CSC 481: Horn Clauses

Rodrigo Canaan
Assistant Professor
Computer Science Department
Cal Poly, San Luis Obispo
rcanaan@calpoly.edu

Motivation

Motivation

- Resolution is powerful, but inefficient in the worst case

Motivation

- Resolution is powerful, but inefficient in the worst case
- While refinements to Resolution or other inference procedures may help in some cases, the problem of first-order entailment is inherently intractable

Motivation

- Resolution is powerful, but inefficient in the worst case
- While refinements to Resolution or other inference procedures may help in some cases, the problem of first-order entailment is inherently intractable
- Horn Clauses are less expressive than full First-Order Logic, but there exist very efficient algorithms to check entailment

Motivation

- Resolution is powerful, but inefficient in the worst case
- While refinements to Resolution or other inference procedures may help in some cases, the problem of first-order entailment is inherently intractable
- Horn Clauses are less expressive than full First-Order Logic, but there exist very efficient algorithms to check entailment
- We will start by studying resolution with Horn clauses in the context of propositional logic, then briefly talk about statements involving predicates, functions and equality.

Horn Clauses

Definition

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  B) Negative Clauses

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  B) Negative Clauses

Horn Clauses

Definition

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

Horn Clauses
Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

Horn Clauses
Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  - Clauses with one positive literal and one or more negative literals (“rules”)

Horn Clauses
Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  - Clauses with one positive literal and one or more negative literals (“rules”)
    - Ex: $\neg a \vee \neg b \vee \neg c \vee d$

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  - Clauses with one positive literal and one or more negative literals (“rules”)
    - Ex: $\neg a \vee \neg b \vee \neg c \vee d$
    - Can be rewritten as an implication: $a \wedge b \wedge c \rightarrow d$

Horn Clauses
Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  - Clauses with one positive literal and one or more negative literals (“rules”)
    - Ex: $\neg a \vee \neg b \vee \neg c \vee d$
    - Can be rewritten as an implication: $a \wedge b \wedge c \rightarrow d$
    - The negative literals make the **body** or **premises** of the rule, the positive literal makes the **head** or **conclusion**

Horn Clauses
Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  - Clauses with one positive literal and one or more negative literals (“rules”)
    - Ex: $\neg a \vee \neg b \vee \neg c \vee d$
    - Can be rewritten as an implication: $a \wedge b \wedge c \to d$
    - The negative literals make the **body** or **premises** of the rule, the positive literal makes the **head** or **conclusion**
  - Clauses with one positive literal and no negative literals (“facts”)

Horn Clauses
Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

A) Positive Clauses

- Clauses with one positive literal and one or more negative literals ("rules")
  - Ex: $\neg a \lor \neg b \lor \neg c \lor d$
  - Can be rewritten as an implication: $a \land b \land c \rightarrow d$
  - The negative literals make the **body** or **premises** of the rule, the positive literal makes the **head** or **conclusion**
- Clauses with one positive literal and no negative literals ("facts")
  - Ex: a

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  A) Positive Clauses

  - Clauses with one positive literal and one or more negative literals ("rules")
    - Ex: $\neg a \vee \neg b \vee \neg c \vee d$
    - Can be rewritten as an implication: $a \wedge b \wedge c \rightarrow d$
    - The negative literals make the **body** or **premises** of the rule, the positive literal makes the **head** or **conclusion**
  - Clauses with one positive literal and no negative literals ("facts")
    - Ex: a

Horn Clauses
Definition

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

B) Negative Clauses

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  B) Negative Clauses

  - Clauses with no positive literals (“goal clauses”)

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  B) Negative Clauses

  - Clauses with no positive literals (“goal clauses”)
    - Ex: $\neg d \vee \neg e$

Horn Clauses

Definition

Horn Clauses are clauses with *at most* one positive literal

- Types of Horn Clauses:

  B) Negative Clauses

  - Clauses with no positive literals ("goal clauses")
    - Ex: $\neg d \vee \neg e$
    - The empty clause [] is a special case of goal clause

What is *not* a Horn clause?

What is *not* a Horn clause?

Clauses with more than one positive literal, used to express incomplete knowledge

What is *not* a Horn clause?

Clauses with more than one positive literal, used to express incomplete knowledge

- Examples

What is *not* a Horn clause?

Clauses with more than one positive literal, used to express incomplete knowledge

- Examples
  - $sun \lor rain$

What is *not* a Horn clause?

Clauses with more than one positive literal, used to express incomplete knowledge

- Examples
  - $sun \vee rain$
  - $beach \rightarrow coconut \vee ice\_cream$

Why Horn clauses?

Why Horn clauses?

- Derivations follow an intuitive format: try to derive new facts by satisfying the premises of rules

Why Horn clauses?

- Derivations follow an intuitive format: try to derive new facts by satisfying the premises of rules
  - Note that if we tried to follow this intuition with clauses denoting incomplete knowledge, we might have to “branch out”

Why Horn clauses?

- Derivations follow an intuitive format: try to derive new facts by satisfying the premises of rules
  - Note that if we tried to follow this intuition with clauses denoting incomplete knowledge, we might have to “branch out”
- Can be done via:

Why Horn clauses?

- Derivations follow an intuitive format: try to derive new facts by satisfying the premises of rules
  - Note that if we tried to follow this intuition with clauses denoting incomplete knowledge, we might have to “branch out”
- Can be done via:
  - Forward chaining: from premises to conclusions

Why Horn clauses?

- Derivations follow an intuitive format: try to derive new facts by satisfying the premises of rules
  - Note that if we tried to follow this intuition with clauses denoting incomplete knowledge, we might have to “branch out”
- Can be done via:
  - Forward chaining: from premises to conclusions
  - Backward chaining: from a desired conclusion to its preconditions

Why Horn clauses?

Why Horn clauses?

- Also some important formal properties:

Why Horn clauses?

- Also some important formal properties:
  - Resolving a Horn clause yields another Horn clause

Why Horn clauses?

- Also some important formal properties:
  - Resolving a Horn clause yields another Horn clause
    - Negative clauses can never be resolved together

Why Horn clauses?

- Also some important formal properties:
  - Resolving a Horn clause yields another Horn clause
    - Negative clauses can never be resolved together
    - Two positive clauses resolve into another positive clause

Why Horn clauses?

- Also some important formal properties:
  - Resolving a Horn clause yields another Horn clause
    - Negative clauses can never be resolved together
    - Two positive clauses resolve into another positive clause
    - A positive and a negative clause resolve into a negative clause

Why Horn clauses?

- Also some important formal properties:
  - Resolving a Horn clause yields another Horn clause
    - Negative clauses can never be resolved together
    - Two positive clauses resolve into another positive clause
    - A positive and a negative clause resolve into a negative clause
  - SLD derivation (see next slide)

SLD derivation

SLD derivation

- If S is a set of Horn clauses and α any negative clause (including possibly the empty clause) entailed by S, there is an SLD derivation $c_1, c_2, \ldots, c_n$ where:

SLD derivation

- If S is a set of Horn clauses and α any negative clause (including possibly the empty clause) entailed by S, there is an SLD derivation $c_1, c_2, \ldots, c_n$ where:
  - c_i is negative for all i

SLD derivation

- If S is a set of Horn clauses and α any negative clause (including possibly the empty clause) entailed by S, there is an SLD derivation $c_1, c_2, \ldots, c_n$ where:
    - c_i is negative for all i
    - $c_1 \in S$

SLD derivation

- If S is a set of Horn clauses and α any negative clause (including possibly the empty clause) entailed by S, there is an SLD derivation $c_1, c_2, \ldots, c_n$ where:
  - c_i is negative for all i
  - $c_1 \in S$
  - $c_n = \alpha$

SLD derivation

- If S is a set of Horn clauses and α any negative clause (including possibly the empty clause) entailed by S, there is an SLD derivation $c_1, c_2, \ldots, c_n$ where:
  - c_i is negative for all i
  - $c_1 \in S$
  - $c_n = \alpha$
  - c_{i+1} resolves from c_i and some clause in S

SLD derivation

SLD derivation

SLD derivation

- In other words, we get a “chain” where each new clause c_i resolved with some clause originally in S to produce the next clause c_{i+1}

SLD derivation

- In other words, we get a “chain” where each new clause c_i resolved with some clause originally in S to produce the next clause c_{i+1}
- SLD stands for Selected literals, Linear pattern over Definite clauses

SLD derivation

- In other words, we get a “chain” where each new clause c_i resolved with some clause originally in S to produce the next clause c_{i+1}
- SLD stands for Selected literals, Linear pattern over Definite clauses
- Every proof in Horn clauses admits an SLD derivation

SLD derivation

- In other words, we get a “chain” where each new clause c_i resolved with some clause originally in S to produce the next clause c_{i+1}
- SLD stands for Selected literals, Linear pattern over Definite clauses
- Every proof in Horn clauses admits an SLD derivation
  - But not every correct derivation is SLD

Example

1. Consider the following KB and try to prove $KB \vdash E$:

   1. $A \rightarrow B$

   2. $C \rightarrow D$

   3. $B \wedge D \rightarrow E$

   4. A

   5. C

   6. $\neg E$

Example

1. Consider the following KB and try to prove $KB \vdash E$:

1. $A \rightarrow B$
2. $C \rightarrow D$
3. $B \wedge D \rightarrow E$
4. A
5. C
6. $\neg E$

SLD derivation:

7. $\neg B \vee \neg D$ (3,6)
8. $\neg B \vee \neg C$ (2,7)
9. $\neg B$ (5,8)
10. $\neg A$ (1,9)
11. [] (4,10)

Example

1.Consider the following KB and try to prove $KB \vdash E$:

1. $A \rightarrow B$
2. $C \rightarrow D$
3. $B \wedge D \rightarrow E$
4. A
5. C
6. $\neg E$

Non SLD derivation:

7. B (1,4)
8. D (2,5)
9. $\neg B \vee \neg D$ (3,6)
10. $\neg D$ (7,9)
11. [] (8,10)

Example

1. Consider the following KB and try to prove $KB \vdash E$:

KB	SLD derivation:		Non SLD derivation:	
1. $A \to B$	7. $\neg B \vee \neg D$	(3,6)	7. B	(1,4)
2. $C \to D$	8. $\neg B \vee \neg C$	(2,7)	8. D	(2,5)
3. $B \wedge D \to E$	9. $\neg B$	(5,8)	9. $\neg B \vee \neg D$	(3,6)
4. A	10. $\neg A$	(1,9)	10. $\neg D$	(7,9)
5. C	11. []	(4,10)	11. []	(8,10)
6. $\neg E$				

Example

1.Consider the following KB and try to prove $KB \vdash E$:

1.$A \rightarrow B$

2.$C \rightarrow D$

3.$B \wedge D \rightarrow E$

4.A

5.C

6.$\neg E$

SLD derivation:

7. $\neg B \vee \neg D$	(3,6)
8. $\neg B \vee \neg C$	(2,7)
9. $\neg B$	(5,8)
10. $\neg A$	(1,9)
11.[]	(4,10)

Non SLD derivation:

7. B	(1,4)
8. D	(2,5)
9. $\neg B \vee \neg D$	(3,6)
10. $\neg D$	(7,9)
11.[]	(8,10)

Exercise

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

  1. $P \rightarrow Q$
  2. $L \wedge M \rightarrow P$
  3. $B \wedge L \rightarrow M$
  4. $A \wedge P \rightarrow L$
  5. $A \wedge B \rightarrow L$
  6. A
  7. B

Exercise

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$
6. A
7. B

A. By forward chaining: start with the facts A and B and try to produce Q

Exercise

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$
6. A
7. B

A. By forward chaining: start with the facts A and B and try to produce Q

    A. Only resolve facts with a negative of other clauses

Exercise

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$
6. A
7. B

A. By forward chaining: start with the facts A and B and try to produce Q
   A. Only resolve facts with a negative of other clauses

B. By backward chaining: start with the negated query $\neg Q$ and try to produce the empty clause t

Exercise

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \land M \rightarrow P$
3. $B \land L \rightarrow M$
4. $A \land P \rightarrow L$
5. $A \land B \rightarrow L$
6. A
7. B

A. By forward chaining: start with the facts A and B and try to produce Q

   A. Only resolve facts with a negative of other clauses

B. By backward chaining: start with the negated query $\neg Q$ and try to produce the empty clause t

   A. Only resolve goals with the positive of other clauses

Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

  1. $P \to Q$
  2. $L \wedge M \to P$
  3. $B \wedge L \to M$
  4. $A \wedge P \to L$
  5. $A \wedge B \to L$
  6. A
  7. B

Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $A \wedge B \rightarrow L$
6. A
7. B
8. $\neg Q$ (query)

Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $B \wedge L \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $\cancel{A} \wedge \cancel{B} \rightarrow L$
6. A
7. B

$8. \neg Q$ (query)

$9. L$ (5,6,7)

Note: we could also do this in two steps, writing first

$8. B \rightarrow L (5,6)$

Then

$9. L (7,8)$

Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $L \wedge M \rightarrow P$
3. $\cancel{B} \wedge \cancel{L} \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $\cancel{A} \wedge \cancel{B} \rightarrow L$
6. A
7. B

8. $\neg Q$ (query)
9. L (5,6,7)
10. M (3,7,9)

Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $P \rightarrow Q$
2. $\cancel{L} \wedge \cancel{M} \rightarrow P$
3. $\cancel{B} \wedge \cancel{L} \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $\cancel{A} \wedge \cancel{B} \rightarrow L$
6. A
7. B

8. $\neg Q$ (query)
9. L (5,6,7)
10. M (3,7,9)
11. P (2,9,10)

Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\cancel{P} \rightarrow Q$
2. $\cancel{L} \wedge \cancel{M} \rightarrow P$
3. $\cancel{B} \wedge \cancel{L} \rightarrow M$
4. $A \wedge P \rightarrow L$
5. $\cancel{A} \wedge \cancel{B} \rightarrow L$
6. A
7. B

8. $\neg Q$ (query)
9. L (5,6,7)
10. M (3,7,9)
11. P (2,9,10)
12. Q (1,11)

Note: Here, it was possible for us to derive L again from 4,6,11, but an efficient algorithm should check that this fact had been generated before and not waste more time processing it.

