Propositional Logic[1]

Lecture 2

[1]The slides have been prepared using the textbook material available on the web, and the slides of the previous editions of the course by Prof. Luigia Carlucci Aiello

Summary

♢ Propositional Logic   Russell & Norvig Sec. 7.4

♢ SAT   Russell & Norvig Sec. 7.6

♢ Reasoning in propositional logic   Russell & Norvig Sec. 7.5, part

Propositional logic is the simplest logic—illustrates basic approach to knowledge representation

Propositional logic: Syntax

$True$ and $False$ are propositional symbols
Other propositional symbols are denoted as P_1, P_2, ...

If S is a propositional symbol, S is a sentence

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ 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)

Propositional logic: Notation

Alternatives to R&N notation:

$\supset, \rightarrow$ for $\Rightarrow$;
$\equiv$ for $\Leftrightarrow$;
$0, 1$ for $True, False$

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i, j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i, j]$.

Percepts acquired after detecting nothing in [1,1], moving right, breeze in [2,1]

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

Propositional logic: Semantics

The truth of a sentence can be determined, given an interpretation m, that assigns a truth value to every propositional symbol:

$True$	is true
$False$	is false
P	is true iff P is true in m
P	is false iff P is false in m

Propositional logic: Semantics

Rules for evaluating truth of complex sentences:

$\neg S$	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

Propositional logic: Semantics

Each interpretation specifies true/false for each proposition symbol

E.g. $\begin{array}{ccc} P_{1,2} & P_{2,2} & P_{3,1} \\ false & true & false \end{array}$

A simple recursive process evaluates an arbitrary sentence, e.g.,

$$\begin{aligned}
&\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \\
&true \wedge (true \vee false) = \\
&true \wedge true = \\
&true
\end{aligned}$$

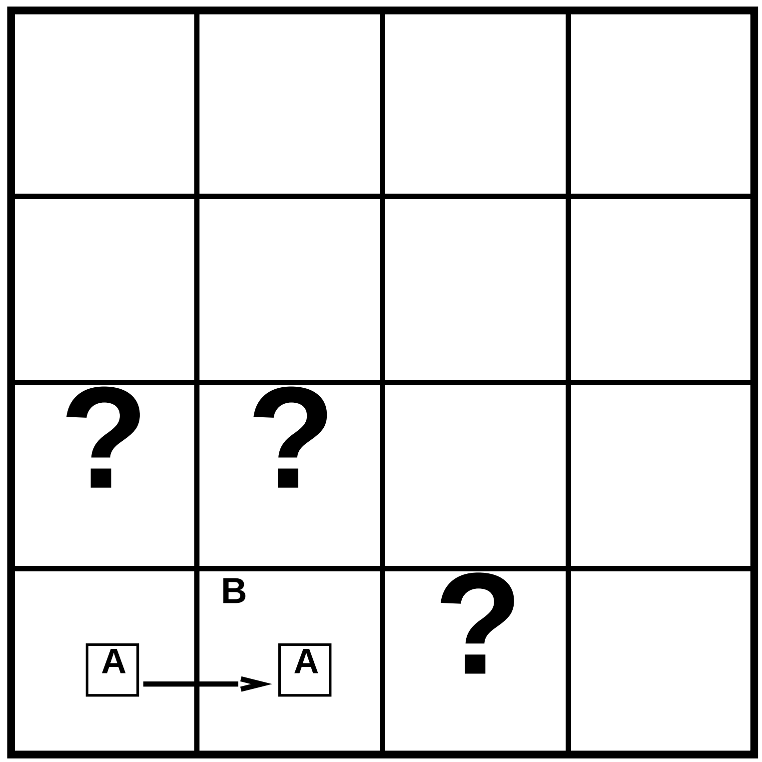
Truth tables for connectives

P	Q	$\neg P$	$P \land Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Wumpus world KB: modeling percepts

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

Is the next move safe? 3 Boolean choices $\Rightarrow$ 8 possible cases



The wumpus world: modeling environment

"Pits cause breezes in adjacent squares"

$$
\begin{aligned}
B_{1,1} &\Leftrightarrow (P_{1,2} \lor P_{2,1}) \\
B_{2,1} &\Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})
\end{aligned}
$$

"A square is breezy *if and only if* there is an adjacent pit"

Entailment (recall)

Entailment:

$$KB \models \alpha$$

Knowledge base KB entails a sentence α

if and only if

α is true in all models of KB

$KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	KB	$\neg P_{1,2}$
$false$	$false$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$false$	$false$	$false$	$false$	$false$	$true$	$false$	$true$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$false$	$true$	$false$	$false$	$false$	$false$	$false$	$false$	$true$
$false$	$true$	$false$	$false$	$false$	$false$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$false$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$false$	$true$	$true$	$\underline{true}$	$\underline{true}$
$false$	$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$true$	$true$	$true$	$true$	$true$	$true$	$true$	$false$	$false$

Checking $\models$ by model enumeration

```
function TT-ENTAILS?(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   symbols ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, { })

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true        //  when KB is false, always return true
   else
      P ← FIRST(symbols)
      rest ← REST(symbols)
      return (TT-CHECK-ALL(KB, α, rest, model ∪ {P = true})
             and
             TT-CHECK-ALL(KB, α, rest, model ∪ {P = false }))
```

Depth-first: $O(2^n)$ for n symbols; problem is co-NP-complete

Logical equivalence

Two sentences are logically equivalent iff true in same models:

$$\alpha \equiv \beta \quad \text{if and only if} \quad \alpha \models \beta \text{ and } \beta \models \alpha$$

Logical equivalence table

$$
\begin{array}{rcll}
(\alpha \land \beta) & \equiv & (\beta \land \alpha) & \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv & (\beta \lor \alpha) & \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv & (\alpha \land (\beta \land \gamma)) & \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv & (\alpha \lor (\beta \lor \gamma)) & \text{associativity of } \lor \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \lor \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv & (\neg\alpha \lor \neg\beta) & \text{de Morgan} \\
\neg(\alpha \lor \beta) & \equiv & (\neg\alpha \land \neg\beta) & \text{de Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv & ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv & ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{distributivity of } \lor \text{ over } \land
\end{array}
$$

Validity and satisfiability

A sentence is valid if it is true in *all* interpretations,

e.g., $True$, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

A sentence is satisfiable (consistent) if it is true in some interpretation (i.e. it has a model)

e.g., $A \lor B$, C

A sentence is unsatisfiable (inconsistent) if it is true in nointerpretation (i.e. it has no model)

e.g., $A \land \neg A$

α is valid if $\neg\alpha$ is unsatisfiable.

