Propositional Model Checking

Outline

I. Forward and Backward Chaining

II. Effective Propositional Model Checking

III. Agents Based on Propositional Logic

* Figures are from the textbook site unless a source is specifically cited.

I. Forward Chaining

Question $KB \models q$?

single proposition symbol

- Begins from positive literals (facts).
- If all the premises of an implications are known, then add its conclusion to KB (as a new fact).
- Continues until q is added or no further inferences can be made.

I. Forward Chaining

Question *KB* $\models q$?

single proposition symbol

- Begins from positive literals (facts).
- If all the premises of an implications are known, then add its conclusion to KB (as a new fact).
- Continues until q is added or no further inferences can be made.

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

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

Example of Forward Chaining

KB:

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

Example of Forward Chaining

KB:

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

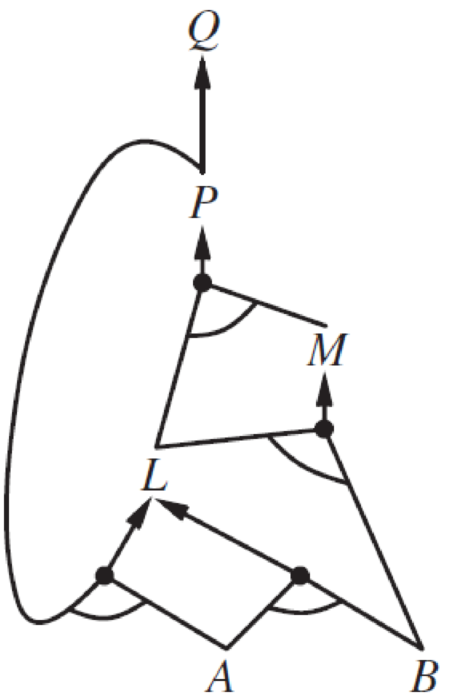
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

AND-OR graph representation



Example of Forward Chaining

KB:

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

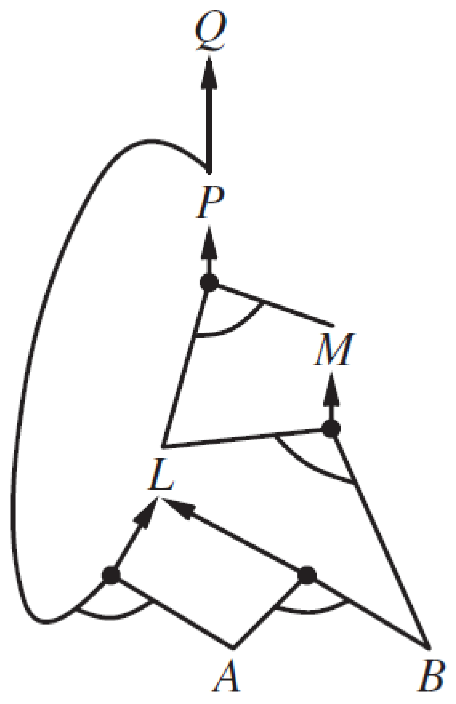
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

AND-OR graph representation



Q. $KB \models Q$?

Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

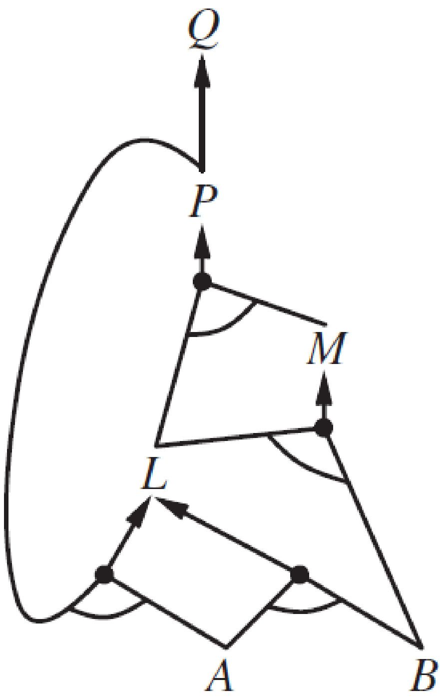
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

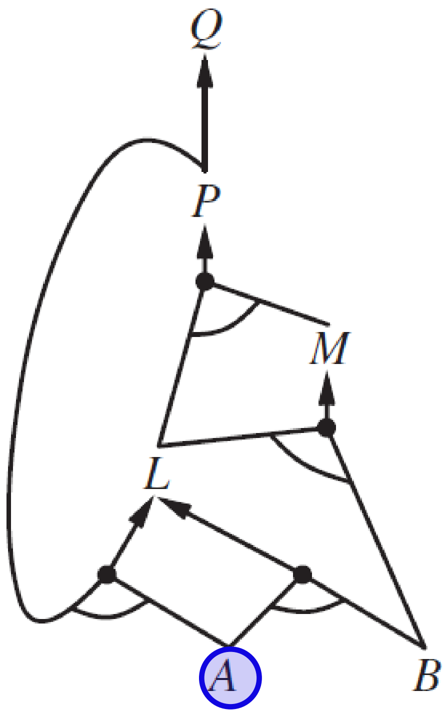
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

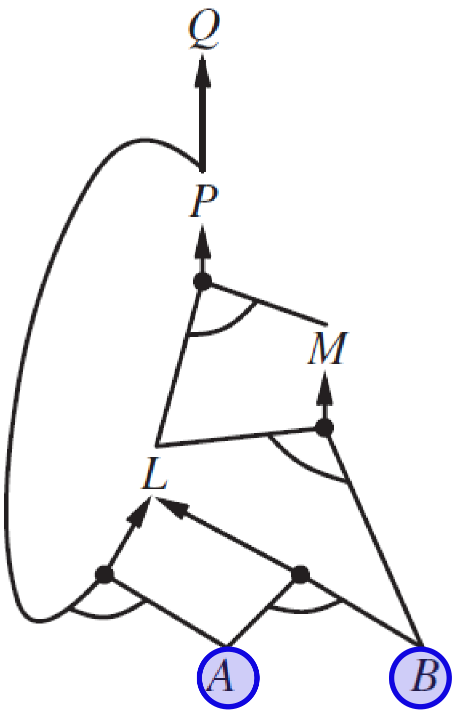
A

B



Execution

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



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

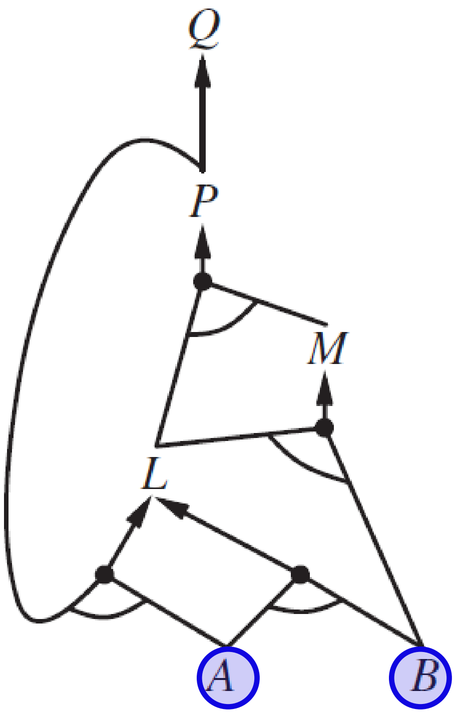
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