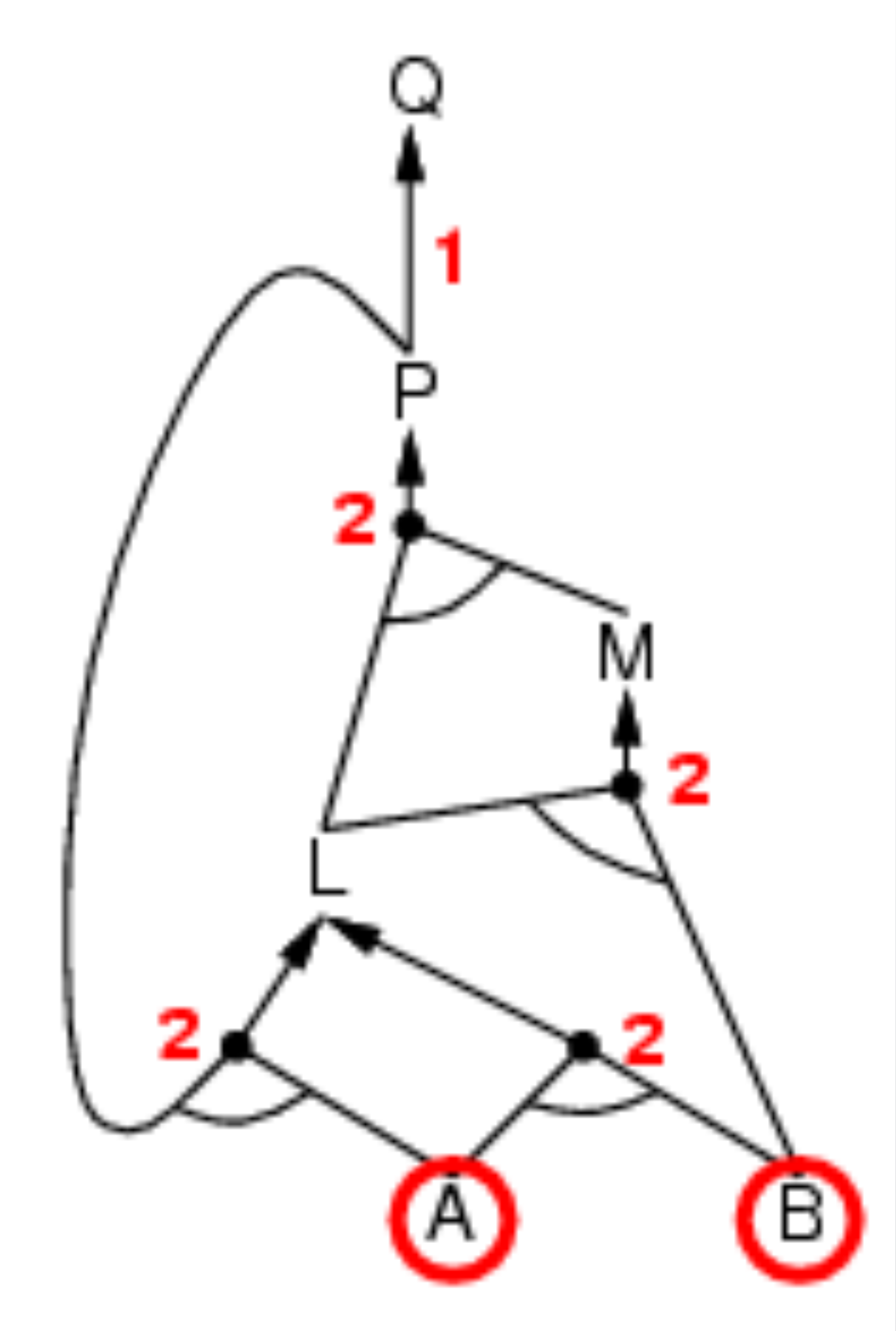
Forward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

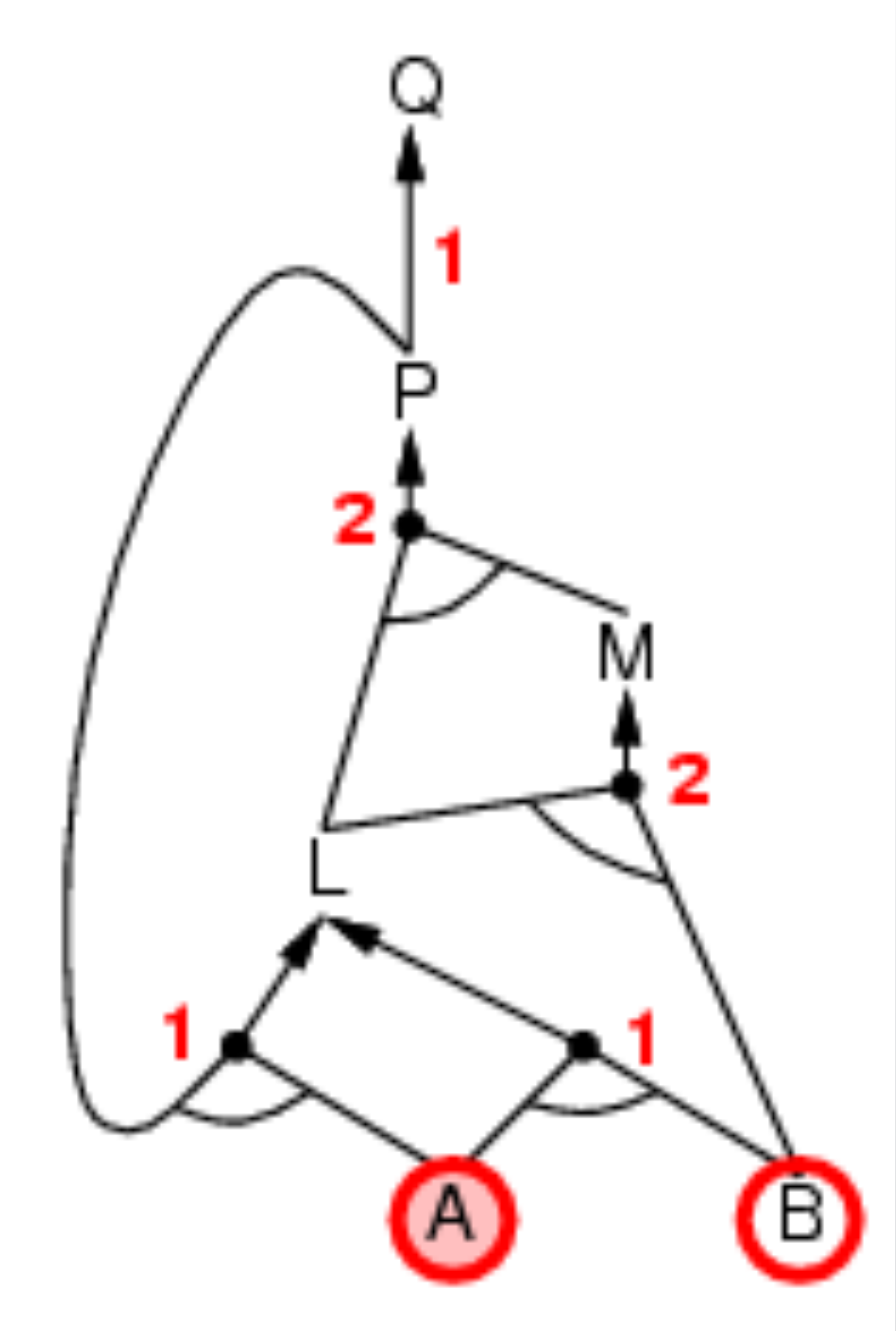
1. $\cancel{P} \to Q$
2. $\cancel{L} \wedge \cancel{M} \to P$
3. $\cancel{B} \wedge \cancel{L} \to M$
4. $A \wedge P \to L$
5. $\cancel{A} \wedge \cancel{B} \to L$
6. A
7. B

8. $\neg Q$	(query)
9. L	(5,6,7)
10. M	(3,7,9)
11. P	(2,9,10)
12. Q	(1,11)
13. []	(8,12)