α is satisfiable if $\neg\alpha$ is not valid.

Inference with Validity and Satisfiability

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

i.e., prove α by *refutation* or *reductio ad absurdum*

Reasoning in propositional logic

Two decision problems:

- ♢ tautology checking ($TAUT$)
- ♢ satisfiability checking (SAT)

$TAUT$ and SAT are exponential in the size of the formula (the number of propositional symbols).

SAT is NP-Complete and $TAUT$ is Co-NP-complete.

Reasoning methods

Reasoning methods can be (roughly) divided in two kinds:

Model checking

–truth table enumeration (always exponential in n)

Deduction

– generation of new sentences through the application of inference rules

Efficient model checking

1. heuristic search, e.g., DPLL

2. local search in model space (sound but incomplete)
   e.g., min-conflicts-like, hill-climbing algorithms

Typically require translation of sentences into a normal form

Conjuntive Normal Form

In **Conjunctive Normal Form (CNF)**:

◇ the KB is represented as a set of clauses.

◇ a *clause* is a disjuntion of literals $L_1 \vee L_2 \vee \cdots \vee L_n$.

Example

$\{\{A, \neg B \, \neg C\}, \{\neg A, B\}\} \;\Leftrightarrow\; (A \vee \neg B \vee \neg C) \wedge (\neg A \vee B)$

◇ Every formula can be rewritten into an equivalent CNF (later).

Search for a model: CSP formulation

Propositional symbol = (boolean) variable
Model = boolean assignment
Formula = constraint to be satisfied

Heuristics DPLL: Davis–Putnam–Logemann–Loveland

- ♢ **early check** of satisfied or unsatisfiable formulae;
- ♢ **pure** symbols heuristics (always positive or negated);
- ♢ **unitary** formulae (only one literal);

DPLL

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
inputs: s, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in s
return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

if every clause in *clauses* is true in *model* **then return** *true*
if some clause in *clauses* is false in *model* **then return** *false*
P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [$P =$ *value*|*model*])
P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
if P is non-null **then return** DPLL(*clauses*, *symbols*–P, [$P =$ *value*|*model*])
P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
return DPLL(*clauses*, *rest*, [$P =$ *true*|*model*]) **or**
DPLL(*clauses*, *rest*, [$P =$ *false*|*model*])

DPLL (implementation)

With a number of additional details this 1963 procedure can be made very efficient!

- separation of disjoint problems
- variable and value ordering
- intelligent backtracking (clause learning)
- random restarts
- clever indexing

Local search: $GSAT$

Local search:

♢ start from a randomly chosen assignment

♢ compute best successor (i.e. change a variable to increase the number of satisfied clauses) and continue search

♢ if fail then restart from new randomly chosen assignment

Local search: $GSAT$

function GSAT(*formula max-restart max-search*) **returns** a truth assignment or failure

```
for i ← 1 to max-restart do
    A ← a random truth assignment
    for j ← 1 to max-search do
        if A satisfies formula then return A
        A ← a random choice of one of the best successors of A
    end
end
return failure
```

Local search is efficient, but incomplete.