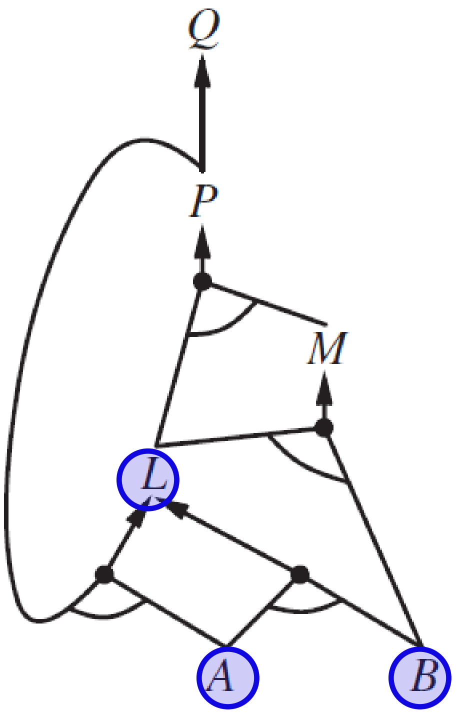
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

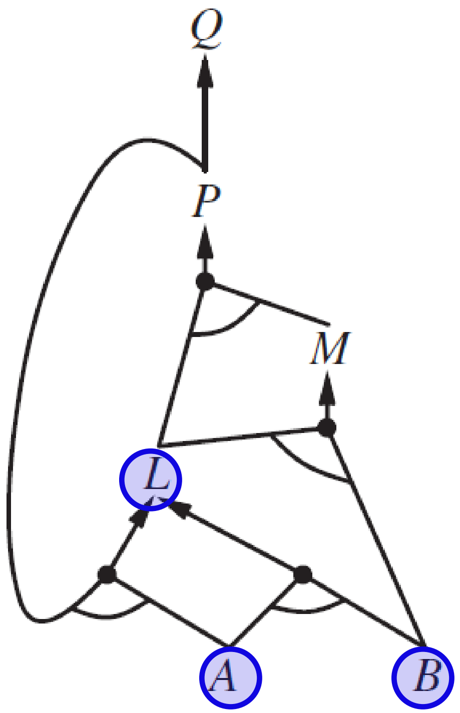
A

B



Execution

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



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

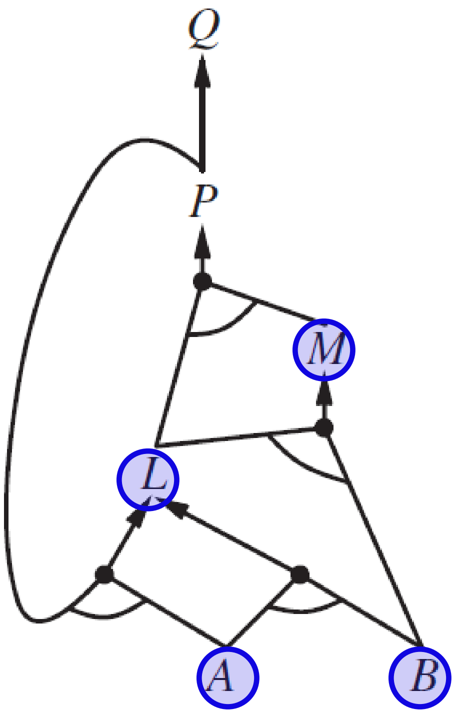
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

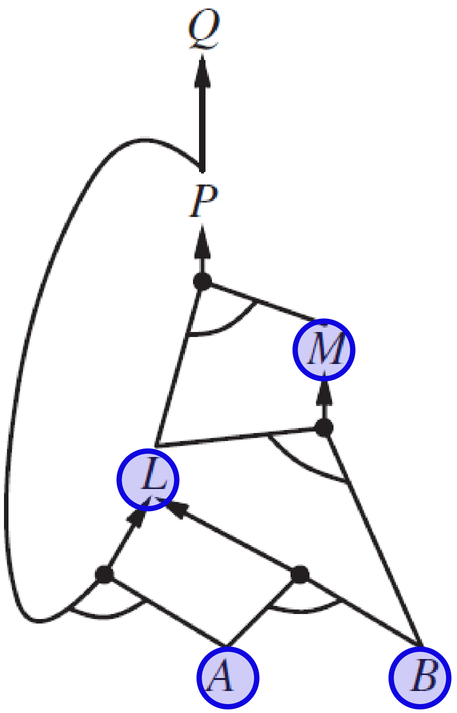
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

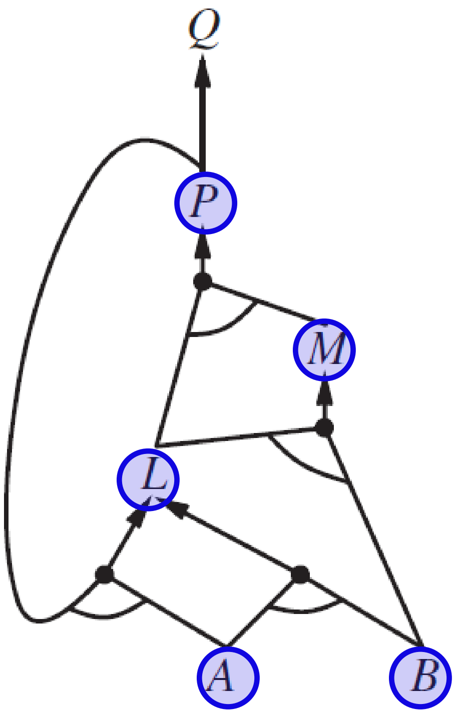
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