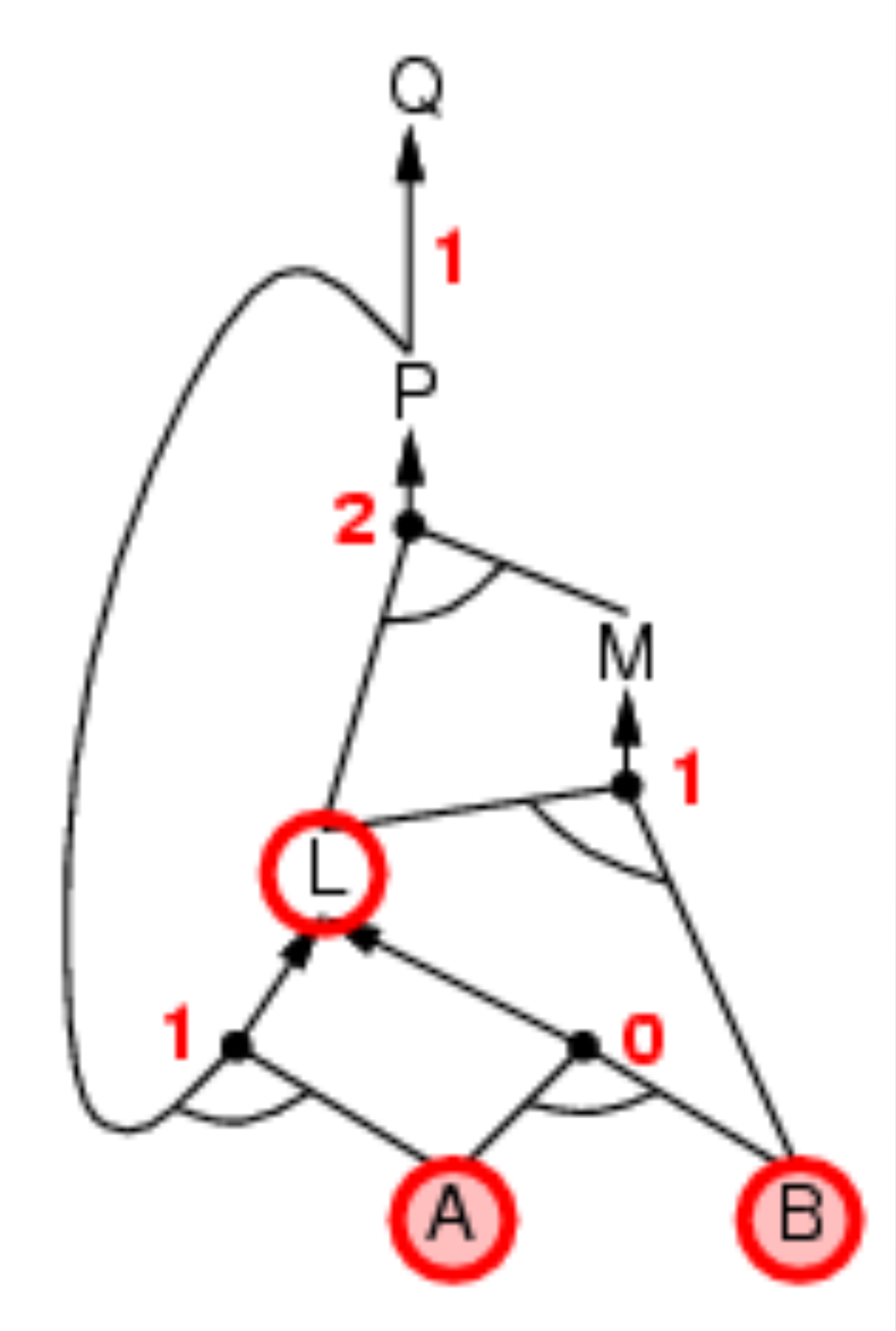
Forward chaining example



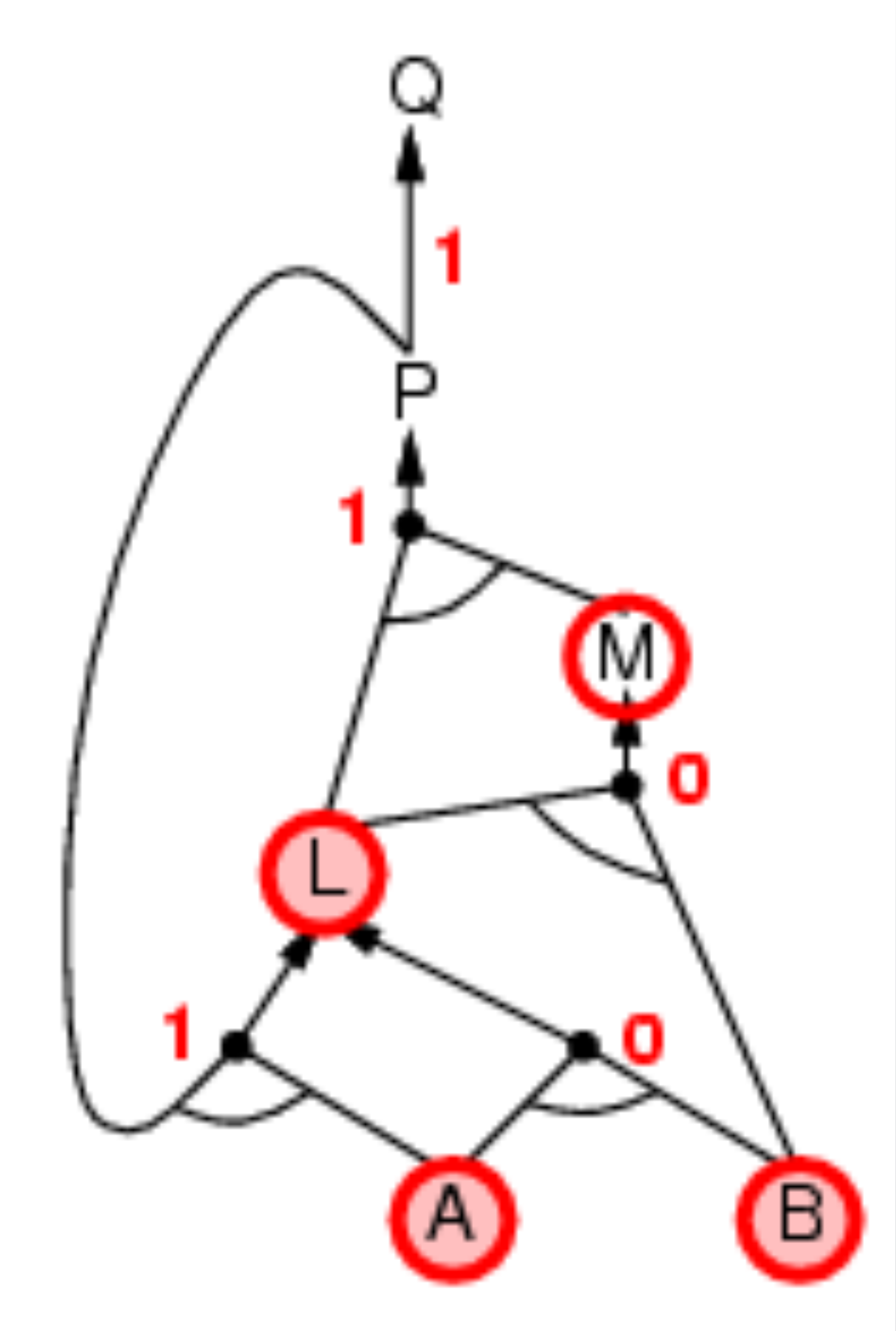
Forward chaining example



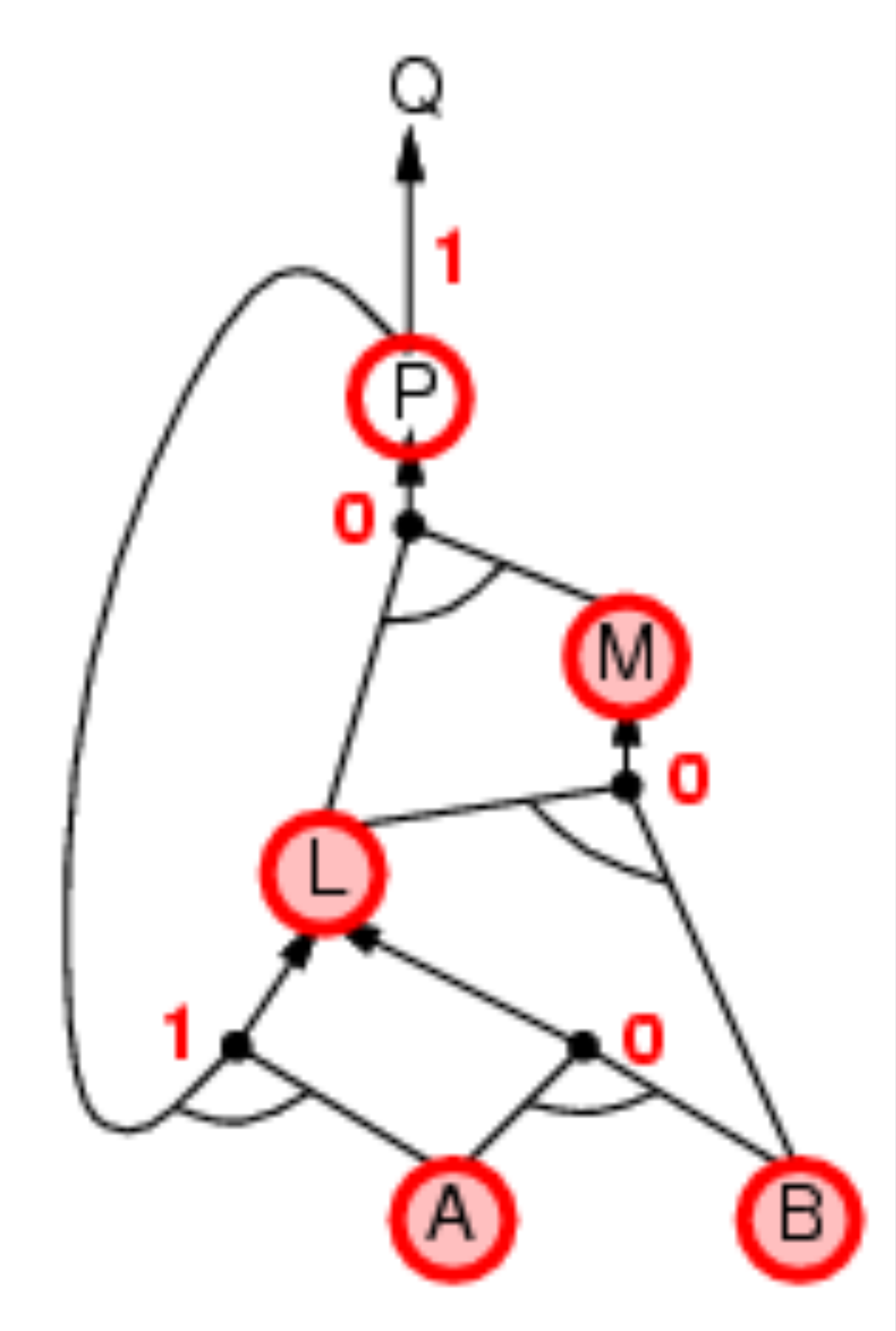
Forward chaining example



Forward chaining example

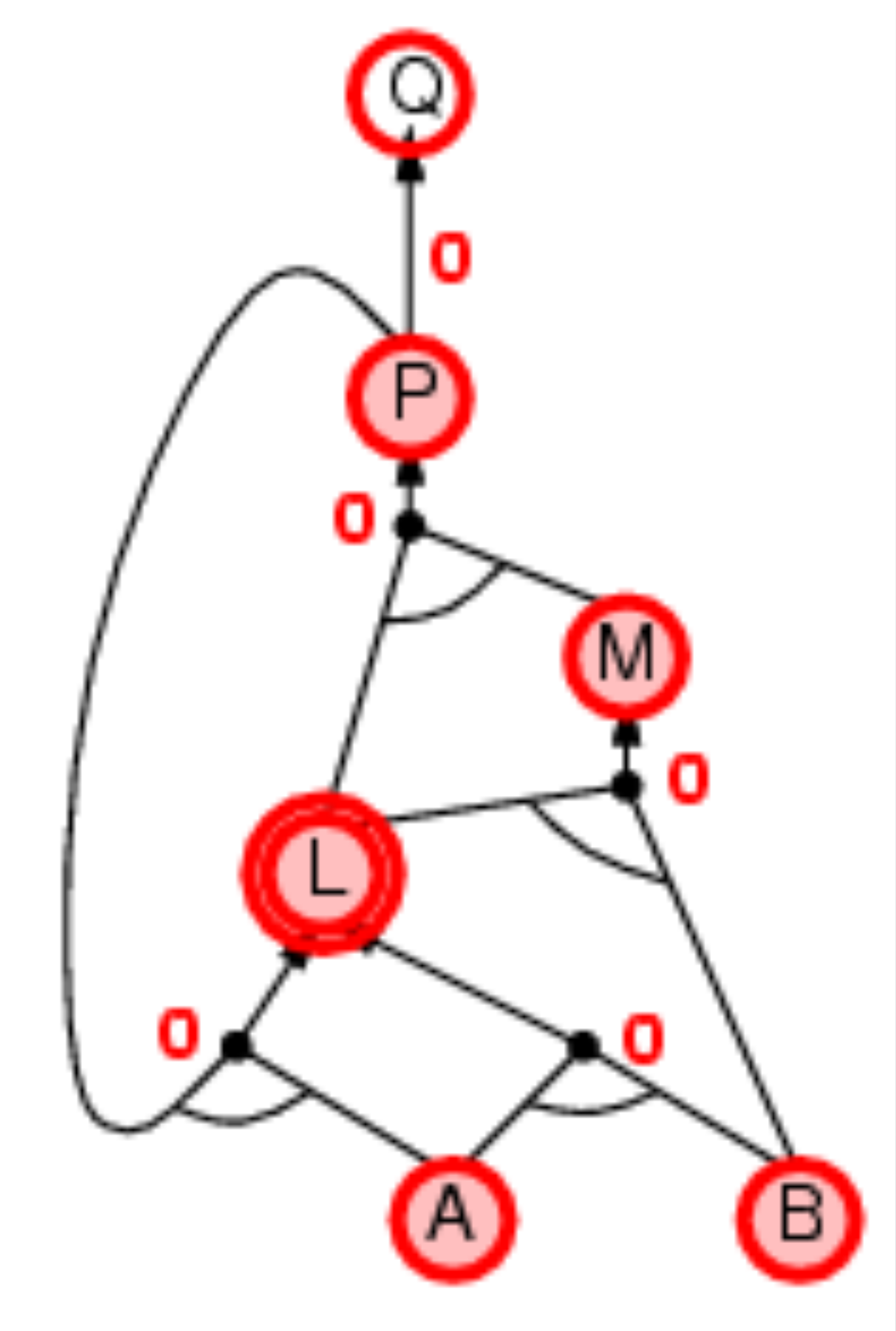


Forward chaining example



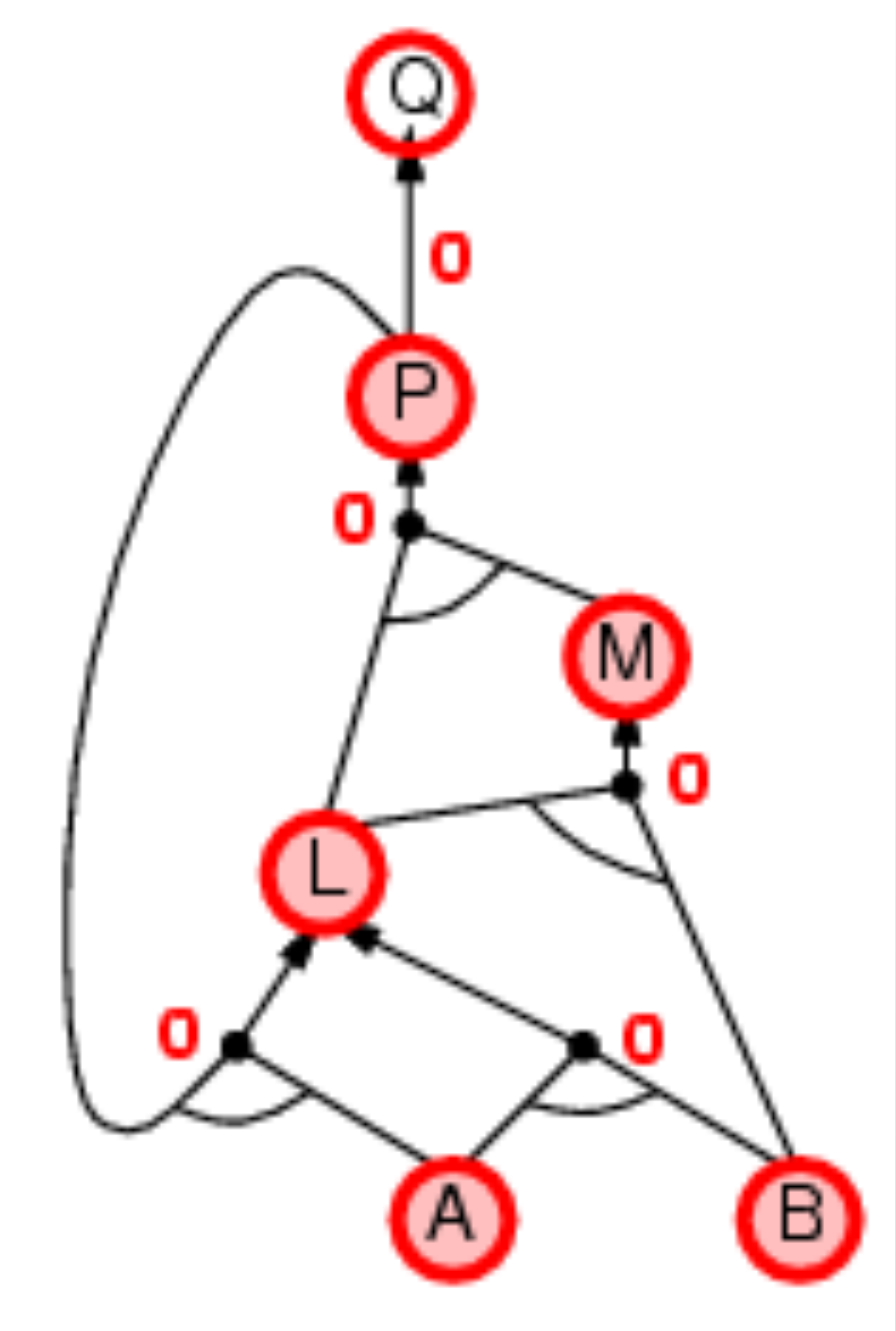
http://aima.eecs.berkeley.edu/slides-ppt/