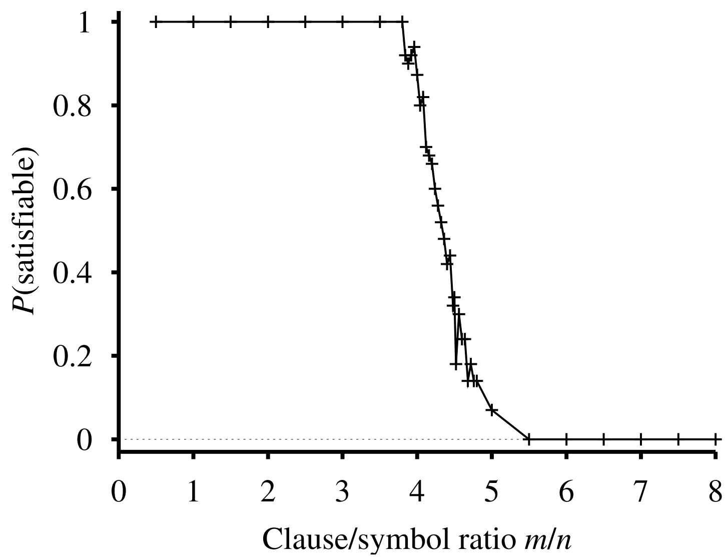
WALKSAT

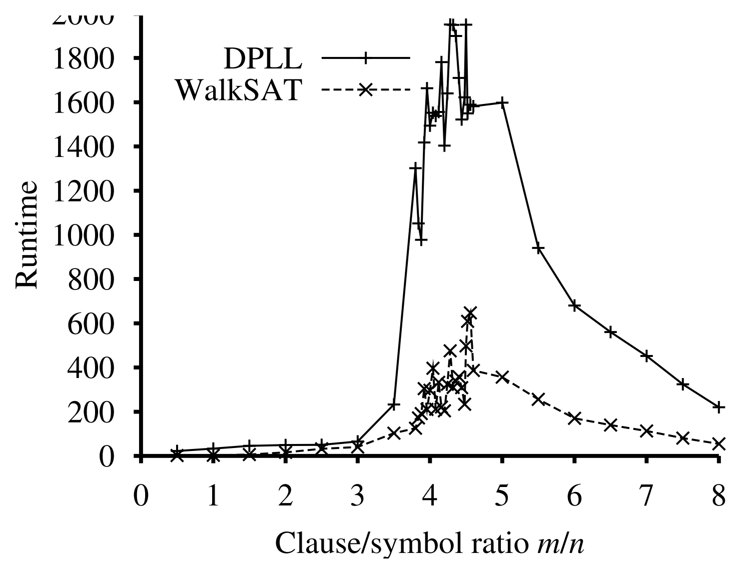
```
function WalkSAT(clauses, p, max-flips) returns a satisfying model or failure
   inputs: clauses, a set of clauses in propositional logic
           p, the probability of choosing to do a "random walk" move, around 0.5
           max-flips, number of flips allowed before giving up

   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
       if RANDOM(0,1) < p
        then flip the value in model of a randomly selected symbol from clause
        else flip whichever symbol in clause maximizes number of satisfied clauses
   return failure
```

Clauses unsatisfiable ?

Analysis of the search space

m = number of clauses
n = number of symbols (fixed to 50)
k = number of literals per clause (fixed to 3)





Difficult problems are around $m/n = 4.3$

Deduction in propositional logic

$$KB \vdash \alpha$$

Logical entailment can be computed by a syntactic process called **deduction** that manipulates (propositional) sentences.

Deduction can be characterized as a search:

- Init state is the formula representing the KB,
- the operators are the **inference rules**
- the final state is the formula to be proven

Inference Rules

♢ The inference rules $\mathcal{R}$ are typically written:

$$\frac{A_1 \cdots A_n}{A} \qquad\qquad \frac{premises}{conclusions}$$

♢ Some deductive systems include also Axioms Ax, but they can can be replaced by inference rules:

$$\frac{}{A}$$

for each $A \in Ax$

Theorems

$\diamondsuit$ Let Γ a set of formulae. A is derived from Γ ($\Gamma \vdash A$) if there exists a sequence of formulae $A_1, \ldots, A_n$ such that:

- A is A_n
- for every i between 1 and n, either $A_i \in \Gamma$ or A_i is a **direct derivation** of the formulae in $A_1, \ldots, A_{i-1}$.

The sequence $A_1, \ldots, A_n$ is a *proof* of A from Γ.

The elements of Γ are called *premises*, or *hypotheses*, or else *assumptions* of A.

Basic properties

$\diamondsuit$ A deduction method $\mathcal{R}$ is *sound* if for every formula A,

$$\vdash_{\mathcal{R}} A \text{ implies } \models A$$

$\diamondsuit$ A deduction method $\mathcal{R}$ is *complete* wrt a set of formulae Γ, if for every $A \in \Gamma$,

$$\models A \text{ implies } \vdash_{\mathcal{R}} A$$

$\diamondsuit$ Soundness and completeness:

$$\vdash \; \equiv \models$$

Deduction

$\Gamma \models A$ is the basic reasoning problem

- direct proof:
  $\Gamma \vdash_{\mathcal{R}} A$
- proof by refutation (reductio ad absurdum):
  $\Gamma \cup \{\neg A\}$ is unsatisfiable
  i.e. from $\Gamma \cup \{\neg A\}$ we get inconsistency

Inference rules for propositional logic

Modus Ponens

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

And Elimination

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

Examples

$$\frac{man \Rightarrow mortal, \qquad man}{mortal}$$

$\Gamma = \{feline \Rightarrow animal, cat \Rightarrow feline, cat\}$

$\Gamma \vdash_{MP} animal$

Deduction in propositional logic

"direct"

- Hilbert system (MP + axiom schemata): first, intuitive, not mechanizable;
- Natural Deduction Several Inference Rules

"refutation"

- Tableau: intuitive and mechanizable;
- **resolution**: born for automated deduction ... basis of PROLOG language

Forward/backward reasoning:
sound and complete (and polynomial) only for Horn clauses

Horn Clauses

Horn Clauses: with at most one positive literal

Definite Clauses: exactly one positive literal

Horn Form (restricted)

KB = *conjunction* of *definite clauses*

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

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

Forward and backward chaining

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

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

Can be used with:

- ◊ forward chaining or
- ◊ backward chaining.

These algorithms are very natural and run in *linear* time

Forward chaining

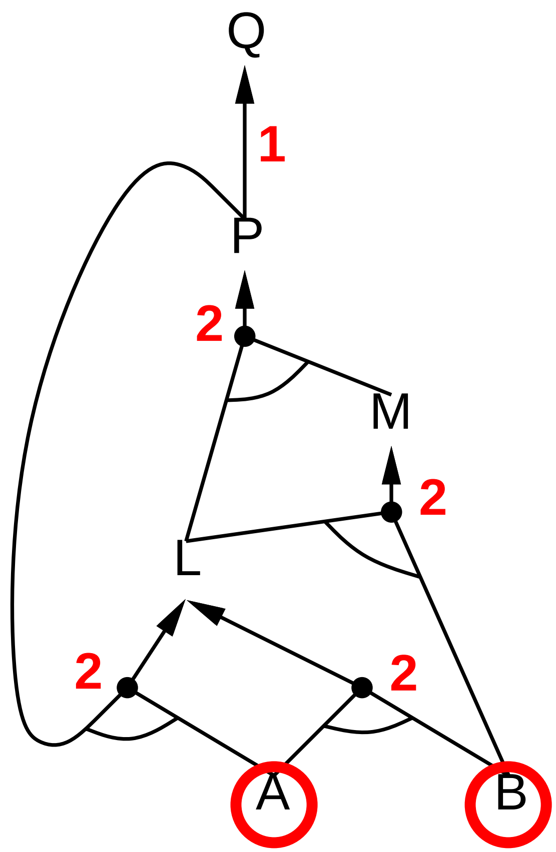
Idea: fire any rule whose premises are satisfied in the KB, add its conclusion to the KB, until query is found (e.g. Q)

