resolution
  - forward chaining
  - backward chaining
- Effective Propositional Model Checking

knowledge-based agents

- Agents that reason by operating on internal representations of knowledge
- Need a model for reasoning For example:

1. If it didn't rain, Harry visited Hagrid today.
2. Harry visited Hagrid or Dumbledore today, but not both.
3. Harry visited Dumbledore today.

Query: Did it rain today?

Example

1. If it didn't rain, Harry visited Hagrid today.
2. Harry visited Hagrid or Dumbledore today, but not both.
3. Harry visited Dumbledore today.

➡ 4. Harry did not visit Hagrid today. #3, #2

➡ 5. It rained today. #4, #1

Knowledge bases

Inference engine ← **domain−independent algorithms**

Knowledge base ← **domain−specific content**

- Sentence is an assertion about the world in a knowledge representation (formal) language
- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - Tell it what it needs to know
    Then it can Ask itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level OR at the implementation level

A simple knowledge-based agent

The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

Formal Definition: Propositional Logic

- A **Proposition** is a statement that is true or false.
- **Propositional symbols/variables** represent propositions. They are like Boolean variables in a programming language. (P, Q, R)
- Operators/Functions

And/Conjunction

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Or/Disjunction

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

Implication/Conditional

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

Not/Negation

P	$\sim P$
T	F
F	T

If and only if/Biconditional

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

Propositional Logic

Sentences are assertions about a world in a knowledge representation language – like Propositional Logic

Sentences in propositional logic are represented by

- Propositional Symbols: P_1, P_2, Harry, …
- Logical connectives (operators) in programming languages:
  - If S is a sentence, ¬S is a sentence (negation)
  - If S_1 and S_2 are sentences, $S_1 \land S_2$ is a sentence (conjunction)
  - If S_1 and S_2 are sentences, $S_1 \lor S_2$ is a sentence (disjunction)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Model

- A **model** is an assignment of a truth value to every propositional **symbol** (*a possible world*)
  - n symbols ⇒ **2^n possible worlds (2^n models)**

- Example:
  - P: It is raining
  - Q: It is Tuesday
  - {P = true, Q = false} is a possible world

- **A knowledge base, KB**, is a set of **sentences** known by a knowledge-based agent to be true.
  - These can be used by a logical agent to make logical inferences about the world

Propositional logic: Semantics (Meaning)

Each model specifies true/false for each proposition symbol in KB

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$
$true$ $true$ $false$

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth of **sentences** with respect to a model ***m***:

$\neg S$	is true iff	S	is false			
$S_1 \wedge S_2$	is true iff	S_1	is true	and	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true	or	S_2	is true
$S_1 \Rightarrow S_2$	is true iff	S_1	is false	or	S_2	is true
i.e.,	is false iff	S_1	is true	and	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true	and	$S_2 \Rightarrow S_1$	is true

Simple recursive process evaluates an arbitrary sentence, e.g.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Entailment: KB |= β

- Entailment means that one thing follows from other things being true (KB)
- In every model in which sentences in the knowledge base KB are true, sentence β is also true.

Models are incomplete but still useful if faithful to real world

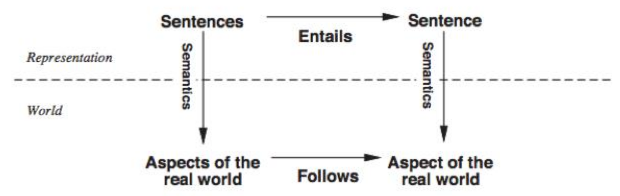


Figure 7.6 Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

Inference

- The process of deriving truth value of new sentences from old ones (those already in the knowledge base)

P: It is a Tuesday.

Q: It is raining.

R: Harry will go for a run.

KB: $(P \land \neg Q) \rightarrow R$ $\quad$ P $\quad$ $\neg Q$

Inference: R

Proof methods

Proof methods divide into (roughly) two kinds:

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 alg.
– Typically require translation of sentences into a normal form

Model checking

truth table enumeration (always exponential in n)
improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
heuristic search in model space
e.g., min-conflicts   uses hill-climbing algorithms

Inference Algorithms: Model Checking

- Does KB |= β ?