Forward chaining example

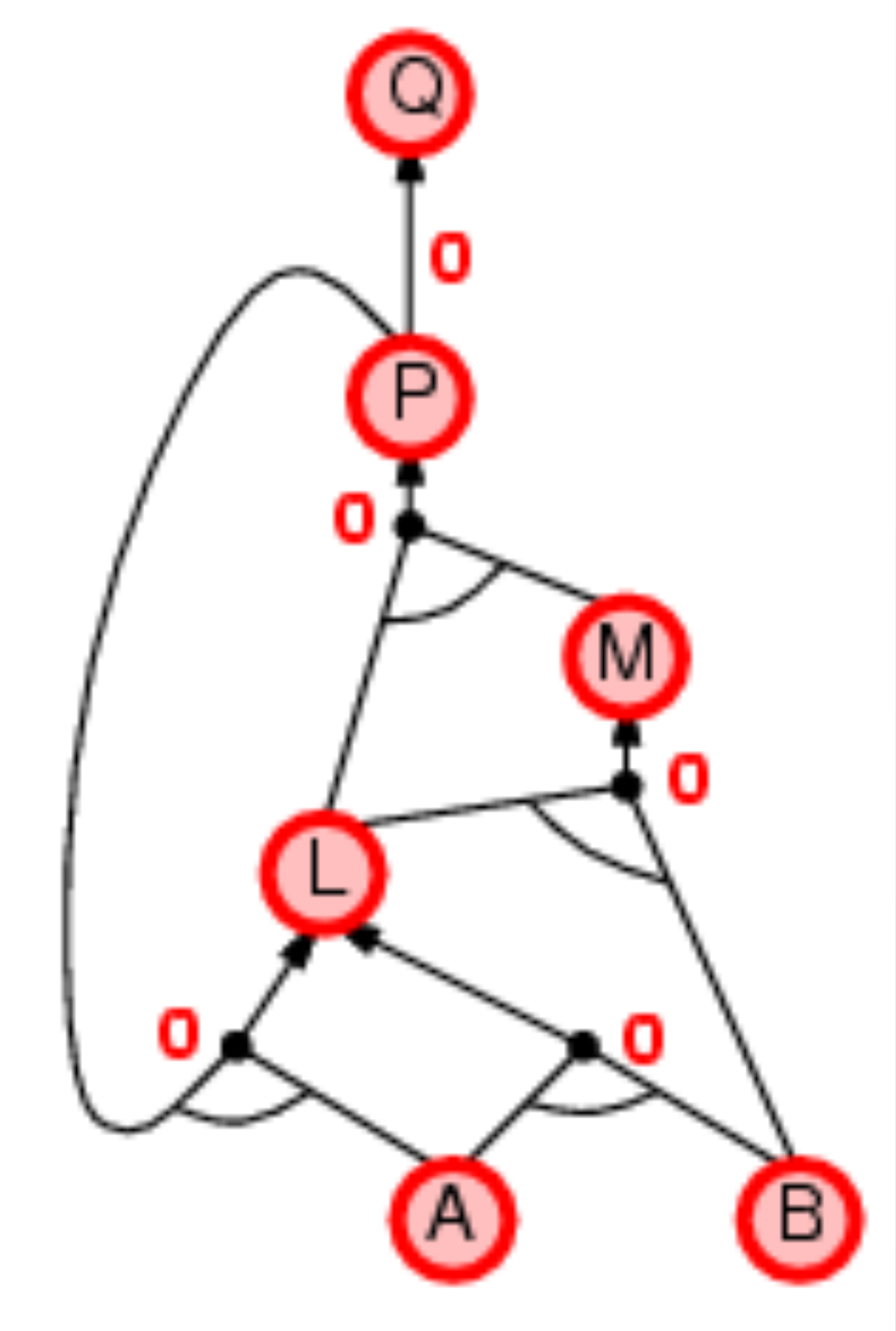


http://aima.eecs.berkeley.edu/slides-ppt/

Forward chaining example



Forward chaining example



Backward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\neg P \lor Q$
2. $\neg L \lor \neg M \lor P$
3. $\neg B \lor \neg L \lor M$
4. $\neg A \lor \neg P \lor L$
5. $\neg A \lor \neg B \lor L$
6. A
7. B

8. $\neg Q$ (query)

Backward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\neg P \lor \cancel{Q}$
2. $\neg L \lor \neg M \lor P$
3. $\neg B \lor \neg L \lor M$
4. $\neg A \lor \neg P \lor L$
5. $\neg A \lor \neg B \lor L$
6. A
7. B

8. $\neg Q$ (query)
9. $\neg P$ (1,8)

Backward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\neg P \vee \cancel{Q}$
2. $\neg L \vee \neg M \vee \cancel{P}$
3. $\neg B \vee \neg L \vee M$
4. $\neg A \vee \neg P \vee L$
5. $\neg A \vee \neg B \vee L$
6. A
7. B

8. $\neg Q$ (query)
9. $\neg P$ (1,8)
10. $\neg L \vee \neg M$ (2,9)

Backward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\neg P \vee \cancel{Q}$
2. $\neg L \vee \neg M \vee \cancel{P}$
3. $\neg B \vee \neg L \vee \cancel{M}$
4. $\neg A \vee \neg P \vee L$
5. $\neg A \vee \neg B \vee L$
6. A
7. B

8. $\neg Q$ (query)
9. $\neg P$ (1,8)
10. $\neg L \vee \neg M$ (2,9)
11. $\neg B \vee \neg L$ (3,10)

Note: resolving (5,10) actually gives us)
$\neg B \vee \neg L \vee \neg L$
But we can eliminate ("collect") the repeated $\neg L$ term (see book sec 4.1)

Backward

- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\neg P \vee \cancel{Q}$
2. $\neg L \vee \neg M \vee \cancel{P}$
3. $\neg B \vee \neg L \vee \cancel{M}$
4. $\neg A \vee \neg P \vee L$
5. $\neg A \vee \neg B \vee \cancel{L}$
6. A
7. B

8. $\neg Q$ (query)

9. $\neg P$ (1,8)

10. $\neg L \vee \neg M$ (2,9)

11. $\neg B \vee \neg L$ (3,10)

12. $\neg A \vee \neg B$ (5,11)

Note: we could instead resolve (4,10) to get

$$\neg A \vee \neg B \vee \neg P$$

But this would lead us to a dead end: after resolving this with 6 (A) and 7 (B), we would get

$$\neg P$$

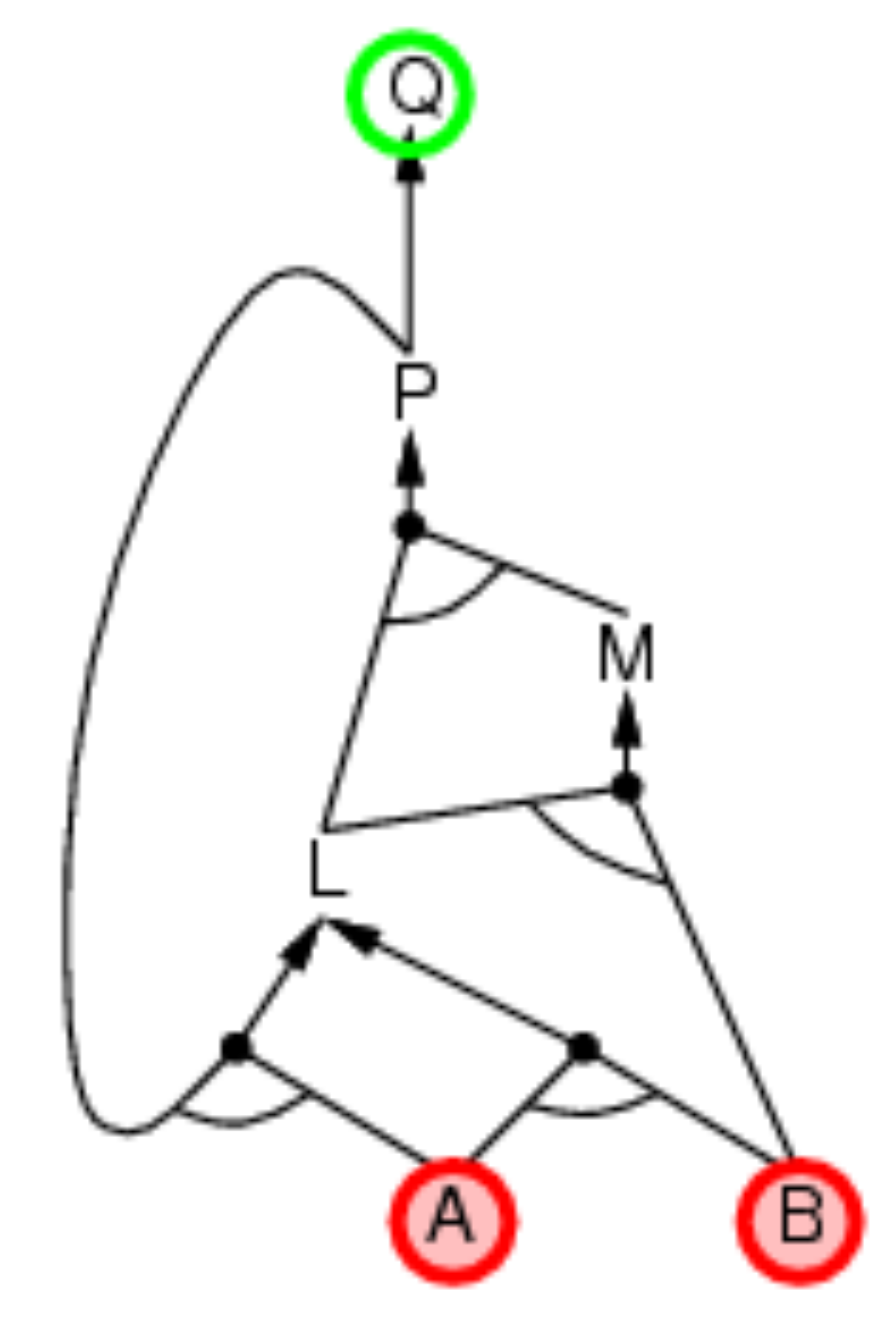
Which is the exact same goal we've already attempted to solve from clause 9. Ideally, our algorithm would detect this repeated goal and move to another available clause such as 5.

Backward

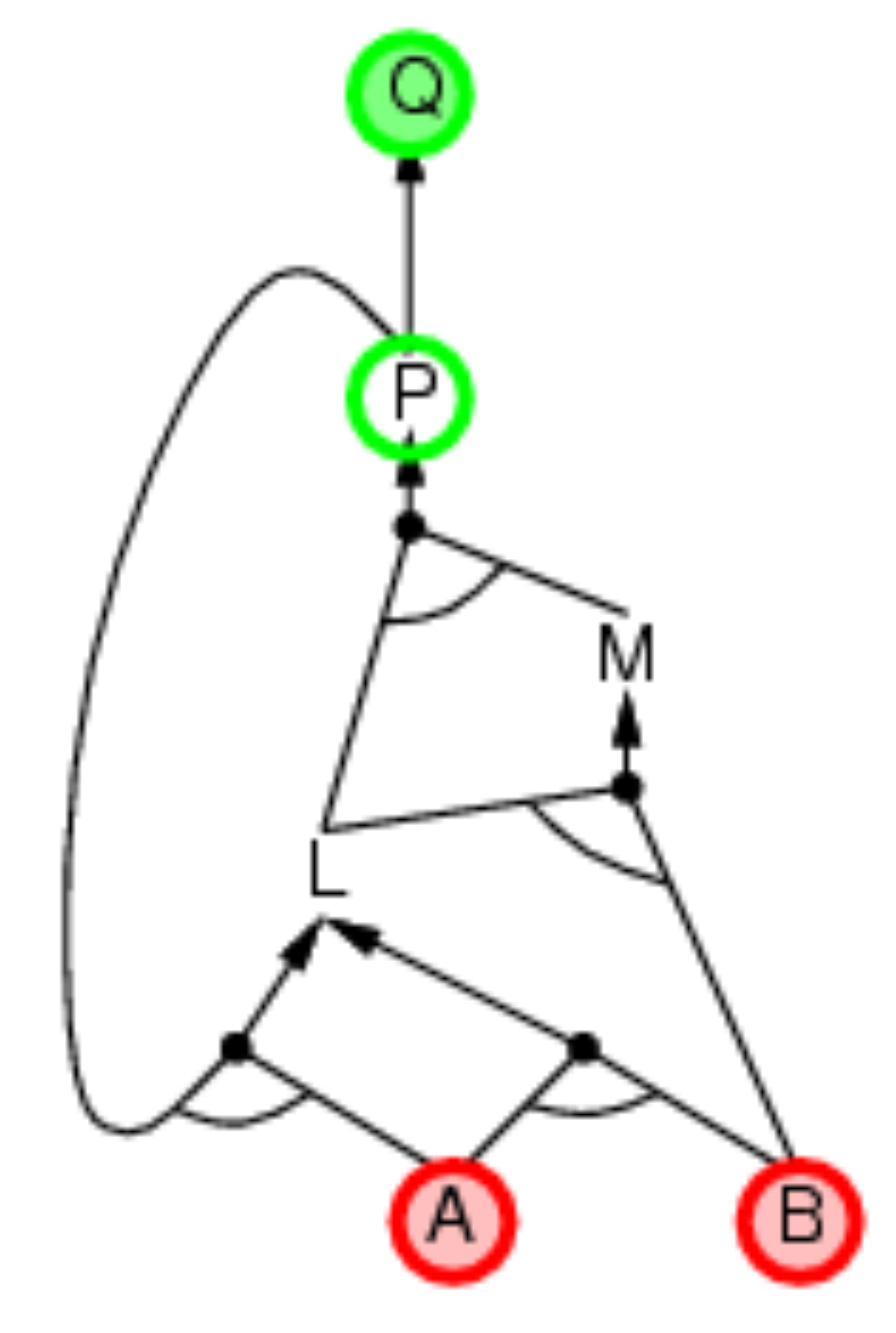
- Consider the following KB (AIMA Ch. 7.5.4), and try to prove $KB \vdash Q$:

1. $\neg P \lor \cancel{Q}$
2. $\neg L \lor \neg M \lor \cancel{P}$
3. $\neg B \lor \neg L \lor \cancel{M}$
4. $\neg A \lor \neg P \lor L$
5. $\neg A \lor \neg B \lor \cancel{L}$
6. A
7. B
8. $\neg Q$ (query)
9. $\neg P$ (1,8)
10. $\neg L \lor \neg M$ (2,9)
11. $\neg B \lor \neg L$ (3,10)
12. $\neg A \lor \neg B$ (5,11)
13. $\neg B$ (6,12)
14. [] (7,13)

Backward chaining example



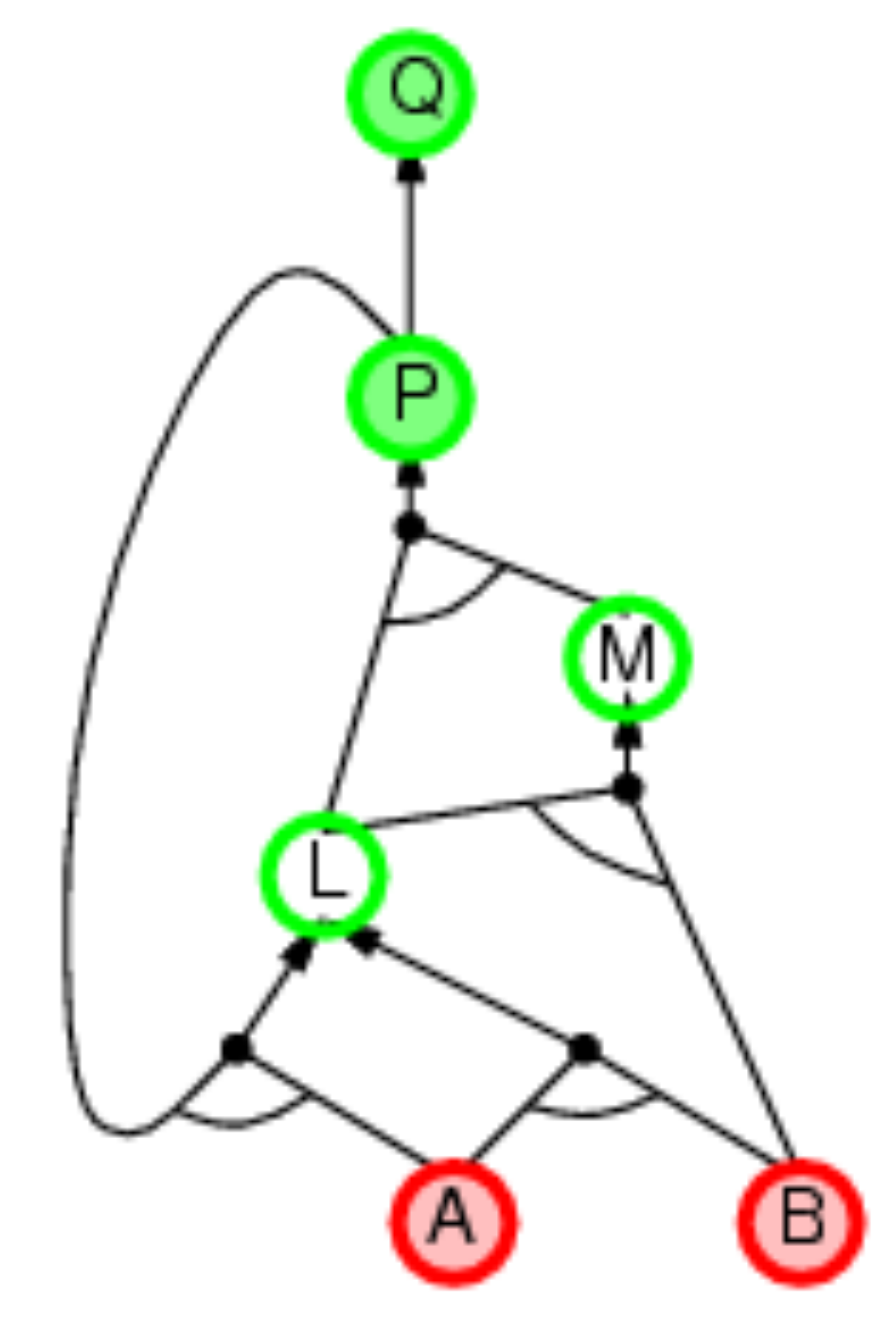
Backward chaining example



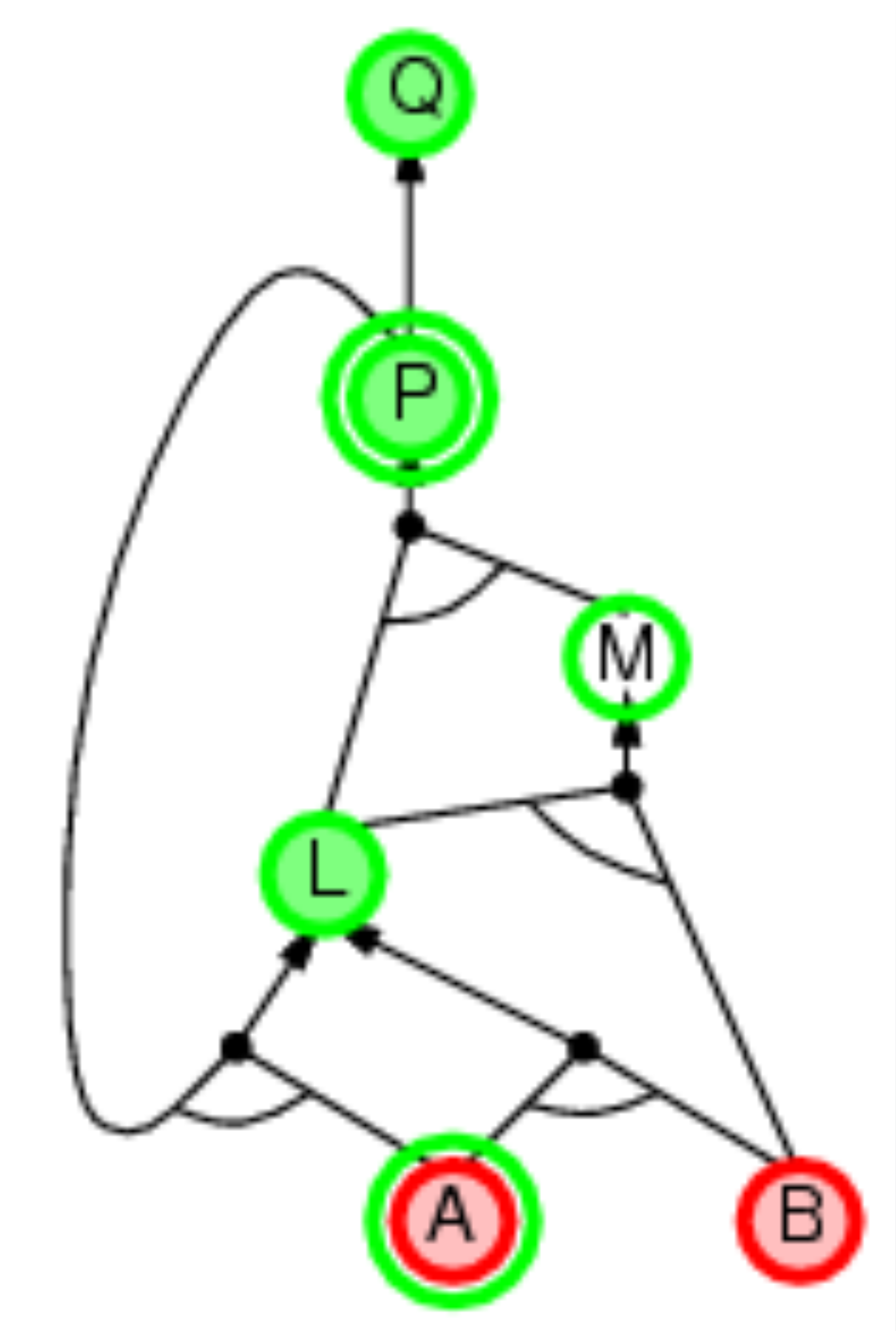
CAL POLY

http://aima.eecs.berkeley.edu/slides-ppt/

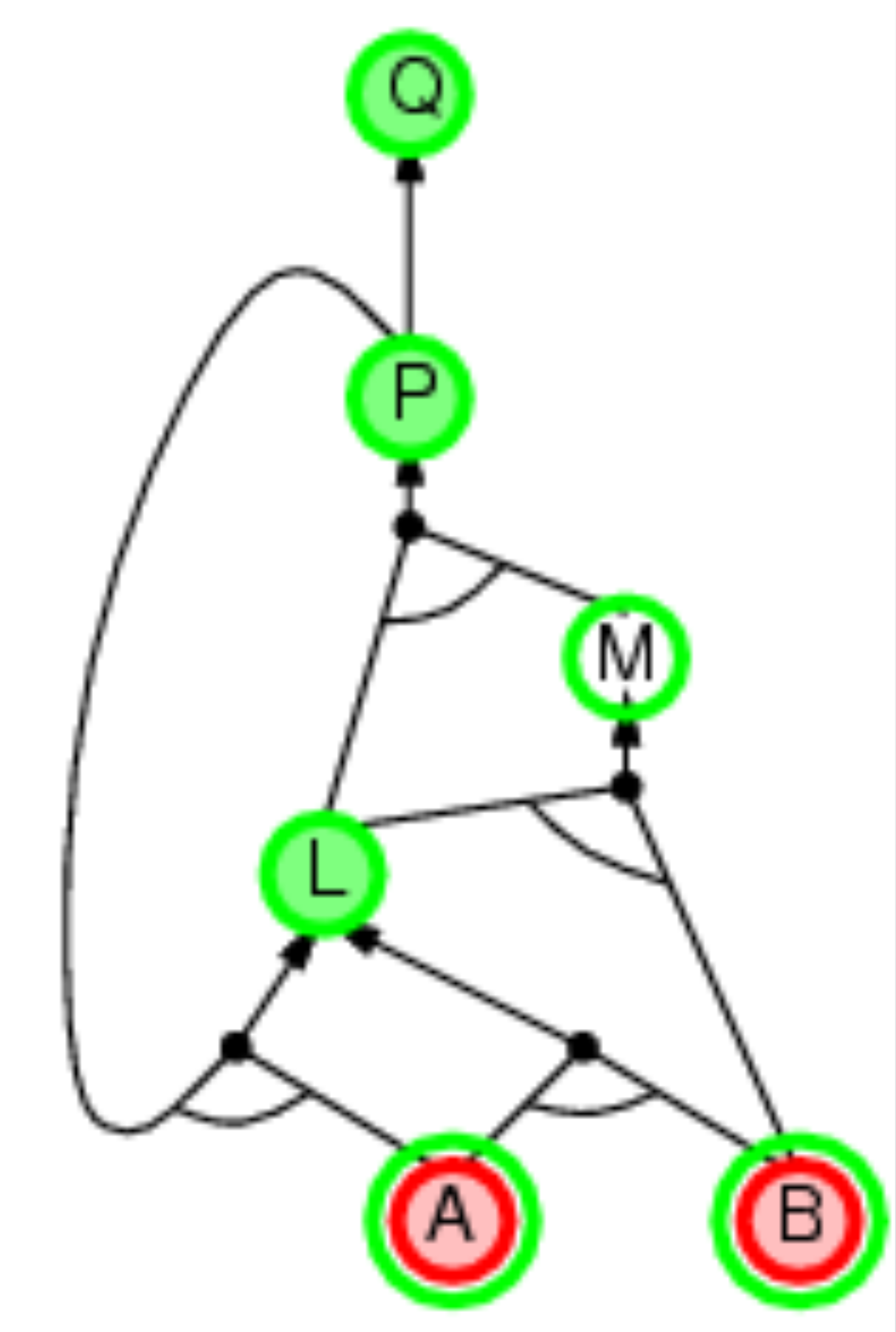
Backward chaining example



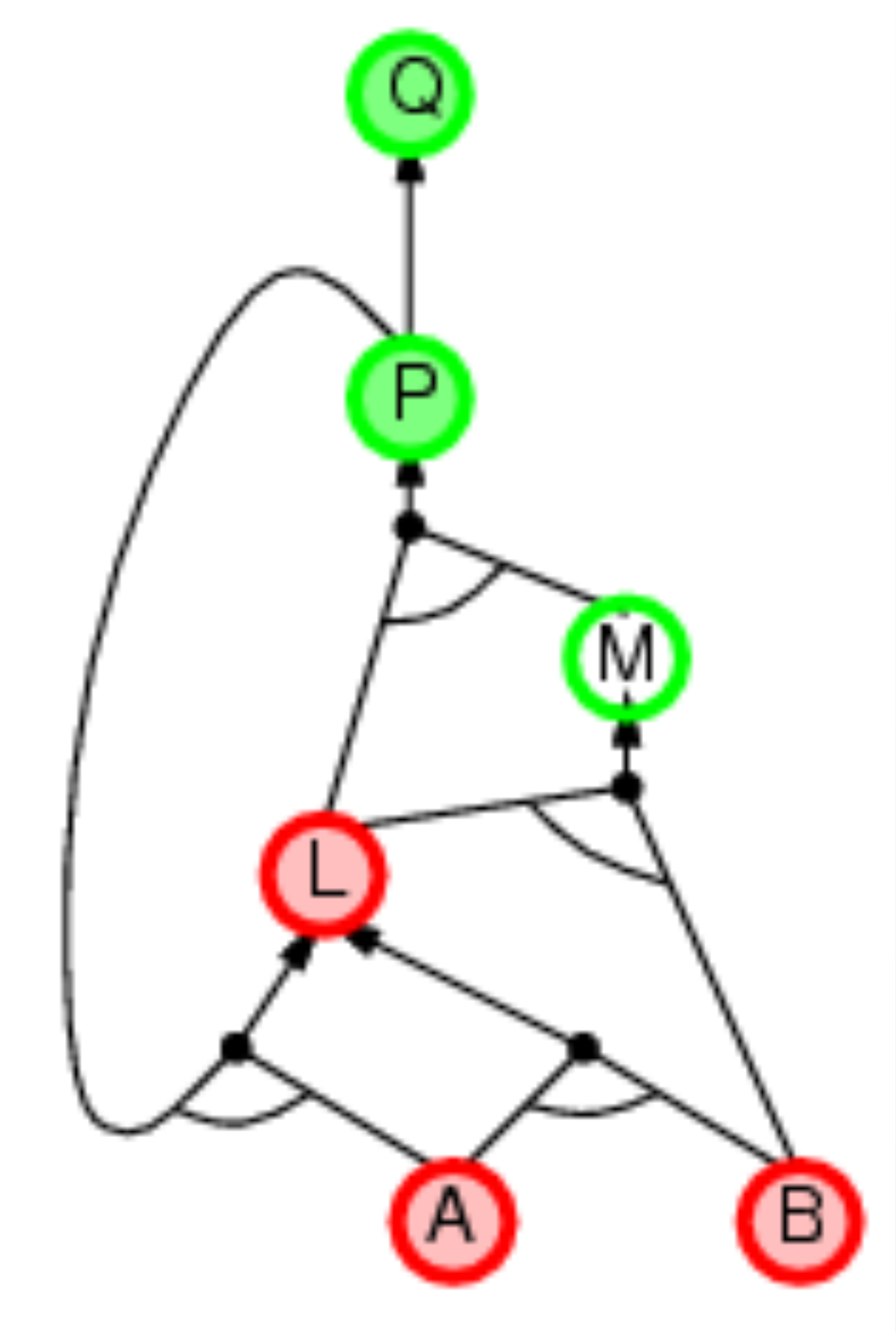
Backward chaining example



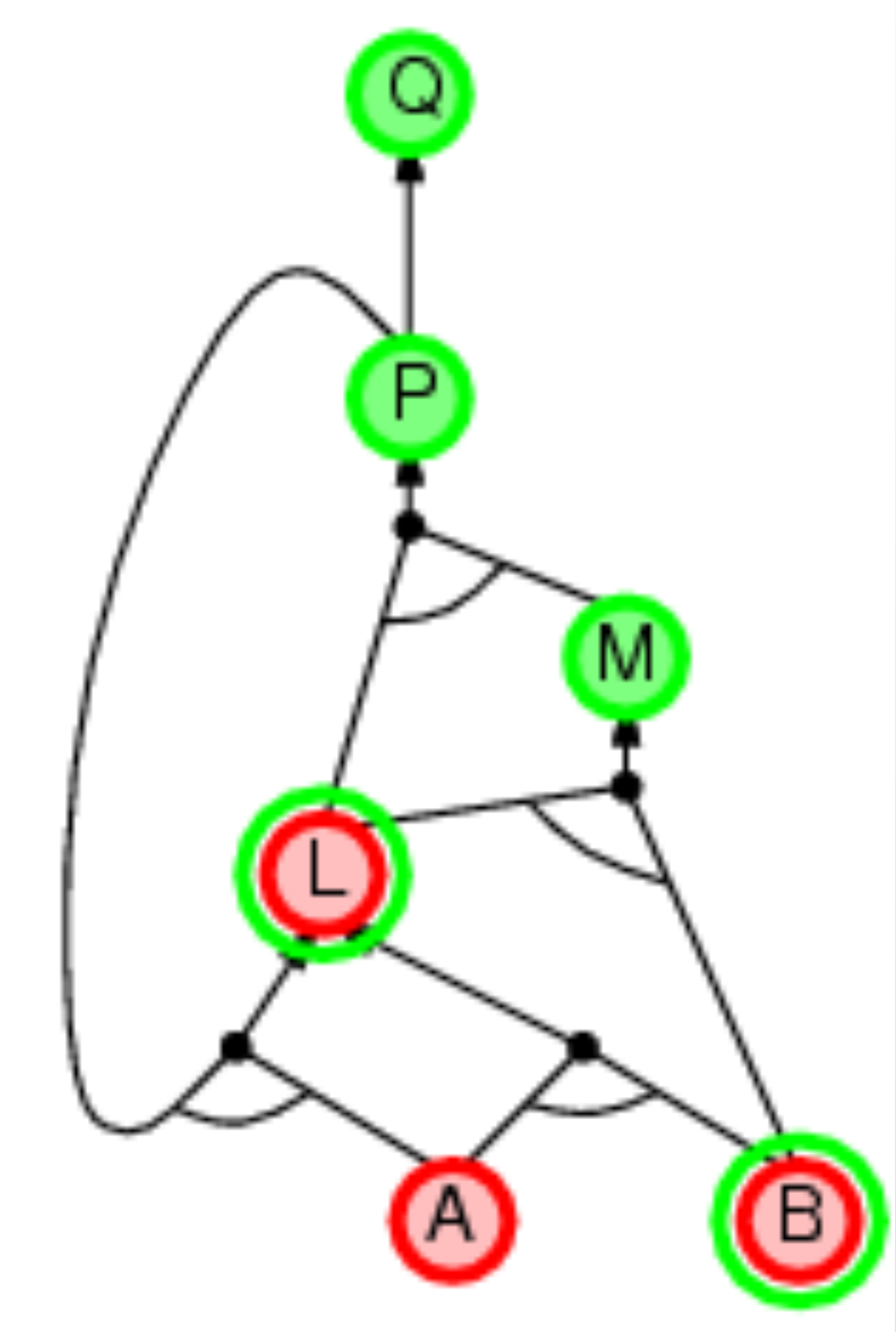
Backward chaining example



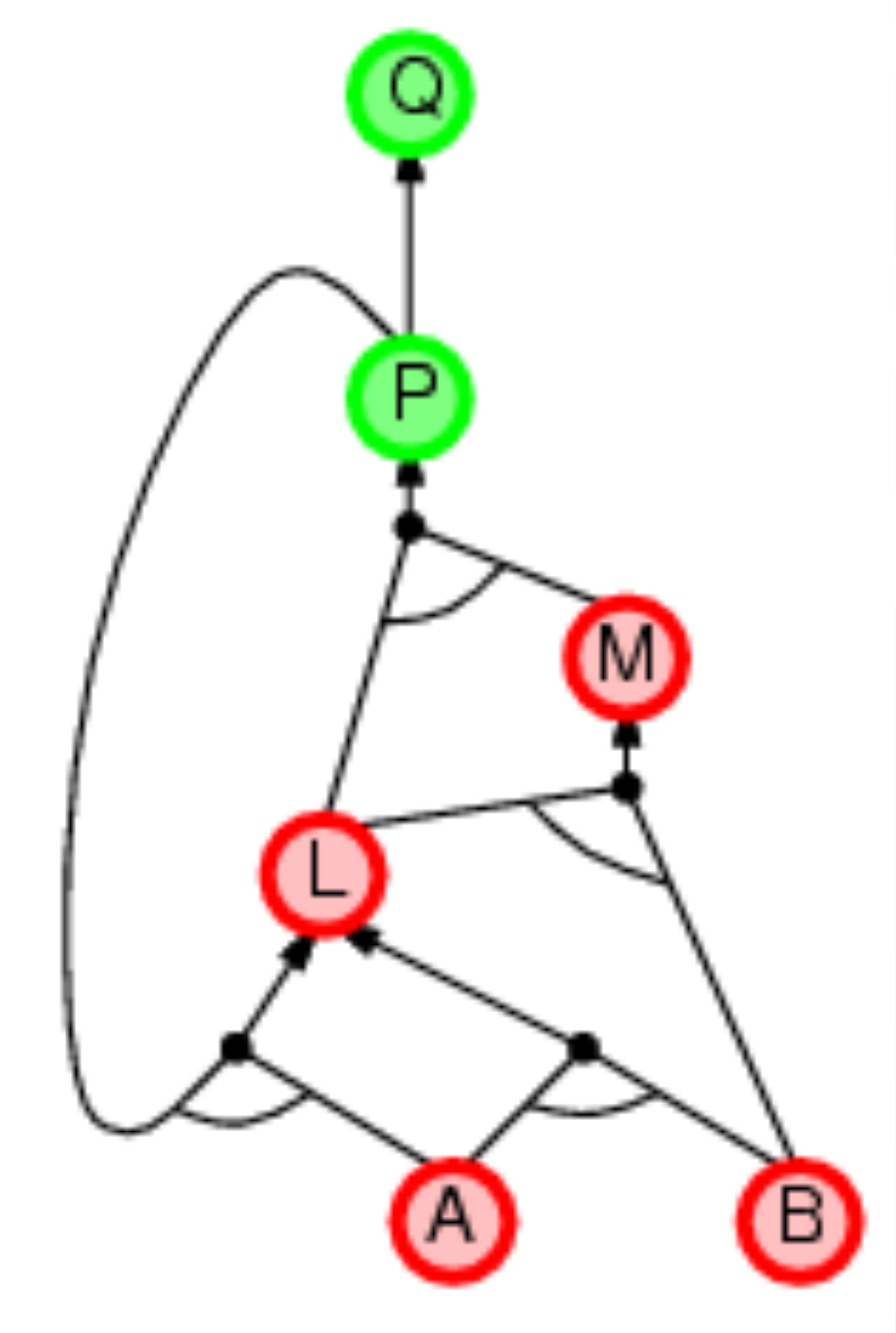
Backward chaining example



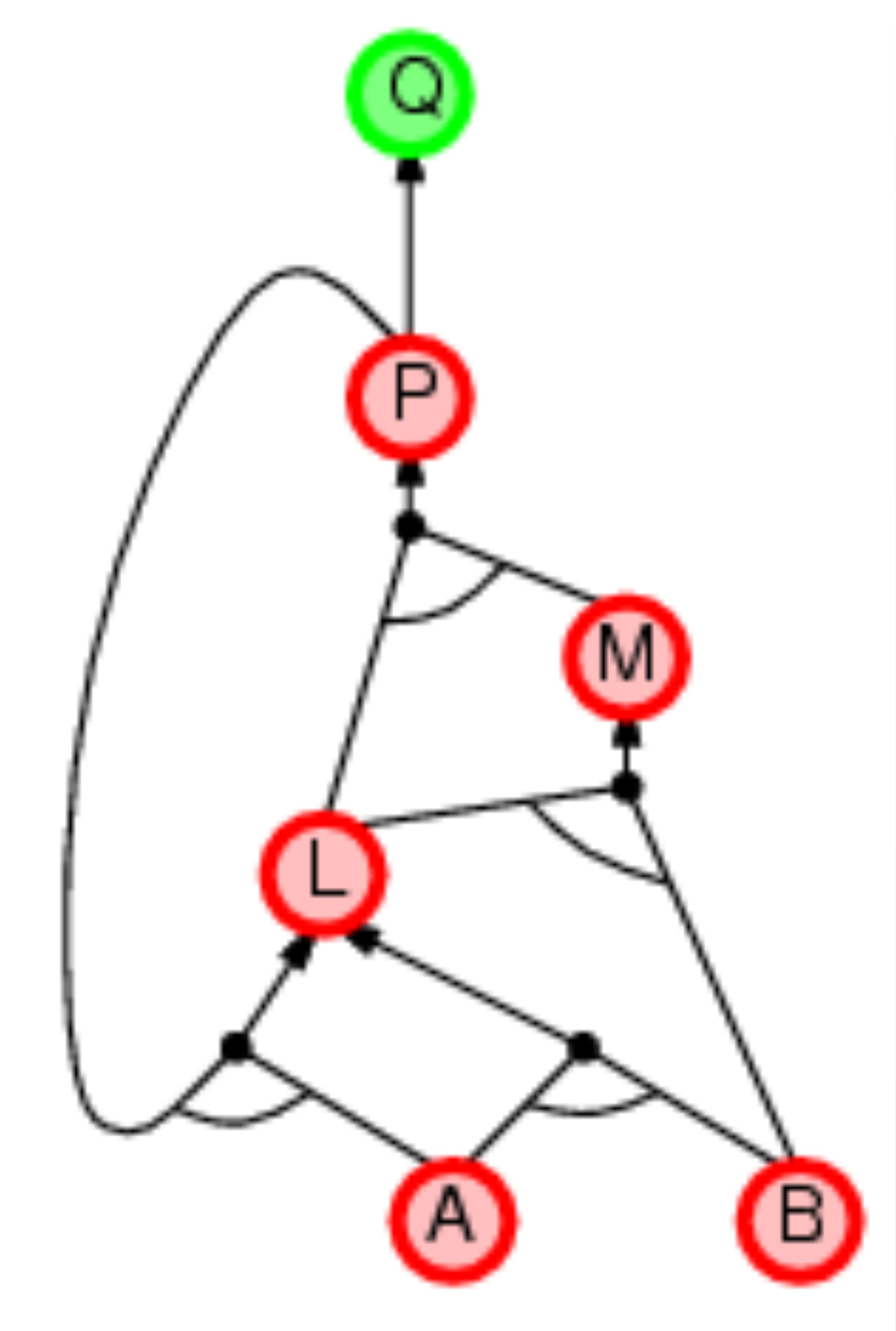
Backward chaining example



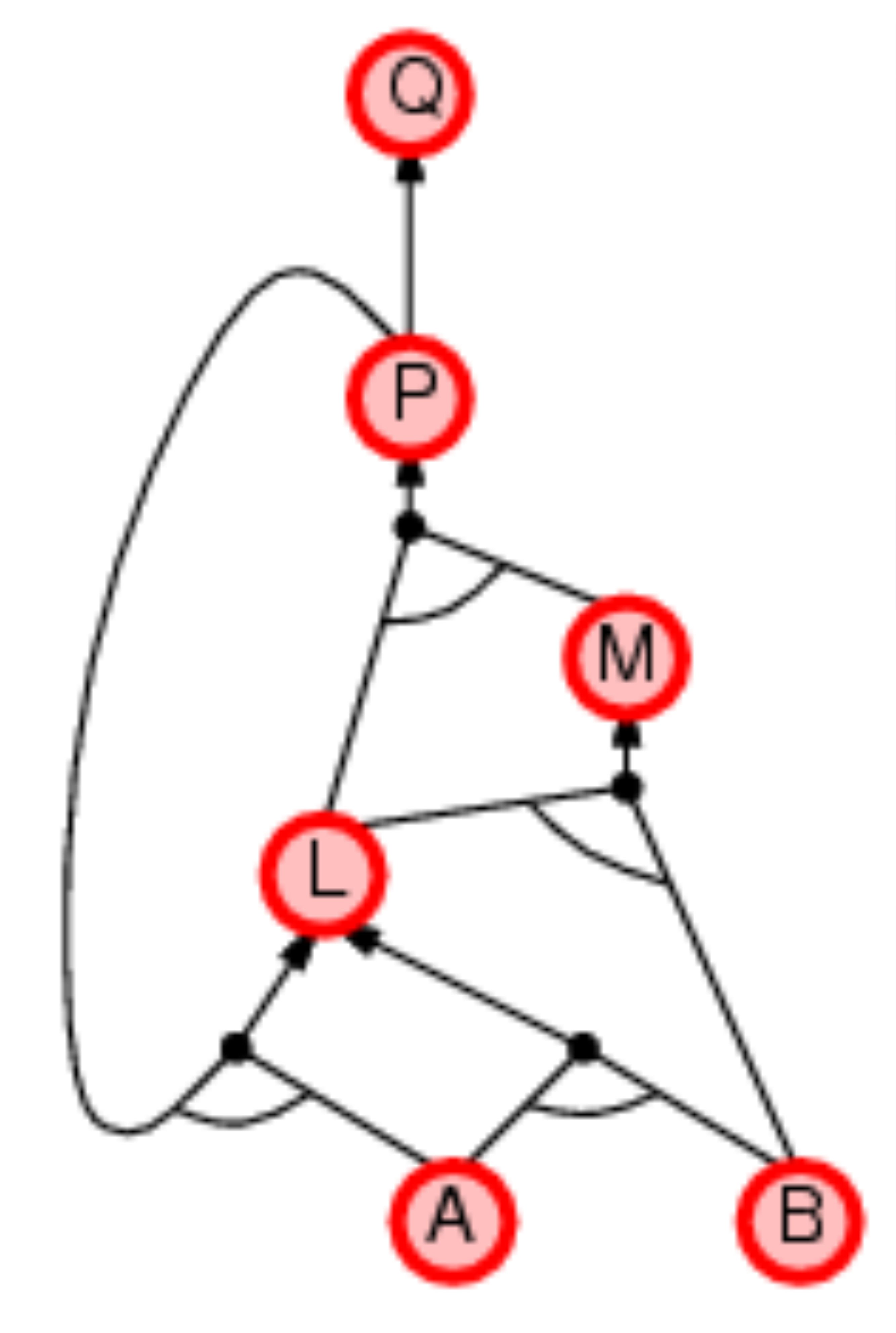
Backward chaining example



Backward chaining example



Backward chaining example



Forward chaining vs Backward chaining

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**
  - It starts by generating new data from existing data

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**
  - It starts by generating new data from existing data
  - Can be useful even without a specific goal in mind

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**
  - It starts by generating new data from existing data
  - Can be useful even without a specific goal in mind
  - But can also can spend a long time generating irrelevant facts

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**
  - It starts by generating new data from existing data
  - Can be useful even without a specific goal in mind
  - But can also can spend a long time generating irrelevant facts
    - Imagine modifying the previous KB with a long chain of rules starting with a new literal C and never reaching Q

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**
  - It starts by generating new data from existing data
  - Can be useful even without a specific goal in mind
  - But can also can spend a long time generating irrelevant facts
    - Imagine modifying the previous KB with a long chain of rules starting with a new literal C and never reaching Q
- Backward Chaining is **goal driven:**

Forward chaining vs Backward chaining

- Forward Chaining is **data driven:**
  - It starts by generating new data from existing data