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

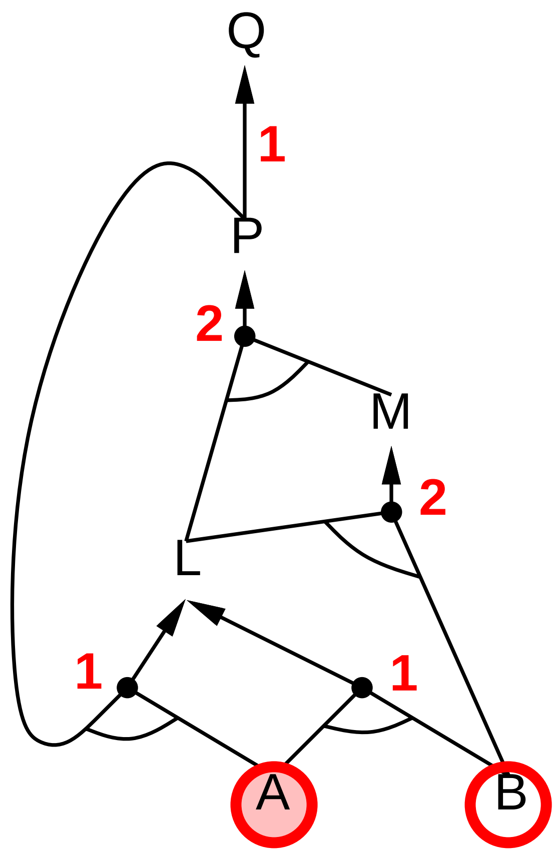
Forward chaining algorithm

```
function PL-FC-Entails?(KB, q) returns true or false
   inputs: KB, a set of propositional Horn clauses
           q, the query, a proposition symbol
   local variables: count, a table indexed by clause, init # premises
                    inferred, a table indexed by symbol, init false
                    agenda, a list of symbols, init 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
```