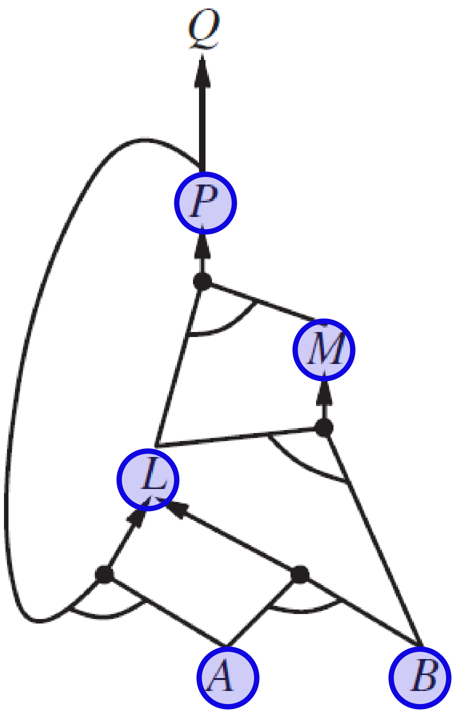
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

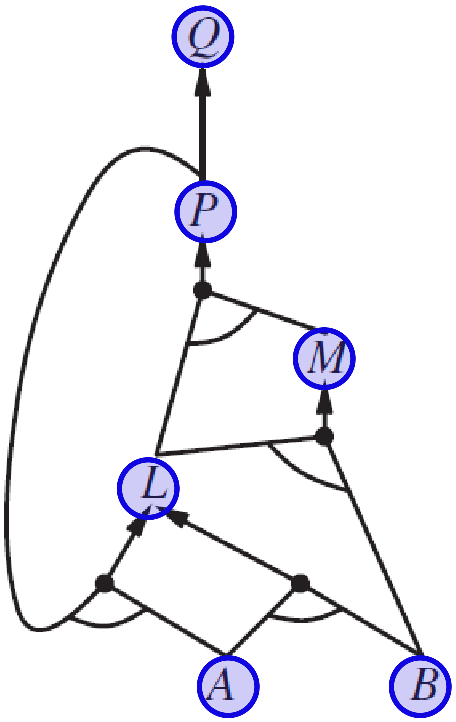
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

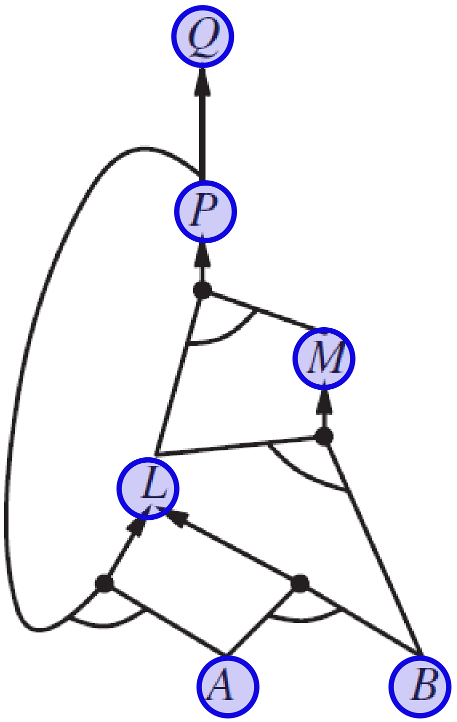
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Soundness of forward chaining: every inference is an application of Modus Ponens.

Execution

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

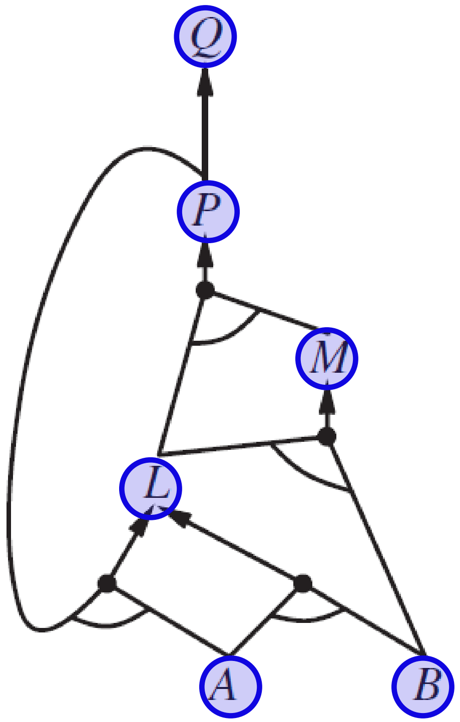
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Soundness of forward chaining: every inference is an application of Modus Ponens.

Completeness: every entailed atomic sentences will be derived.

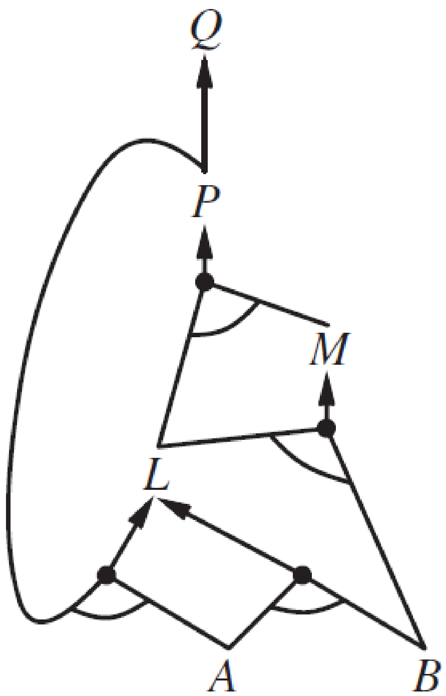
Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