  - Can be useful even without a specific goal in mind
  - But can also can spend a long time generating irrelevant facts
    - Imagine modifying the previous KB with a long chain of rules starting with a new literal C and never reaching Q
- Backward Chaining is **goal driven:**
  - We only need to consider clauses that contain one of the current goals as consequences

Pseudocode of naive Forward Chaining

```
1  def naive_FC(KB)
2  #Returns True if KB is unsatisfiable, False otherwise
3      repeat = True
4      while repeat:
5          repeat = False
6          for each clause c1 in KB:
7              for negative literal l1 in c1:
8                  for each clause c2 in KB:
9                      for positive literal l2 in c2:
10                         c1.match = True
11             if all c1.match are true:
12                 new_fact = positive literal in c1
13                 if not_in(new_fact, KB):
14                     KB.append(new_fact)
15                     # If at least one fact was added, we repeat
16                     # the while loop on line 4
17                     repeat = True
18
```

Pseudocode of naive Forward Chaining

```
1  def naive_FC(KB)
2  #Returns True if KB is unsatisfiable, False otherwise
3      repeat = True
4      while repeat:
5          repeat = False
6          for each clause c1 in KB:
7              for negative literal l1 in c1:
8                  for each clause c2 in KB:
9                      for positive literal l2 in c2:
10                         c1.match = True
11             if all c1.match are true:
12                 new_fact = positive literal in c1
13                 if not_in(new_fact, KB):
14                     KB.append(new_fact)
15                     # If at least one fact was added, we repeat
16                     # the while loop on line 4
17                     repeat = True
18
```

If KB has **C** clauses and **L** literals in total (including multiple occurrences of a literal)

Pseudocode of naive Forward Chaining

```
1  def naive_FC(KB)
2  #Returns True if KB is unsatisfiable, False otherwise
3      repeat = True
4      while repeat:
5          repeat = False
6          for each clause c1 in KB:
7              for negative literal l1 in c1:
8                  for each clause c2 in KB:
9                      for positive literal l2 in c2:
10                         c1.match = True
11             if all c1.match are true:
12                 new fact = positive literal in c1