- **Model Checking**: To determine if KB ⊨ β :
  - Enumerate all possible models.
  - If in every model where KB is true, β is true, then KB entails β.
  - Otherwise, KB does not entail β.

Model Checking is

- Sentences
  - P: It is a Tuesday.
  - Q: It is raining.
  - R: Harry will go for a run

KB: (P ∧ ¬Q) → R, P, ¬Q
Query: R
(Does KB |= R?)

P	*Q*	*R*	KB
false	false	false	
false	false	true	
false	true	false	
false	true	true	
true	false	false	
true	false	true	
true	true	false	
true	true	true	

Model Checking is

- Sentences
  - P: It is a Tuesday.
  - Q: It is raining.
  - R: Harry will go for a run

KB: (P ∧ ¬Q) → R, P, ¬Q
Query: R
(Does KB |= R?)

P	*Q*	*R*	KB
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	false
true	false	true	true
true	true	false	false
true	true	true	false

Model Checking is

- Sentences
  - P: It is a Tuesday.
  - Q: It is raining.
  - R: Harry will go for a run

KB: (P ∧ ¬Q) → R, P, ¬Q
Query: R
(Does KB |= R?)

P	*Q*	*R*	KB
false	false	false	false
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	false
true	false	true	true
true	true	false	false
true	true	true	false

Stopped here

Last Time: Propositional Logic

- Propositional symbols: have T/F values (Atomic facts about the world)
- Operators: $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$
- World: A set of propositional symbols
- Sentences: assertions about the world (expressions using symbols and operators)
- Model: Assignment of truth value to every propositional symbol in world
- Knowledge Base, KB: { sentences known to be True}
- Entailment: KB $\models \beta$ used to make inferences:
  In every model where KB is true, β is also true
- Model checking: check every model $\Rightarrow 2^n$ yikes!

Sample Logic Library in Python: Logic.py

```
class Symbol(Sentence):
   def __init__(self, name):
      self.name = name
   ….

class Not(Sentence):
   def __init__(self, operand):
      Sentence.validate(operand)
      self.operand = operand
…

class And(Sentence):
   def __init__(self, *conjuncts):
      for conjunct in conjuncts:
   …

class Or(Sentence):
   def __init__(self, *disjuncts):
      for disjunct in disjuncts:
   …
```

Example: Using logic.py harry.py

from logic import *

```
rain=Symbol("rain")              # It is raining
hagrid=Symbol("hagrid")          # Harry visited Hagrid
dumbledore=Symbol("dumbledore") # Harry visited Dumbledore
sentence = And(rain, hagrid)
print(sentence.formula())
```

```
>>> Python harry.py    # run
>>> rain ∧ hagrid      # output
```


Example: Using logic.py harry.py

```
from logic import *
rain = Symbol("rain")                    # It is raining
hagrid = Symbol("hagrid")                # Harry visited Hagrid
dumbledore = Symbol("dumbledore")        # Harry visited Dumbledore
knowledge = And(          # knowledge is an “And” of mult sentences
   Implication(Not(rain), hagrid),         #    (¬rain) ⇒ hagrid
   Or(hagrid, dumbledore),
   Not(And(hagrid, dumbledore)),
   dumbledore
)
print(knowledge.formula))    =>    ???
```

Want to do model checking

```
def model_check(knowledge, query):
    """Checks if knowledge base entails query."""
    def check_all(knowledge, query, symbols, model):
        """Checks if knowledge base entails query, given a particular model."""
        # If model has an assignment for each symbol
        if not symbols:
            # If knowledge base is true in model, then query must also be true for entailment
            if knowledge.evaluate(model):
                return query.evaluate(model)
            return True                if knowledge is not true then ignore so return true
        else:       # Choose one of the remaining unused symbols
            remaining = symbols.copy()
            p = remaining.pop()
```


Want to do model checking

```
            # Create a model where the symbol is true
            model_true = model.copy()
            model_true[p] = True
            # Create a model where the symbol is false
            model_false = model.copy()
            model_false[p] = False
            # Ensure entailment holds in both models
            return (check_all(knowledge, query, remaining, model_true) and
                    check_all(knowledge, query, remaining, model_false))

    # Get all symbols in both knowledge and query
    symbols = set.union(knowledge.symbols(), query.symbols())

    # Check that knowledge entails query
    return check_all(knowledge, query, symbols, dict())
```