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

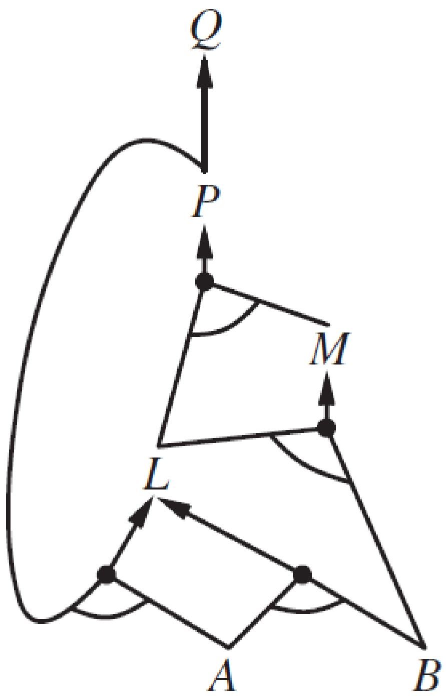


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

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

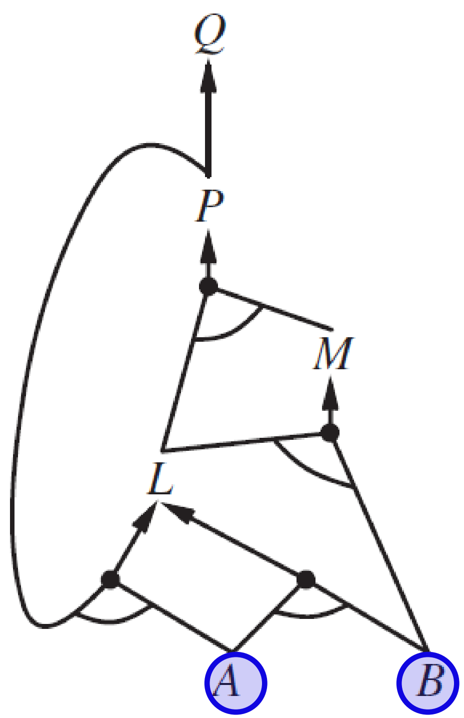


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

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

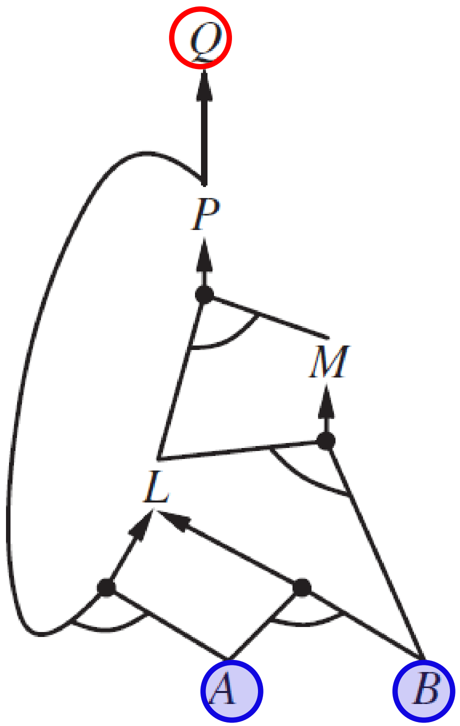


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

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



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

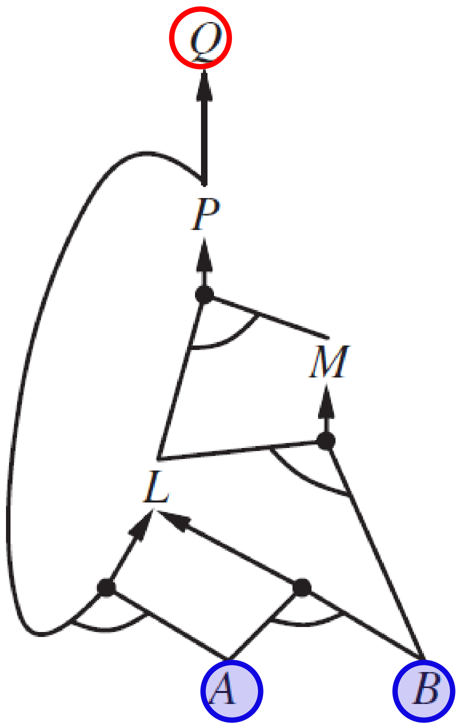
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

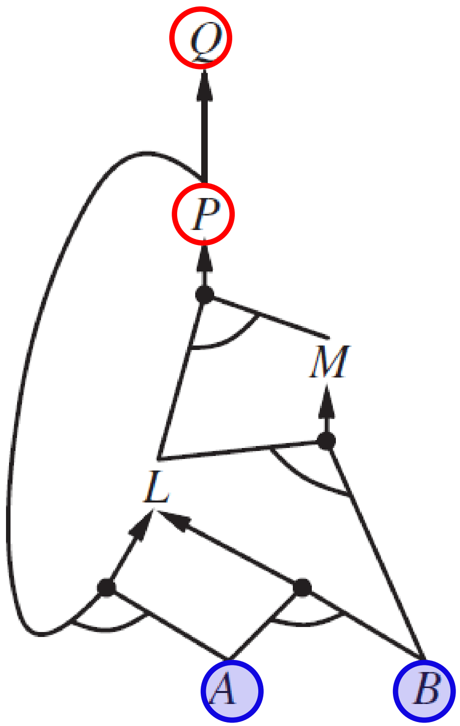


Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

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



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

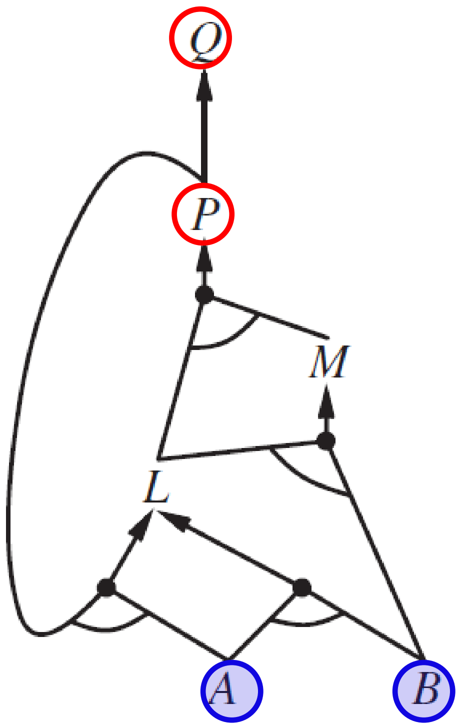
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

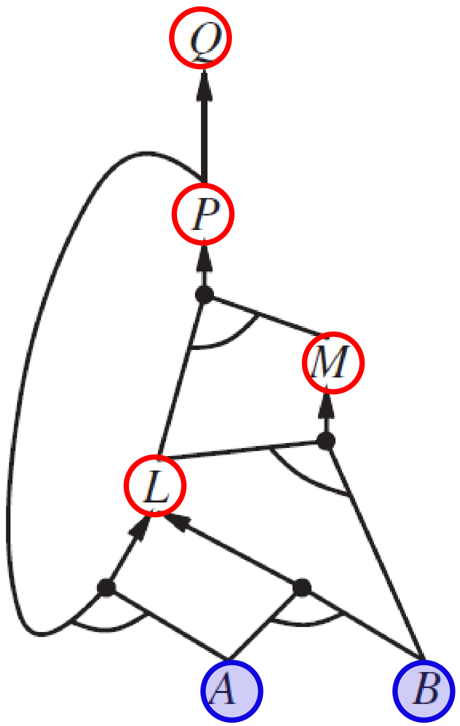
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