13                 if not_in(new_fact, KB):
14                     KB.append(new_fact)
15                     # If at least one fact was added, we repeat
16                     # the while loop on line 4
17                     repeat = True
18 
```

If KB has **C** clauses and **L** literals in total
(including multiple occurrences of a literal)

This repeats some # of times (call it O(k)

This goes over every clause and literal
So it is O (C+L)

This is also O (C+L)
But can easily be made O(C)
with an index to the only positive literal

This could also be O (C+L) if implemented inefficiently

Pseudocode of naive Forward Chaining

```
1  def naive_FC(KB)
2  #Returns True if KB is unsatisfiable, False otherwise
3      repeat = True
4      while repeat:
5          repeat = False
6          for each clause c1 in KB:
7              for negative literal l1 in c1:
8                  for each clause c2 in KB:
9                      for    If c2 is a fact and the same as l1
10                         flag l1 as resolved
11             if all  negative literals in c1 are resolved
12                 new fact = positive literal in c1
13                 if not_in(new_fact, KB):
14                     KB.append(new_fact)
15                     # If at least one fact was added, we repeat
16                     # the while loop on line 4
17                     repeat = True
18
```

If KB has **C** clauses and **L** literals in total (including multiple occurrences of a literal)

This repeats some # of times (call it O(k)

This goes over every clause and literal
So it is O (C+L)

This is also O (C+L)
But can easily be made O(C)
with an index to the only positive literal

This could also be O (C+L) if implemented inefficiently

Even if k is small, the whole algorithm could be O(C+L)^2

Pseudocode: Forward Chaining

- **Forward chaining is sound and complete for Horn KB**
- **This version can be linear in the size of the KB**

```
function PL-FC-ENTAILS?(KB, q) returns true or false
   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 to be true

   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
```

http://aima.eecs.berkeley.edu/slides-ppt/

Pseudocode: Forward Chaining

- **Forward chaining is sound and complete for Horn KB**
- **This version can be linear in the size of the KB**

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  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 to be true