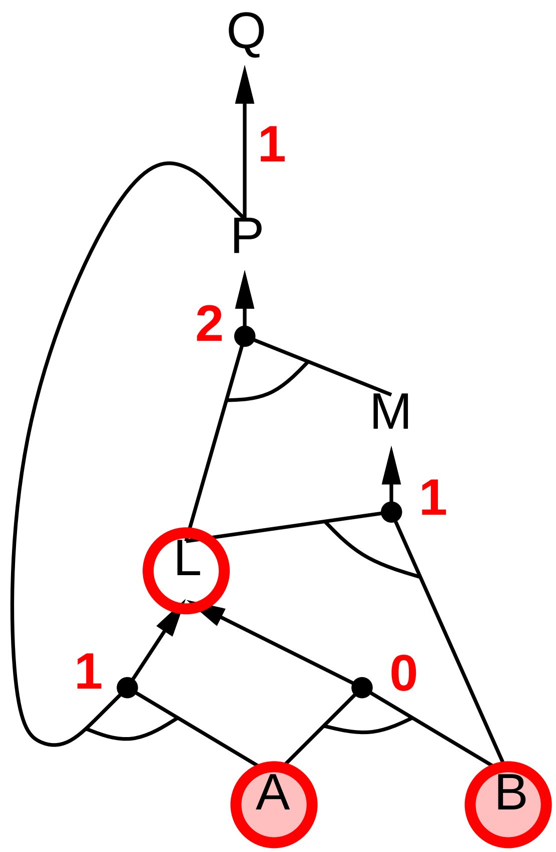
Forward chaining example



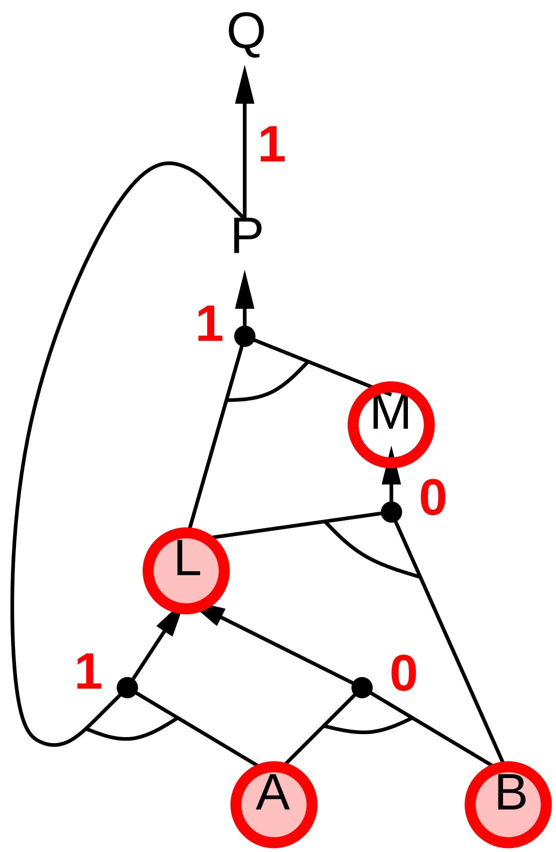
Forward chaining example



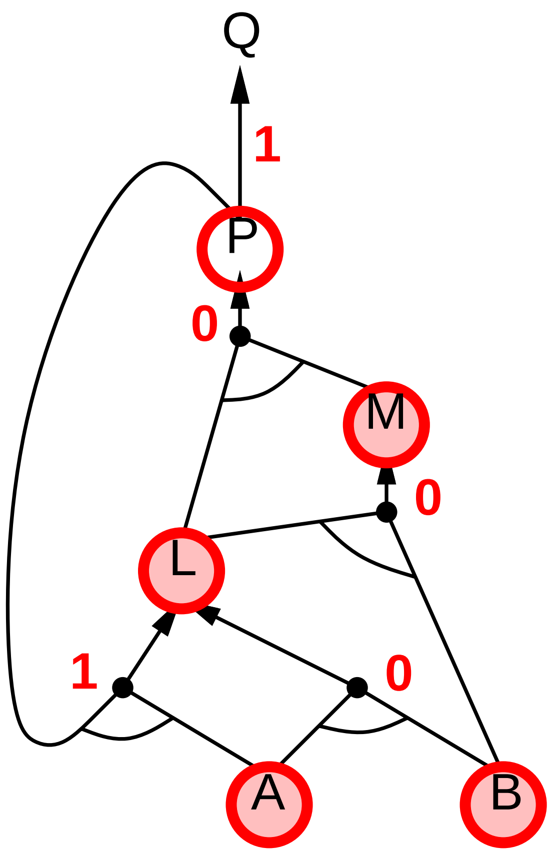
Forward chaining example



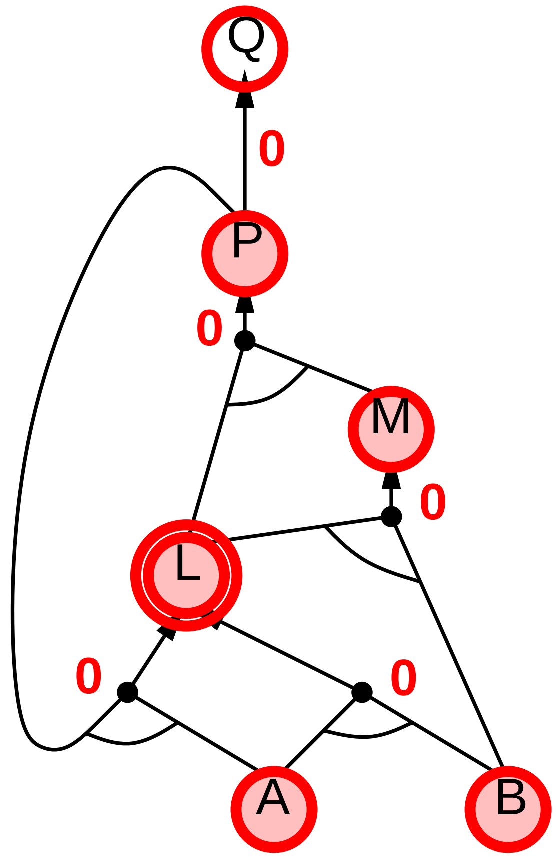
Forward chaining example



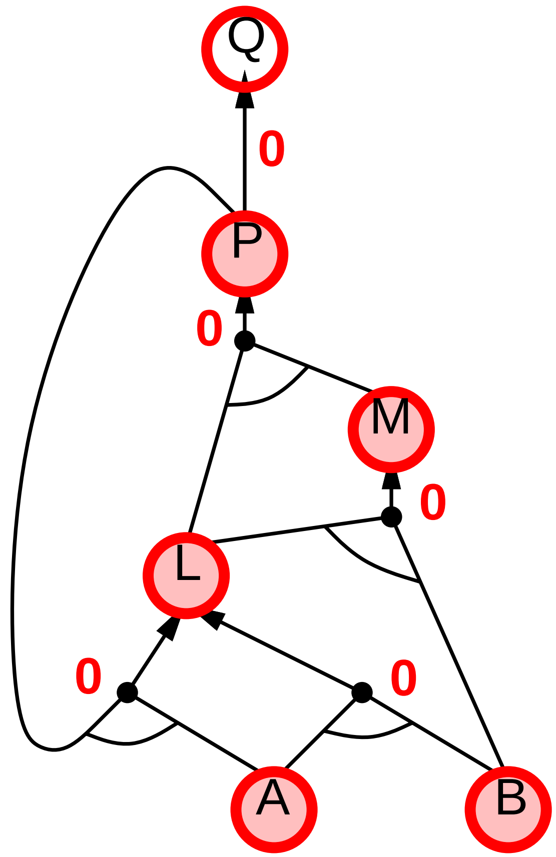
Forward chaining example



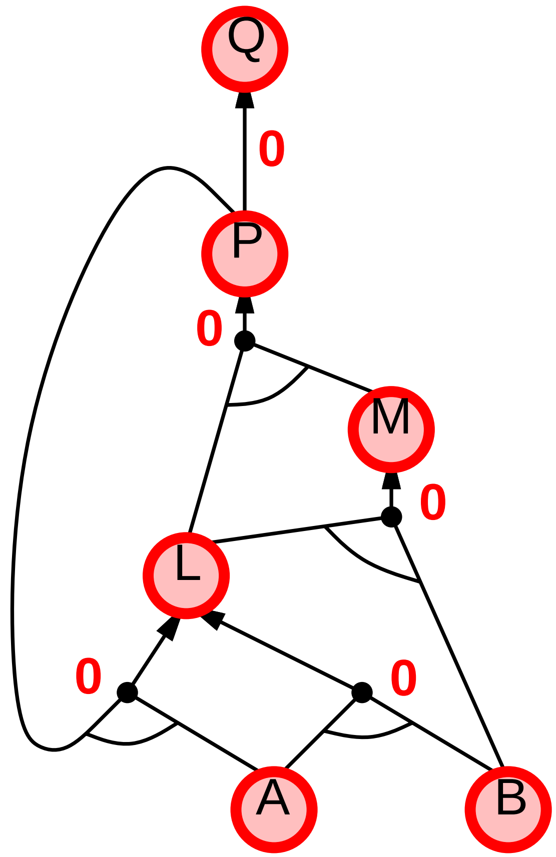
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a fixed point where no new atomic sentences are derived

2. Consider the final state as a model m, assigning $true$ to symbols derived in the KB and $false$ to the others

3. Every clause in the original KB is true in m
   Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m
   Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m
   Therefore the algorithm has not reached a fixed point!