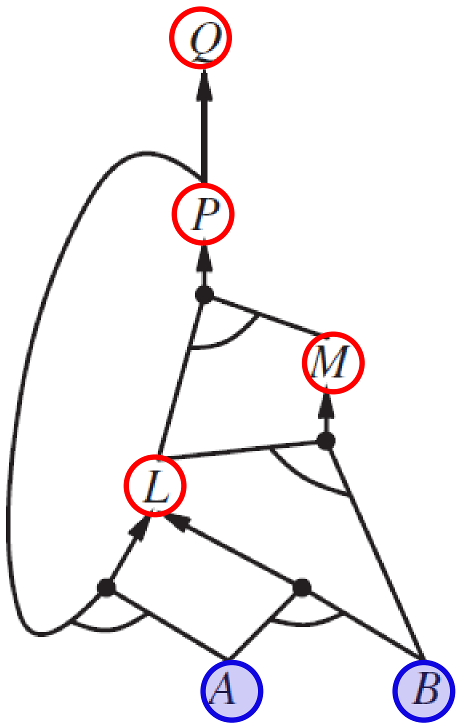
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \land M \Rightarrow P$

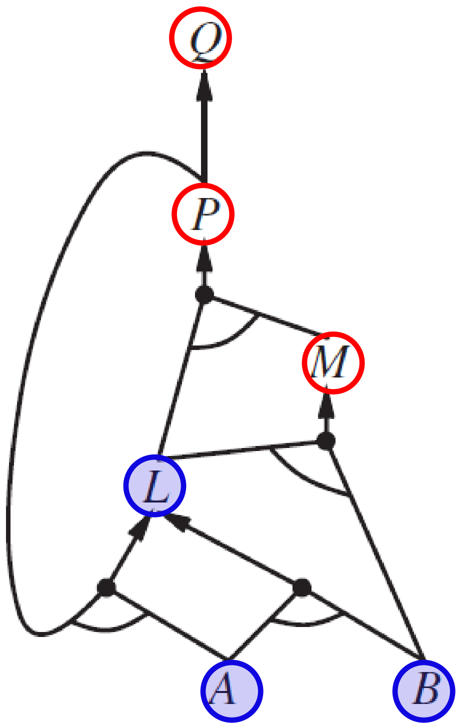
$B \land L \Rightarrow M$

$A \land P \Rightarrow L$

$A \land B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \land M \Rightarrow P$

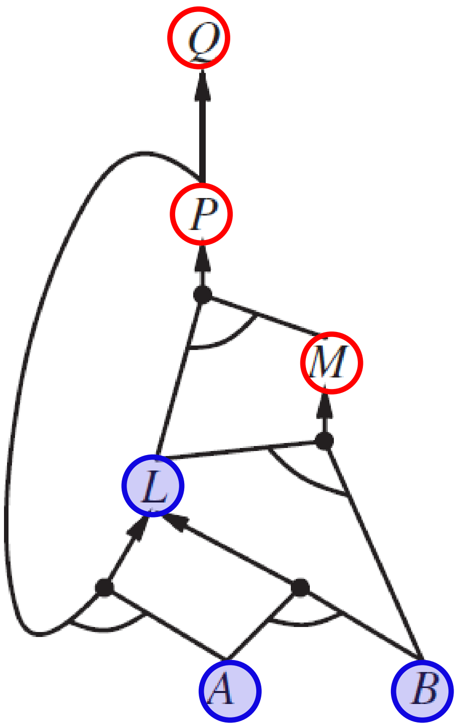
$B \land L \Rightarrow M$

$A \land P \Rightarrow L$

$A \land B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

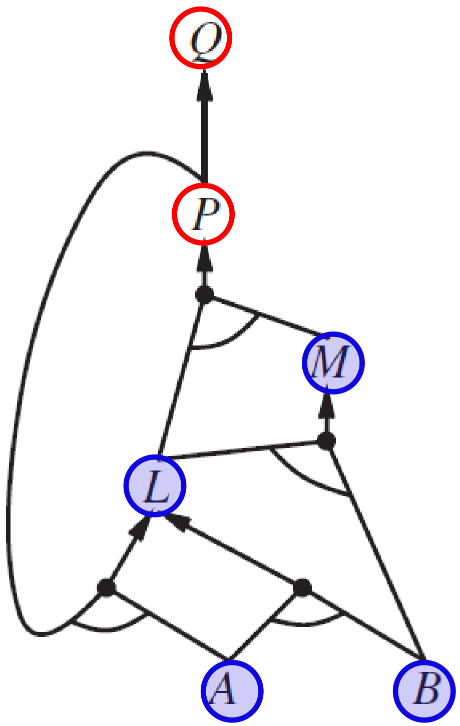
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

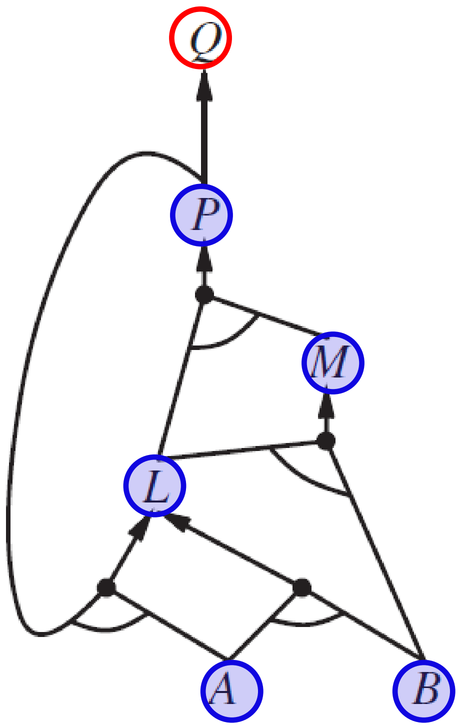
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

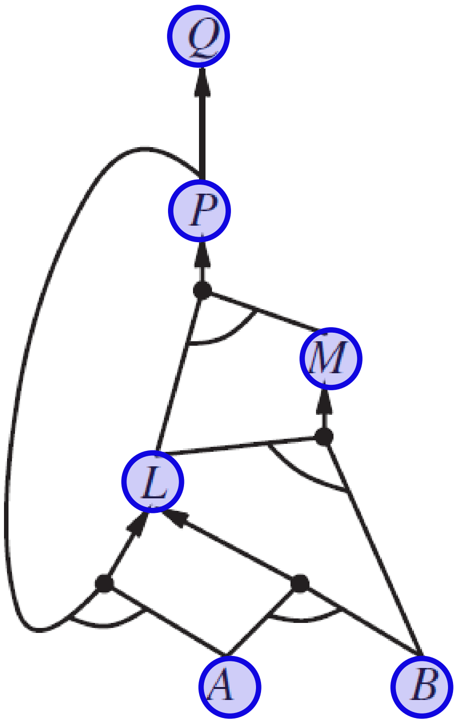
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Backward Chaining

- If q is true, no work is needed.
- Otherwise, finds implications in the KB whose conclusion is q.
- If all the premises of one of these implications can be proved true (recursively by backward chaining), then q is true.