Example: Using logic.py harry.py

```
from logic import *
rain = Symbol("rain")                    # It is raining
hagrid = Symbol("hagrid")                # Harry visited Hagrid
dumbledore = Symbol("dumbledore")        # Harry visited Dumbledore

knowledge = And(          # knowledge is an “And” of mult sentences
   Implication(Not(rain), hagrid),       #   (¬rain) ⇒ hagrid
   Or(hagrid, dumbledore),
   Not(And(hagrid, dumbledore)),
   dumbledore
)
>>> print(model_check(knowledge, rain))    # info known, query
>>> True
```


Inference Algorithms: Resolution

Inference Rules: Modus Ponens

$$\frac{\alpha \rightarrow \beta \qquad \alpha}{\beta}$$

If it is raining, then Harry is inside.

It is raining.

Harry is inside.

Inference Rules: AND elimination

$$\frac{\alpha \land \beta}{\alpha}$$

Harry is friends with Ron and Hermione.

Harry is friends with Hermione.

Inference Rules: Double Negation Elimination

$$\frac{\neg(\neg\alpha)}{\alpha}$$

It is not true that Harry did not pass the test.

Harry passed the test.

Inference Rules: Conditional Elimination

$$\frac{\alpha \rightarrow \beta}{\neg\alpha \vee \beta}$$

If it is raining, then Harry is inside.

It is not raining or Harry is inside.

Inference Rules: Biconditional Elimination

$$\frac{\alpha \leftrightarrow \beta}{(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)}$$

It is raining if and only if Harry is inside.

If it is raining, then Harry is inside, and if Harry is inside, then it is raining.

Inference Rules: DeMorgan's Laws

$$\frac{\neg(\alpha \wedge \beta)}{\neg\alpha \vee \neg\beta}$$

$$\frac{\neg(\alpha \vee \beta)}{\neg\alpha \wedge \neg\beta}$$

It is not true that both Harry and Ron passed the test.

Harry did not pass the test or Ron did not pass the test.

It is not true that Harry or Ron passed the test.

Harry did not pass the test and Ron did not pass the test.

Distributive Properties

$$\frac{(\alpha \land (\beta \lor \gamma))}{(\alpha \land \beta) \lor (\alpha \land \gamma)}$$

$$\frac{(\alpha \lor (\beta \land \gamma))}{(\alpha \lor \beta) \land (\alpha \lor \gamma)}$$

Logical equivalence

Two sentences are logically equivalent iff true in same models:
$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$(\alpha \land \beta)$	$\equiv$	$(\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta)$	$\equiv$	$(\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma)$	$\equiv$	$(\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma)$	$\equiv$	$(\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg \alpha)$	$\equiv$	α	double-negation elimination
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg \beta \Rightarrow \neg \alpha)$	contraposition
$(\alpha \Rightarrow \beta)$	$\equiv$	$(\neg \alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta)$	$\equiv$	$((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta)$	$\equiv$	$(\neg \alpha \lor \neg \beta)$	De Morgan
$\neg(\alpha \lor \beta)$	$\equiv$	$(\neg \alpha \land \neg \beta)$	De Morgan
$(\alpha \land (\beta \lor \gamma))$	$\equiv$	$((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma))$	$\equiv$	$((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Resolution

- Convert the inference rules and data to Conjunctive Normal Form
- Conjunctive Normal Form (CNF) is a sentence in Propositional logic consisting of:
  - clauses connected by $\wedge$ (AND) where each
  - clause is literal symbols connected by $\vee$ (OR)
- The using resolution determine if the KB entails what we are trying to prove

Resolution

$$\frac{P \vee Q \qquad \neg P}{Q}$$

$$\frac{P \vee Q_1 \vee Q_2 \vee \ldots \vee Q_n \qquad \neg P}{Q_1 \vee Q_2 \vee \ldots \vee Q_n}$$

Resolution

$$\frac{P \vee Q \qquad \neg P \vee R}{Q \vee R}$$

$$\frac{P \vee Q_1 \vee Q_2 \vee \ldots \vee Q_n \qquad \neg P \vee R_1 \vee R_2 \vee \ldots \vee R_m}{Q_1 \vee Q_2 \vee \ldots \vee Q_n \vee R_1 \vee R_2 \vee \ldots \vee R_m}$$