4. Hence m is a model of KB

5. If $KB \models q$, q is true in *every* model of KB, including m

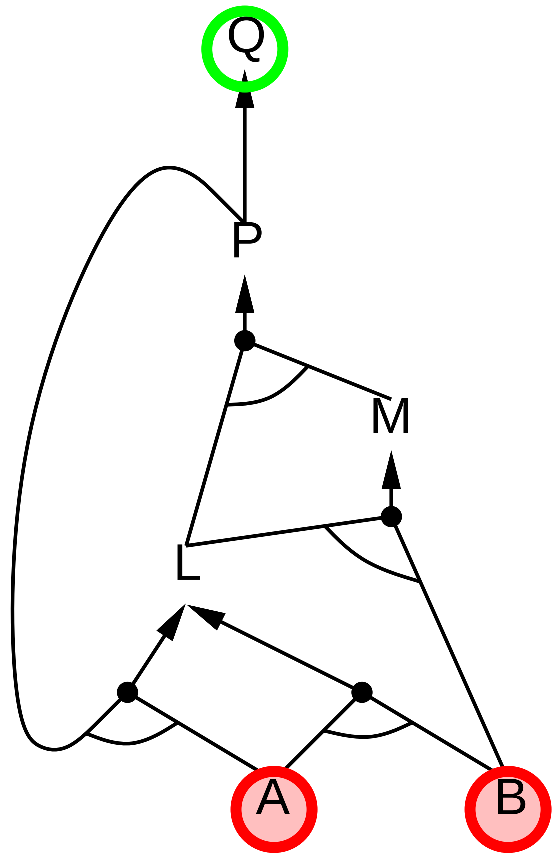
Backward chaining

Idea: work backwards from the query q:
  to prove q by BC,
    check if q is known already, or
    prove by BC all premises of some rule concluding q

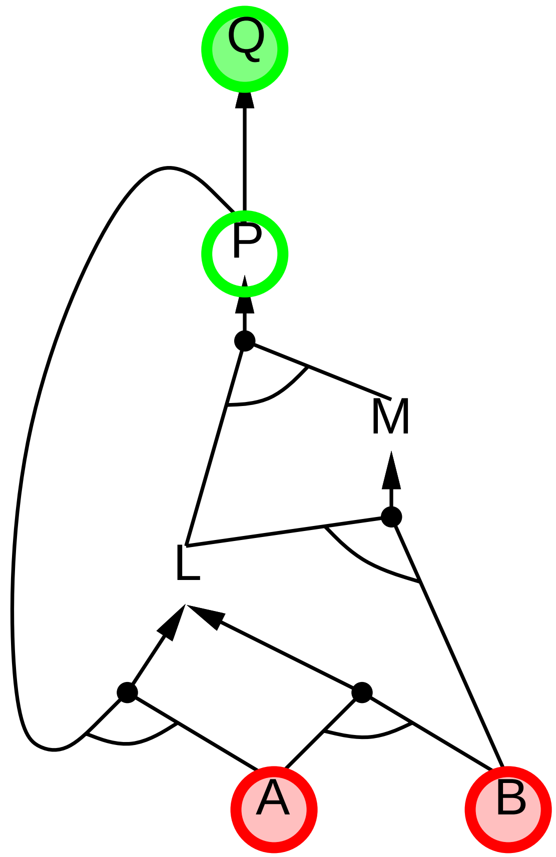
Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
  1) has already been proved true, or
  2) has already failed