Q. *KB* $\models Q$?

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

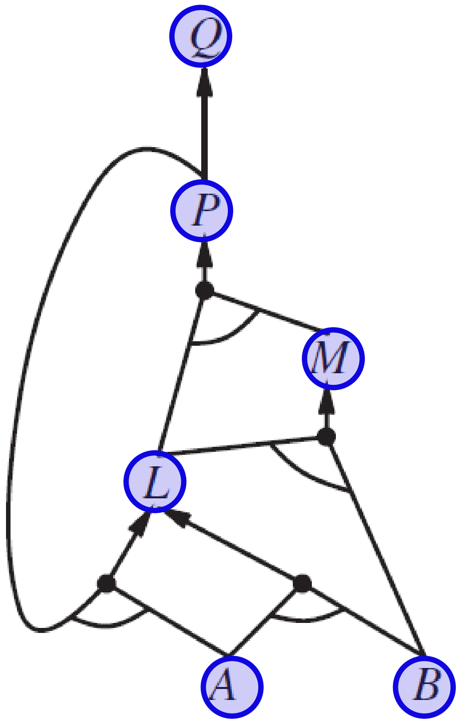
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



AND-OR graph search!

Forward vs. Backward Chaing

- Applicable range
  - Prove the entailment of a single proposition symbol
  - KB consists of definite clauses only.

Either P or $P_1 \land P_2 \land \cdots P_k \Rightarrow Q$

- Forward chaining is *data-driven*, automatic, unconscious processing.
- It may perform a lot of work that is irrelevant to the goal.
- Backward chaining is *goal-driven*, and appropriate for problem solving.
- It may run in time sublinear in the size of KB, since it touches only relevant facts.

II. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

II. Effective Propositional Model Checking

$KB \vDash \beta$ if and only if $\underbrace{KB \wedge \neg\beta}$ is unsatisfiable.

One sentence in propositional logic (PL)

II. Effective Propositional Model Checking

$KB \vDash \beta$ if and only if $\underbrace{KB \wedge \neg\beta}$ is unsatisfiable.

One sentence in propositional logic (PL)

Satisfiability problem Is a sentence s in PL satisfiable?

II. Effective Propositional Model Checking

$KB \vDash \beta$ if and only if $\underbrace{KB \wedge \neg\beta}$ is unsatisfiable.

One sentence in propositional logic (PL)

Satisfiability problem Is a sentence s in PL satisfiable?

- Cast the problem as one of constraint satisfaction.

Many combinatorial problems in computer science can be reduced to checking the satisfiability of a propositional sentence.

II. Effective Propositional Model Checking

$KB \models \beta$ if and only if $KB \wedge \neg\beta$ is unsatisfiable.

One sentence in propositional logic (PL)

Satisfiability problem Is a sentence s in PL satisfiable?

- Cast the problem as one of constraint satisfaction.

  Many combinatorial problems in computer science can be reduced to checking the satisfiability of a propositional sentence.

♦ Complete backtracking search (DPLL algorithm)

♦ Incomplete local search (WALKSAT algorithm)

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

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*, *model* ∪ {*P*=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if *P* is non-null **then return** DPLL(*clauses*, *symbols* – *P*, *model* ∪ {*P*=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {*P*=*true*}) **or**
        DPLL(*clauses*, *rest*, *model* ∪ {*P*=*false*}))

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ 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*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) **or**
      DPLL($clauses, rest, model \cup \{P{=}false\}$))

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

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, *model* ∪ {P=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
      DPLL(*clauses*, *rest*, *model* ∪ {P=*false*}))

Early termination: a clause is true if any of its literals is true. E.g., $A \lor \neg B \lor \neg C$ is true if A is true (regardless of the values assigned to B and C).

Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., A and B are pure in $A \lor \neg B$, $\neg B \lor \neg C$, $C \lor A$ while C is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \lor \neg C$, enabling C to become a pure symbol.

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

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, *model* ∪ {P=*value*})
  P, *value* ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
      DPLL(*clauses*, *rest*, *model* ∪ {P=*false*}))

Truth value to assign to P

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., A and B are pure in $A \vee \neg B$, $\neg B \vee \neg C$, $C \vee A$ while C is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \vee \neg C$, enabling C to become a pure symbol.

DPLL Algorithm

Davis, Putnam, Logemann, and Loveland (1960, 1962)

With enhancements, modern solvers can handle a problem with a multiple of 10^7 variables.

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*, { })

Truth value to assign to P

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

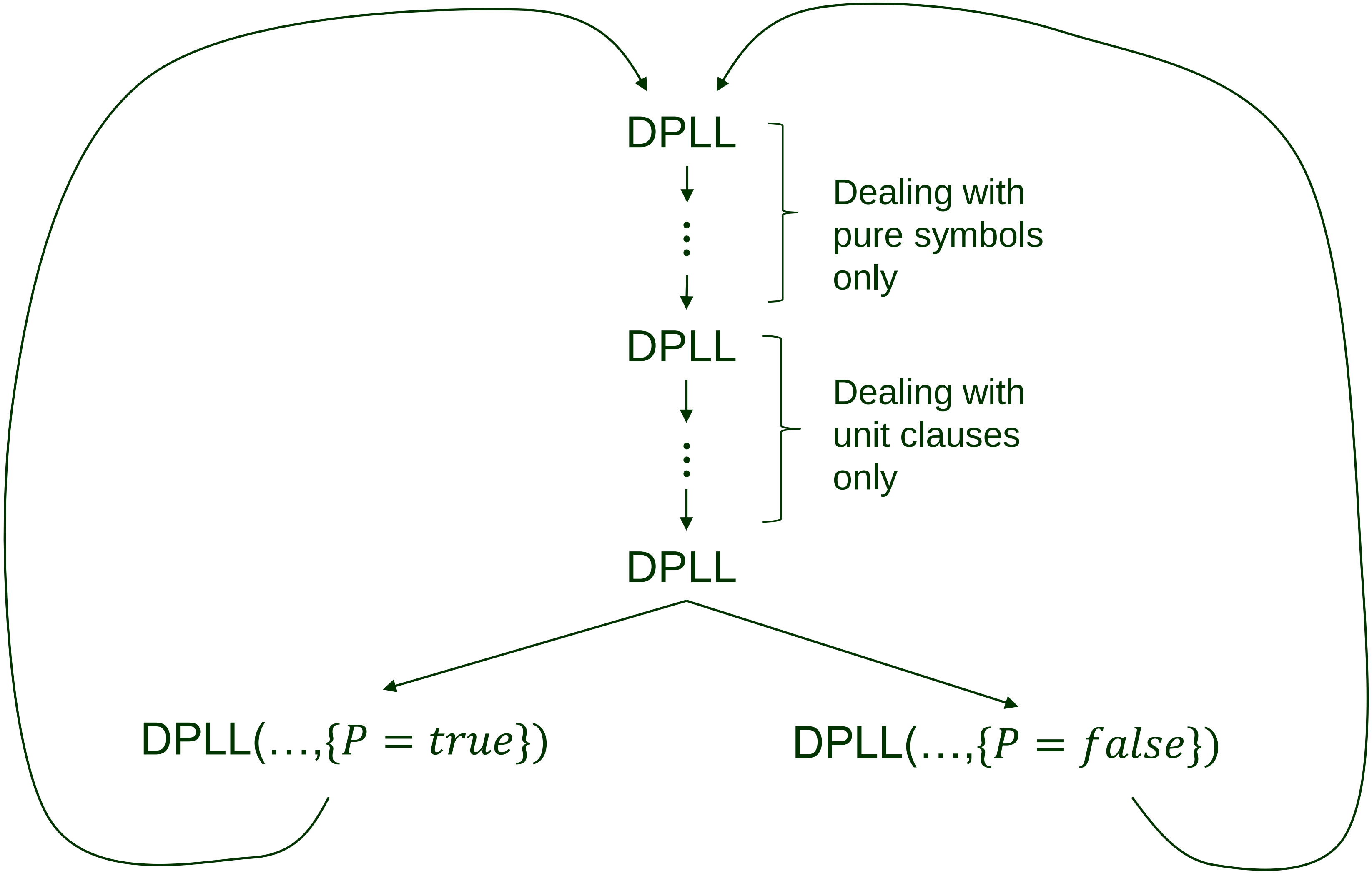
  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  if P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  P,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