Conjunctive Normal Form

Def: A clause is a disjunction of literals

e.g. P ∨ Q ∨ R

A logical sentence is in Conjuntive Normal Form (CNF) if it is a conjunction of clauses

e.g. (A ∨ B ∨ C) ∧ (D ∨ ¬E) ∧ (F ∨ G)

Conversion to CNF

- Eliminate biconditionals
  - turn (α ↔ β) into (α → β) ∧ (β → α)
- Eliminate implications
  - turn (α → β) into ¬α ∨ β
- Move ¬ inwards using De Morgan's Laws
  - e.g. turn ¬(α ∧ β) into ¬α ∨ ¬β
- Use distributive law to distribute ∨ wherever possible

Example: Conversion to CNF

(P ∨ Q) → R	
¬(P ∨ Q) ∨ R	eliminate implication
(¬P ∧ ¬Q) ∨ R	De Morgan's Law
(¬P ∨ R) ∧ (¬Q ∨ R)	distributive law

Example: Inference by resolution

$$\frac{P \vee Q \qquad \neg P \vee R}{(Q \vee R)}$$

$$\frac{P \vee Q \vee S \qquad \neg P \vee R \vee S}{(Q \vee S \vee R \vee S)}$$

$$\frac{P \vee Q \vee S \qquad \neg P \vee R \vee S}{(Q \vee R \vee S)}$$

$$\frac{P \qquad \neg P}{(\,)}$$

False

Inference by Resolution

To determine if KB ⊨ α:

Check if (KB ∧ ¬α) is a contradiction?

If so, then KB ⊨ α.

Otherwise, no entailment.

Inference by Resolution

To determine if KB ⊨ α:

- Convert (KB ∧ ¬α) to Conjunctive Normal Form.
- Keep checking to see if we can use resolution to produce a new clause.
  - If ever we produce the empty clause (equivalent to False), we have a contradiction, and KB ⊨ α.
  - Otherwise, if we can't add new clauses, no entailment.

Inference by Resolution

Does (A ∨ B) ∧ (¬B ∨ C) ∧ (¬C) entail A?

(A ∨ B) ∧ (¬B ∨ C) ∧ (¬C) ∧ (¬A)

(A ∨ B)   (¬B ∨ C)   (¬C)   (¬A)

(A ∨ B)   (¬B ∨ C)   (¬C)   (¬A)   (¬B)

(A ∨ B)   (¬B ∨ C)   (¬C)   (¬A)   (¬B)   (A)

( )

Proving as a Search Problem

- initial state: starting knowledge base
- actions: inference rules
- transition model: new knowledge base after inference
- goal test: check statement we're trying to prove
- path cost function: number of steps in proof

Forward and backward chaining

Horn Form (restricted)

KB = conjunction of Horn clauses

Horn clause =

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

KB example, $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Modus Ponens (for Horn Form): complete for Horn KBs

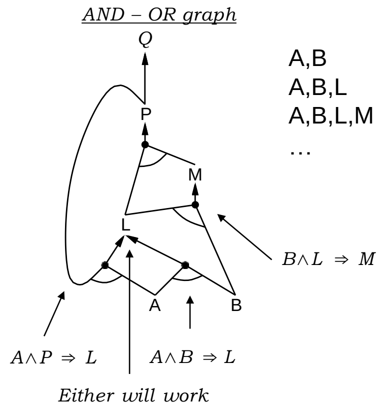
$$\frac{a_1, \ldots, a_n, \qquad a_1 \wedge \cdots \wedge a_n \;\Rightarrow\; \beta}{\beta}$$

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in linear time

Forward chaining

Idea: fire any rule whose premises are satisfied in the 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



A,B $A \wedge B \Rightarrow L$
A,B,L $B \wedge L \Rightarrow M$
A,B,L,M $L \wedge M \Rightarrow P$
…