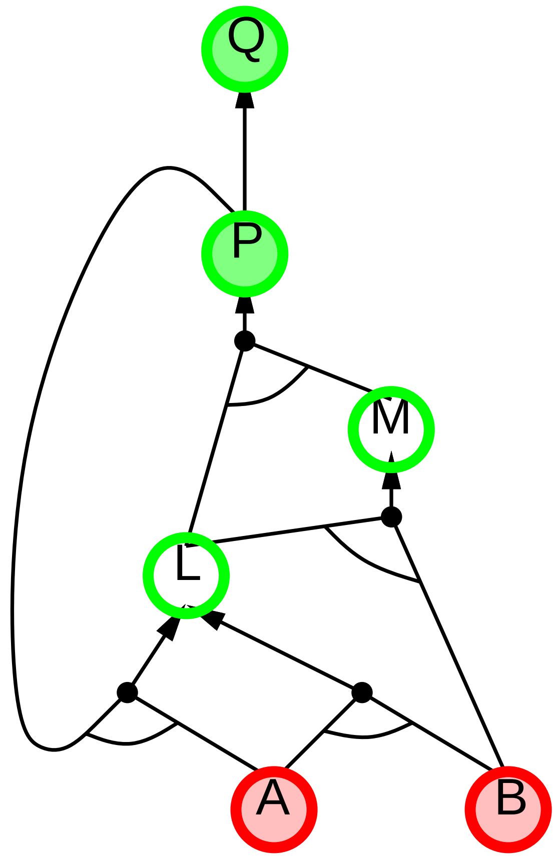
Backward chaining example



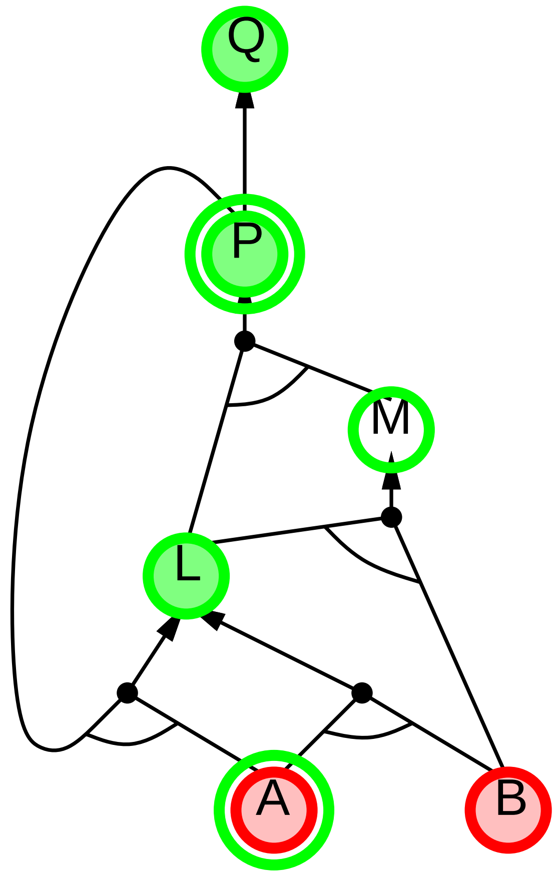
Backward chaining example



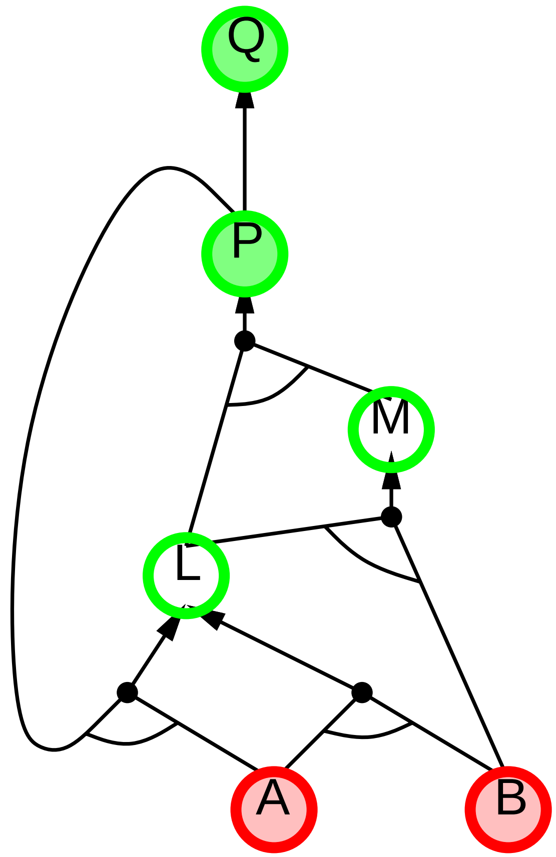
Backward chaining example



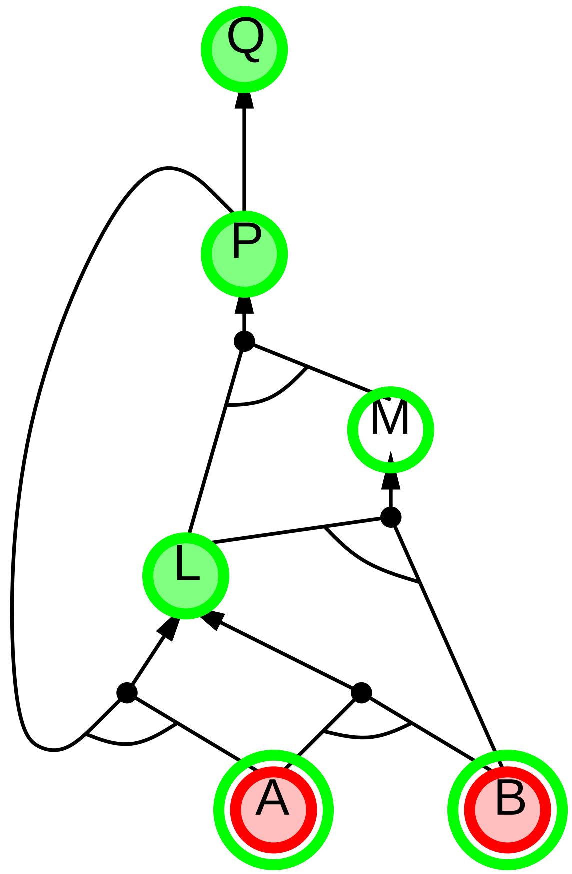
Backward chaining example



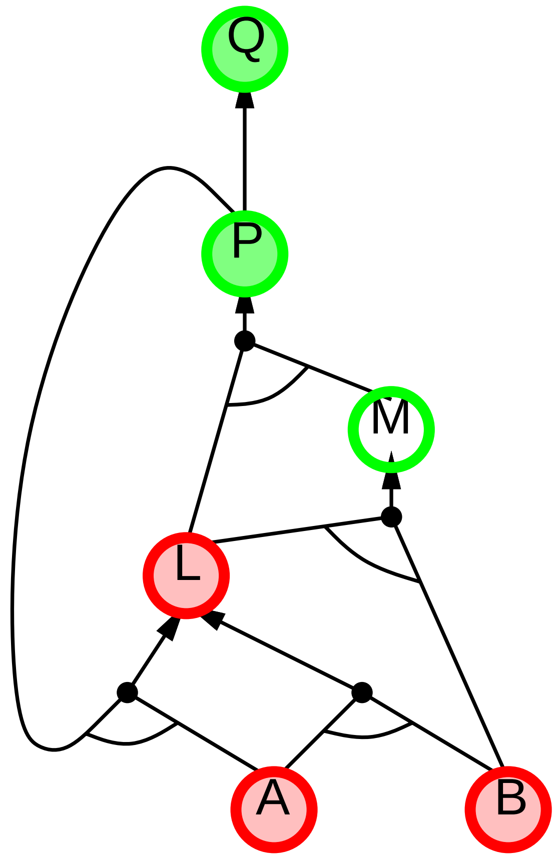
Backward chaining example



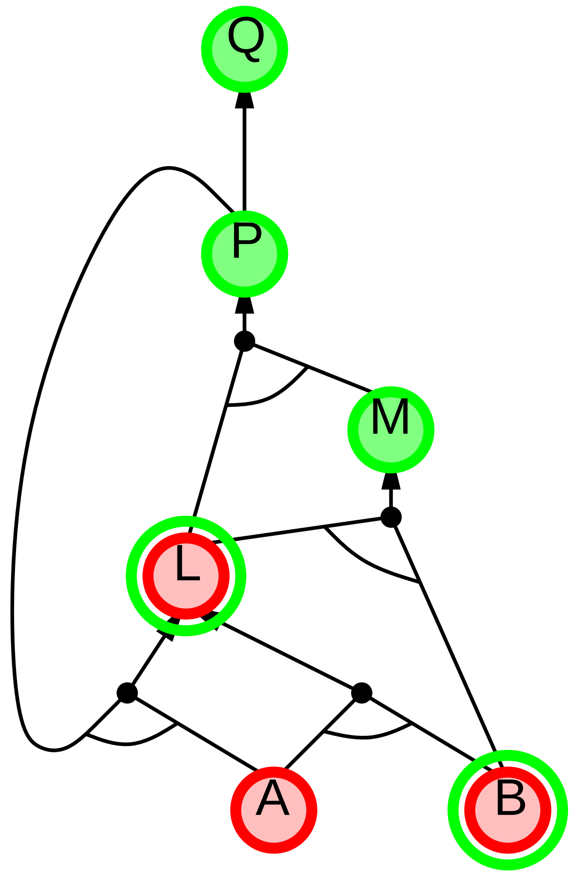
Backward chaining example



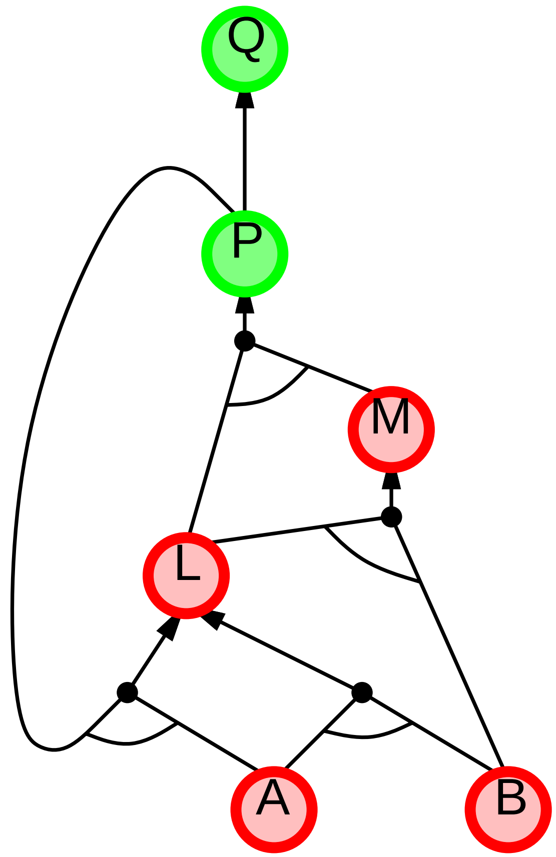