Early termination: a clause is true if any of its literals is true. E.g., $A \vee \neg B \vee \neg C$ is true if A is true (regardless of the values assigned to B and C).

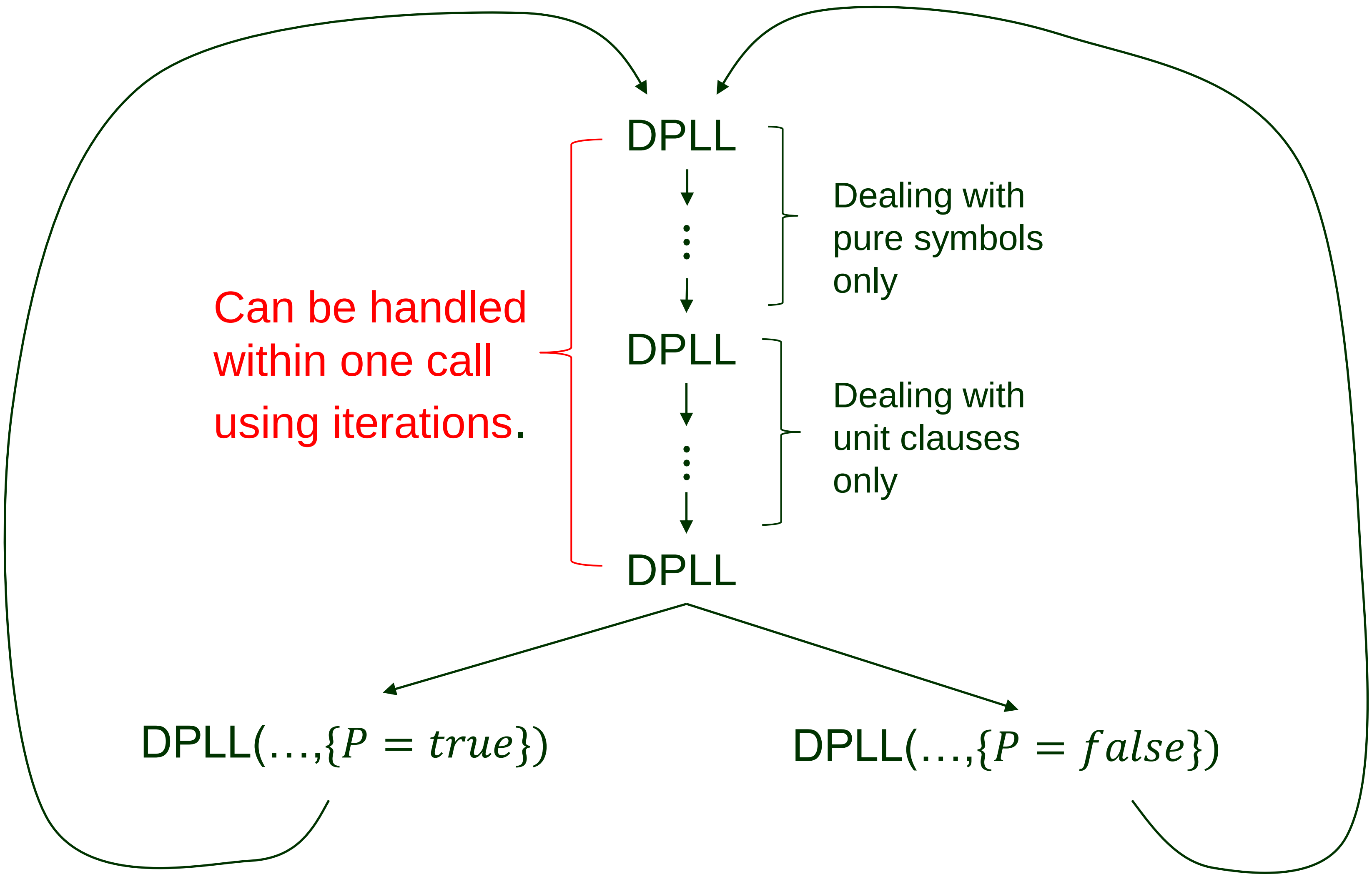
Pure symbol: a symbol appearing always positive or always negative in all clauses. E.g., A and B are pure in $A \vee \neg B$, $\neg B \vee \neg C$, $C \vee A$ while C is not pure. Assignment $A \leftarrow$ true will reduce the set to $\neg B \vee \neg C$, enabling C to become a pure symbol.

Unit clause propagation on a clause in which all literals but one are assigned *false*. E.g., $\neg B \vee \neg C$ simplifies to the unit clause $\neg C$ if B = *true*.

Recursion Tree



Recursion Tree



Local Search Algorithms

- Take steps in the space of complete assignments, flipping the truth value of one symbol at a time.
- Use an evaluation that counts the number of unsatisfied clauses.
- Escape local minima using various forms of randomness.
- Find a good balance between greediness and randomness.