  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
```

This in total does some constant amount of work For each clause and *distinct* symbol in the KB, Which is at worst O(C+L)

With a map from each symbol to its occurrences, every premise can be handled in constant time.This effectively ticks every literal in the body of a rule only once, and adds its conclusion only once, so it does O(C+L) work

The amount of work of the inner loop does not depend on the outer loop so the whole algorithm is O(C+L)

1.- A v -B v -C v D

2.A

3.B

4.C

5.(query) -D

Forward Chaining

1.- A v -B v -C v D

2.A

3.B

4.C

5.(query) -D

6. -B v -C v D (1,2)

7.-C v D (3,6)

8. D (4,7)

9.[] (5,8)

Always resolving a free **positive** clause (“fact”) with the **body** of a rule

Backward Chaining

1.- A v -B v -C v D

2.A

3.B

4.C

5.(query) -D

6.- A v -B v -C (1,5)

7.-B v -C (2,6)

8.-C (3,7)

9.[] (4,8)

Always resolving a free **negative** clause (“goal”) with the **head** of a rule

1.- A v -B v -C v D

2.A

3.B

4.C

5.(query) -D

Backward Chaining

1.- A v -B v -C v D

2.A

3.B

4.C

5.(query) -D

6.- A v -B v -C (1,5)

7.-B v -C (2,6)

8.-C (3,7)

9.[] (4,8)

Forward Chaining

1.- A v -B v -C v D

2.A

3.B

4.C

5.(query) -D

6. -B v -C v D (1,2)

7.-C v D (3,6)

8. D (4,7)

9.[] (5,8)

Analysis of performance

Analysis of performance

- Naive implementations of either algorithm can be quadratic or even exponential in time

Analysis of performance

- Naive implementations of either algorithm can be quadratic or even exponential in time
- With careful use of data structures, **forward chaining** can be made to run in linear time with respect to the size of the KB (total number of literals in all formulas)

Analysis of performance

- Naive implementations of either algorithm can be quadratic or even exponential in time
- With careful use of data structures, **forward chaining** can be made to run in linear time with respect to the size of the KB (total number of literals in all formulas)
- **Backward chaining** is stated by the Brachman & Levesque book to require an exponential number of steps, but AIMA says it can be made linear

Analysis of performance

- Naive implementations of either algorithm can be quadratic or even exponential in time
- With careful use of data structures, **forward chaining** can be made to run in linear time with respect to the size of the KB (total number of literals in all formulas)
- **Backward chaining** is stated by the Brachman & Levesque book to require an exponential number of steps, but AIMA says it can be made linear
  - For propositional Horn clauses, I think AIMA is correct, but for us, the exact time complexity doesn't matter

Analysis of performance

- Naive implementations of either algorithm can be quadratic or even exponential in time
- With careful use of data structures, **forward chaining** can be made to run in linear time with respect to the size of the KB (total number of literals in all formulas)
- **Backward chaining** is stated by the Brachman & Levesque book to require an exponential number of steps, but AIMA says it can be made linear
  - For propositional Horn clauses, I think AIMA is correct, but for us, the exact time complexity doesn't matter
- Prolog uses backward chaining as its backbone!

What about FOL

What about FOL

- SLD resolution can also work with statements using quantifiers, predicates, functions and equality

What about FOL

- SLD resolution can also work with statements using quantifiers, predicates, functions and equality
- We resolve universals in the same way as before: through unification

What about FOL

- SLD resolution can also work with statements using quantifiers, predicates, functions and equality
- We resolve universals in the same way as before: through unification
  - Example: from

What about FOL

- SLD resolution can also work with statements using quantifiers, predicates, functions and equality
- We resolve universals in the same way as before: through unification
  - Example: from
    - $\forall x \,.\, dog(x) \rightarrow mammal(x)$

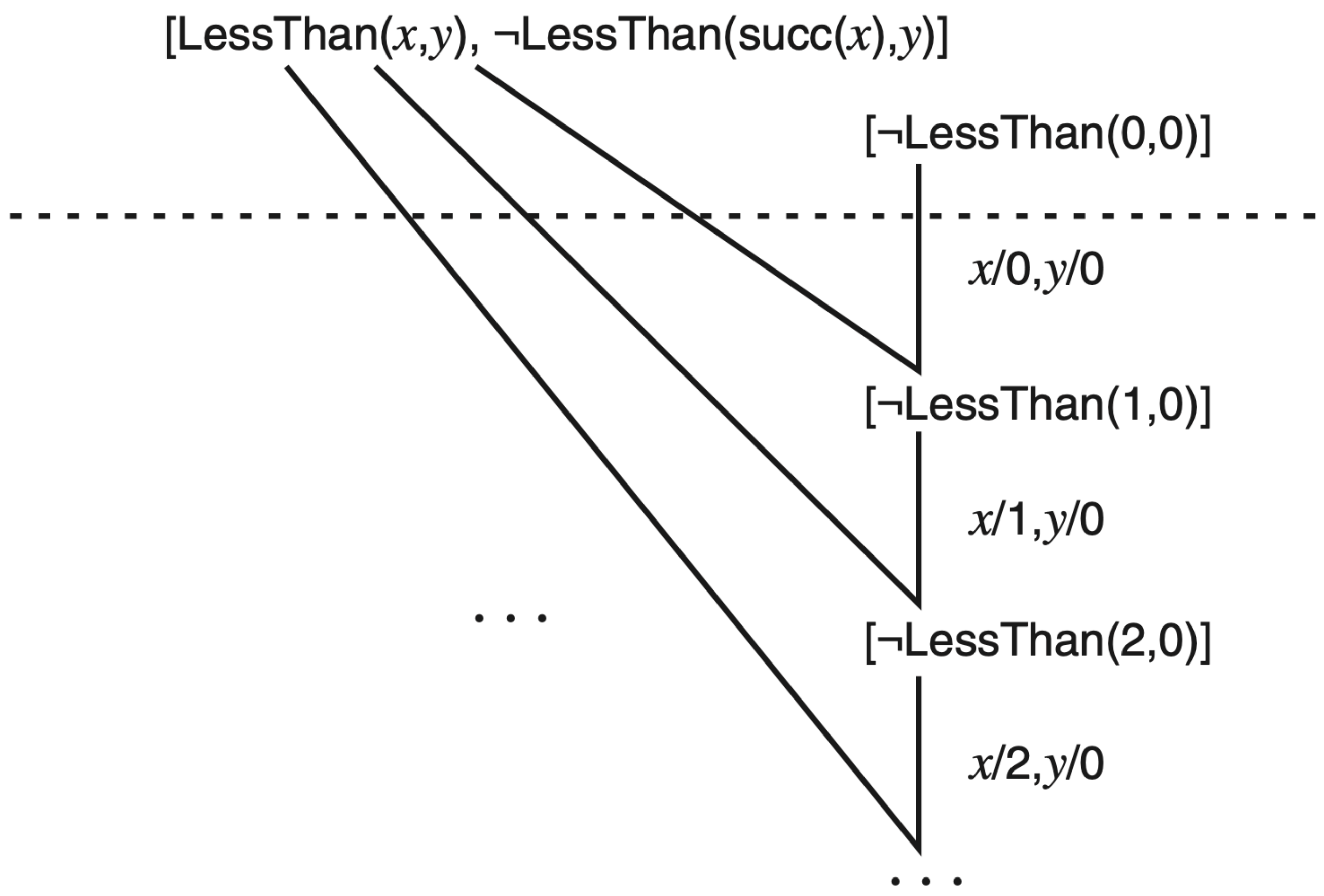
What about FOL

- SLD resolution can also work with statements using quantifiers, predicates, functions and equality
- We resolve universals in the same way as before: through unification
  - Example: from
    - $\forall x \,.\, dog(x) \to mammal(x)$
    - $dog(toto)$

What about FOL

- SLD resolution can also work with statements using quantifiers, predicates, functions and equality
- We resolve universals in the same way as before: through unification
  - Example: from
    - $\forall x \,.\, dog(x) \rightarrow mammal(x)$
    - $dog(toto)$
  - We can derive $mammal(toto)$

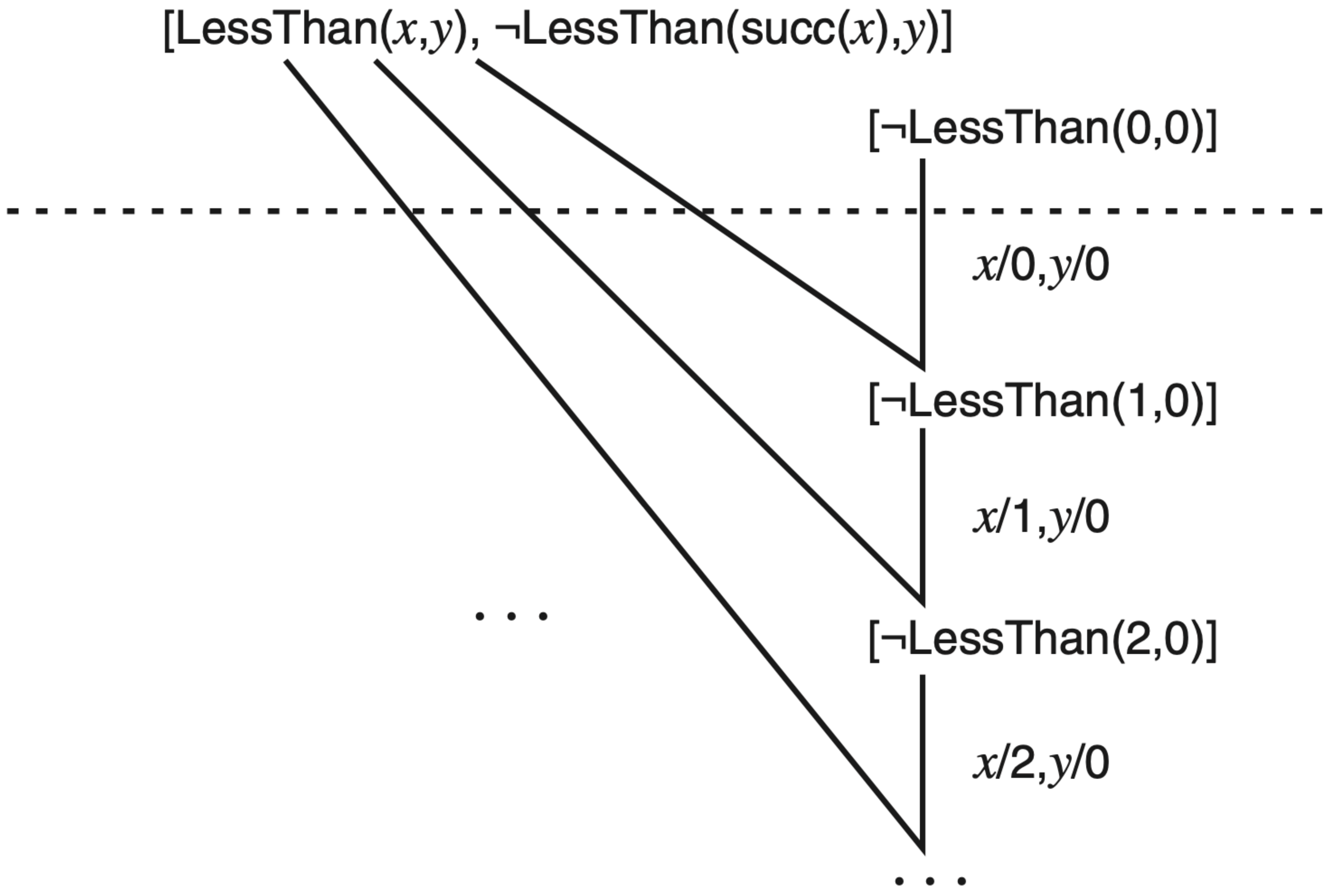
What about FOL?



Brachman & Levesque, Fig 4.11

What about FOL?

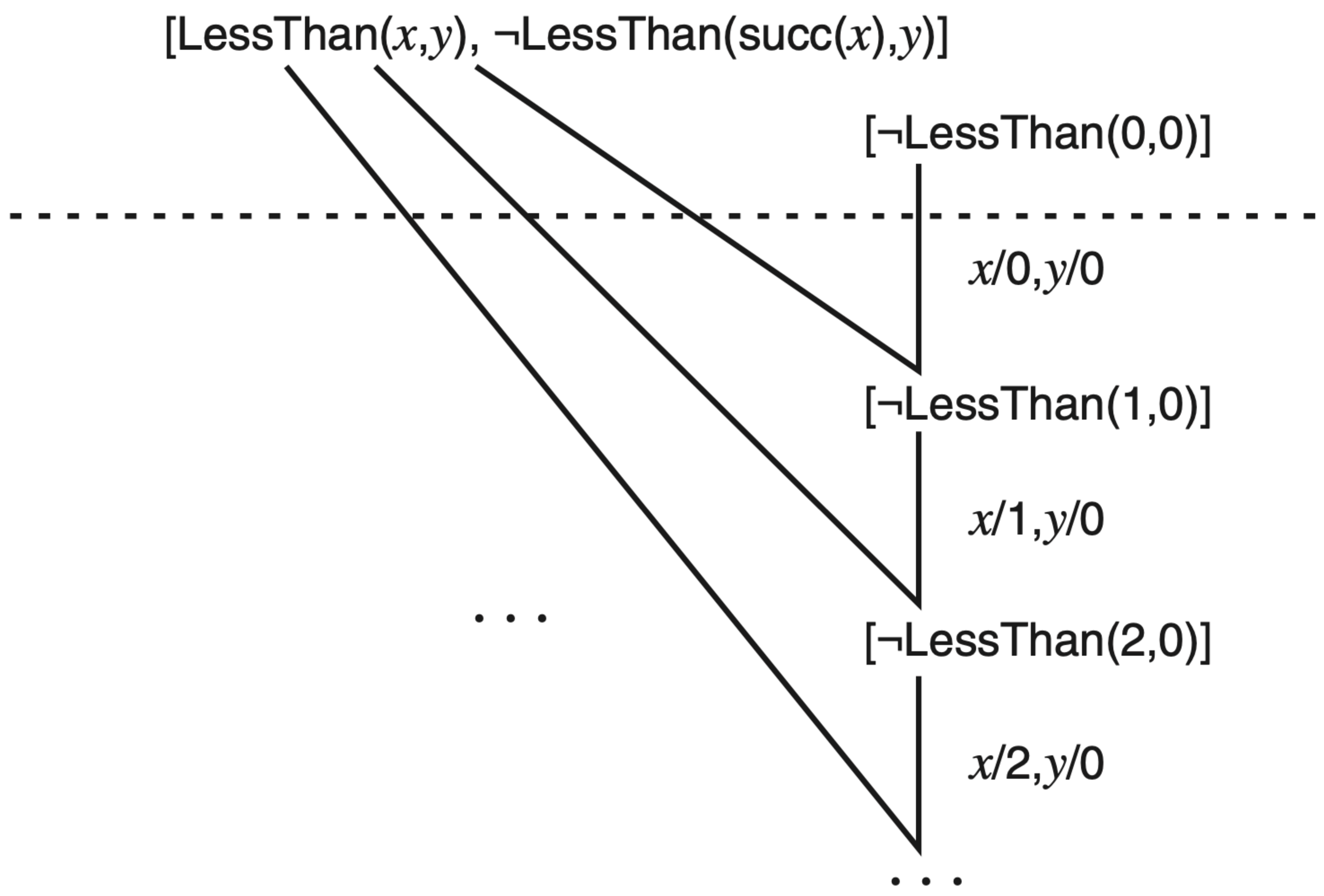
- However, some statements involving quantifiers may still lead to infinite loops



Brachman & Levesque, Fig 4.11

What about FOL?

- However, some statements involving quantifiers may still lead to infinite loops
- The problem of checking if a set of FoL Horn clauses entails an atom is *undecidable*



Brachman & Levesque, Fig 4.11

Then what?

Then what?

- Checking entailment using Horn clauses is usually much faster (linear in the propositional case) but can still fail

Then what?

- Checking entailment using Horn clauses is usually much faster (linear in the propositional case) but can still fail
- There is no silver bullet to cover all cases: the problem is in general *undecidable*

Then what?

- Checking entailment using Horn clauses is usually much faster (linear in the propositional case) but can still fail
- There is no silver bullet to cover all cases: the problem is in general *undecidable*
- But we can try to give as much control to the user as possible in deciding how deduction takes place

Then what?

- Checking entailment using Horn clauses is usually much faster (linear in the propositional case) but can still fail
- There is no silver bullet to cover all cases: the problem is in general *undecidable*
- But we can try to give as much control to the user as possible in deciding how deduction takes place
  - This is the theme of chapter 6 of the Knowledge Representation and Reasoning book

Lab

Lab

- We will implement forward chaining in Python, both naively and a more efficient version