Backward chaining example



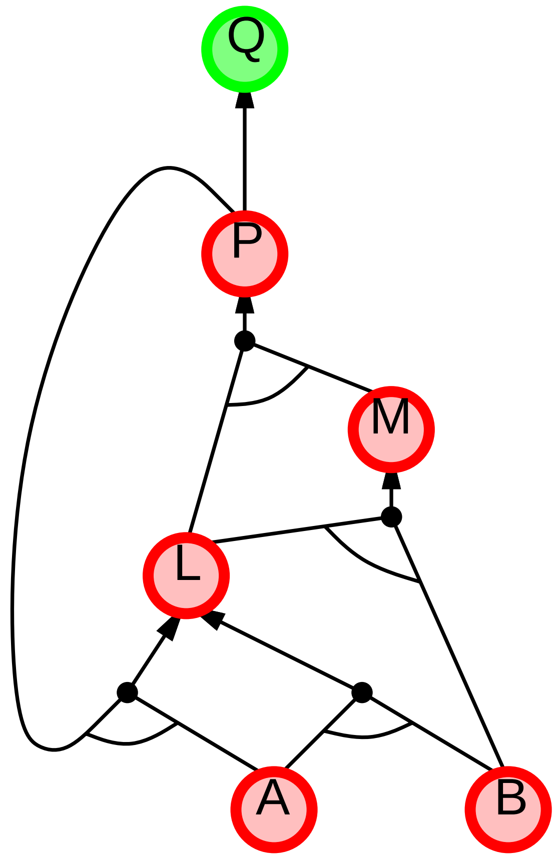
Backward chaining example



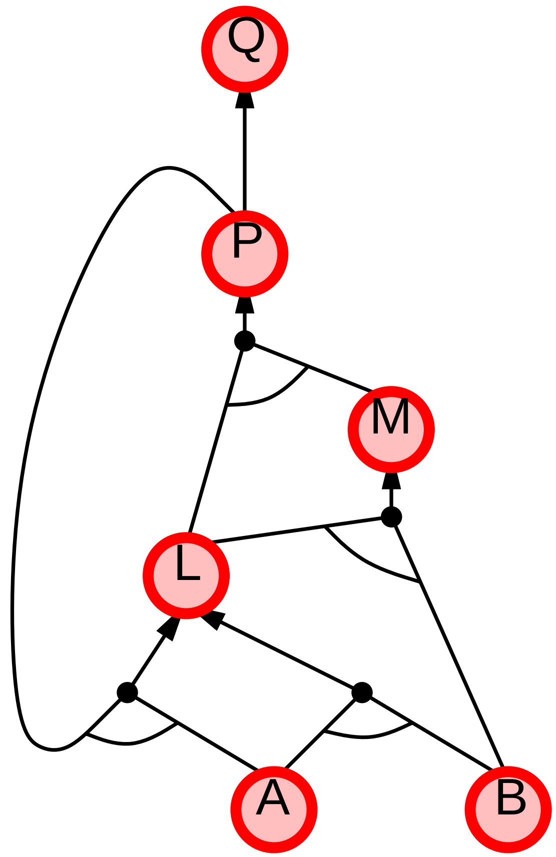
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

FC is data-driven, cf. 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

Summary

- syntax and semantics of propositional logic
- model checking
- inference
- efficient model checking
- efficient inference (with Horn clauses)