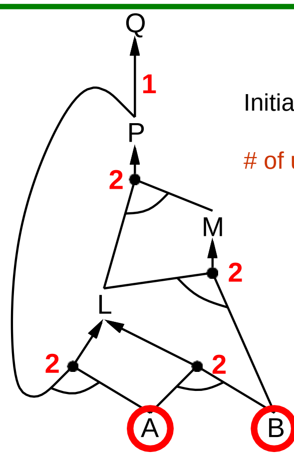
Forward chaining algorithm

```
function PL-FC-Entails?(KB, q) returns true or false
   inputs: KB, the knowledge base, a set of propositional Horn clauses
           q, the query, a proposition symbol
  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 in KB

   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 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

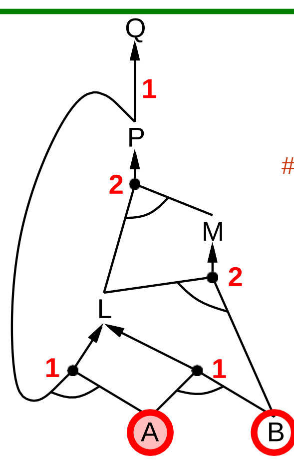


Initialize

# of unsatisfied premises

Forward chaining example

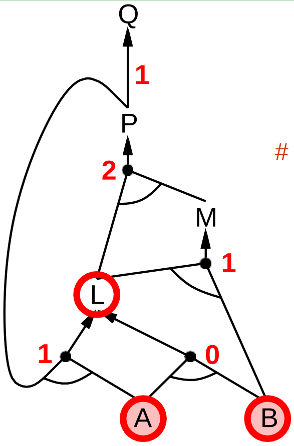
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



# of unsatisfied premises

Forward chaining example

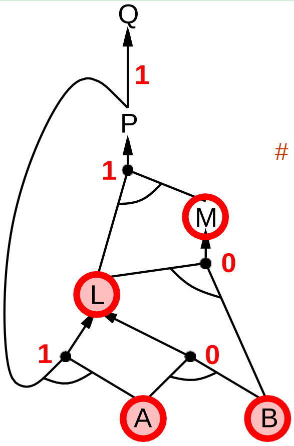
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



# of unsatisfied premises

Forward chaining example

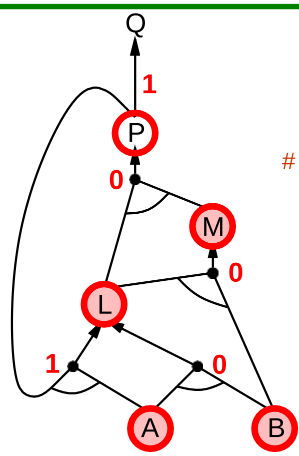
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



# of unsatisfied premises

Forward chaining example

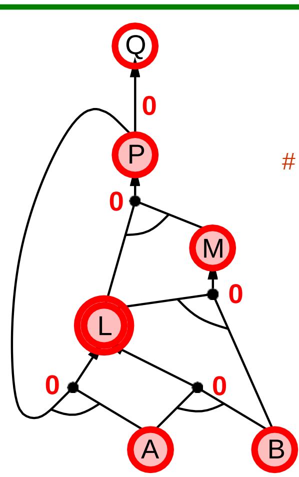
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



\# of unsatisfied premises

Forward chaining example

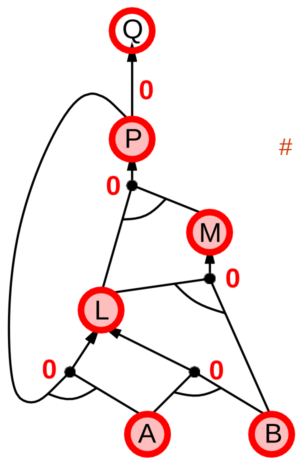
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



\# of unsatisfied premises

Forward chaining example

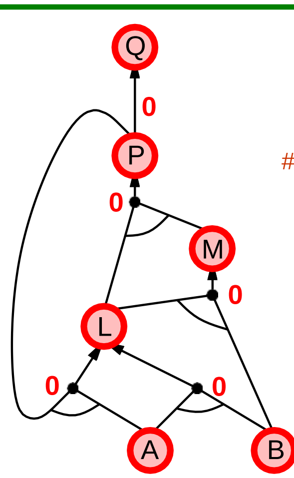
$P \Rightarrow Q$
$L \land M \Rightarrow P$
$B \land L \Rightarrow M$
$A \land P \Rightarrow L$
$A \land B \Rightarrow L$
A
B



# of unsatisfied premises

Forward chaining example

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



# of unsatisfied premises

Wrap up needed