The WALKSAT Algorithm

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, typically around 0.5
    max_flips, number of value flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for each $i = 1$ **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) $\leq p$ **then**
      flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

The WALKSAT Algorithm

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, typically around 0.5
max_flips, number of value flips allowed before giving up

```
model ← a random assignment of true/false to the symbols in clauses
for each 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 the number of satisfied clauses
return failure
```

The WALKSAT Algorithm

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, typically around 0.5
    max_flips, number of value flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for each $i = 1$ **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) $\leq p$ **then**
      flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

III. Agent Based on Propositional Logic

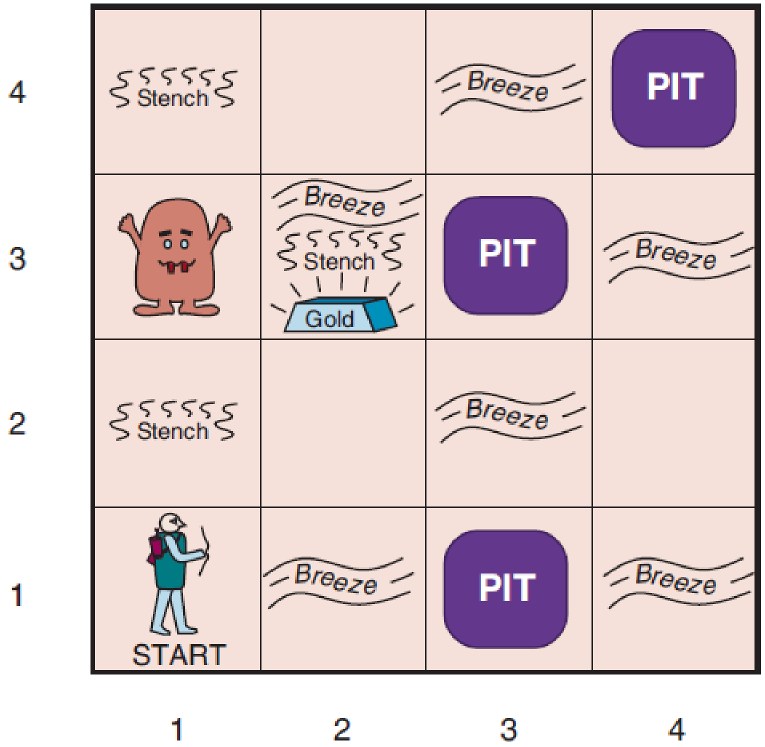
- Write down a complete logical model of the effects of action.
- How logical inference can be used by an agent?.
- How to keep track of the world without resorting to inference history?
- How to use logical inference to construct plans based on the KB?

Knowledge base (KB):

♣ general knowledge about how the world works

♣ percept sentences obtained in a particular world

Current State in the Wumpus World



Axioms:

$\neg P_{1,1}$ $\neg W_{1,1}$

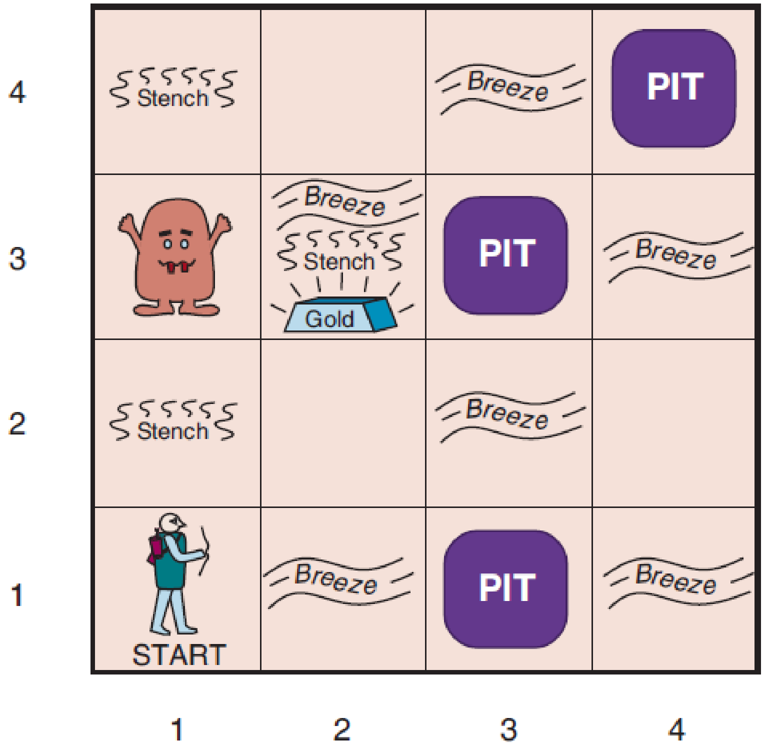
$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$ // 16 rules of this type

$S_{1,1} \Leftrightarrow (W_{1,2} \lor W_{2,1})$ // 16

…

- $P_{x,y} =$ *true* if there is a pit in $[x, y]$.
- $W_{x,y} =$ *true* if there is a wumpus in $[x, y]$, dead or alive.
- $B_{x,y} =$ *true* if the agent perceives a breeze in $[x, y]$.
- $S_{x,y} =$ *true* if the agent perceives a stench in $[x, y]$.

Current State in the Wumpus World



Axioms:

$\neg P_{1,1}$ $\neg W_{1,1}$

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ // 16 rules of this type

$S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1})$ // 16

$\dots$

- Exactly one wumpus

$W_{1,1} \vee W_{1,2} \vee \cdots \vee W_{4,3} \vee W_{4,4}$ // ≥ 1 wumpus

$\neg W_{i,j} \vee \neg W_{k,l}$ $1 \leq i,j,k,l \leq 4$ and $(i,j) \neq (k,l)$

// ≤ 1 Wumpus;

// $\frac{16\times 15}{2} = 120$ rules

- $P_{x,y} =$ *true* if there is a pit in $[x, y]$.
- $W_{x,y} =$ *true* if there is a wumpus in $[x, y]$, dead or alive.
- $B_{x,y} =$ *true* if the agent perceives a breeze in $[x, y]$.
- $S_{x,y} =$ *true* if the agent perceives a stench in $[x, y]$.

Representing Percepts

A percept asserts something only about the current time.

$Stench^4$: the agent senses stench at time step 4 (in square A).

$\neg Stench^3$: the agent senses no stench at time step 3 (in square B).

Representing Percepts

A percept asserts something only about the current time.

Stench ^{4}: the agent senses stench at time step 4 (in square A).

$\neg$*Stench* ^{3}: the agent senses no stench at time step 3 (in square B).

Associate propositions with time steps for aspects of the world that changes over time.

$L_{1,1}^0$: the agent is in square $[1, 1]$ at time step 0.

Representing Percepts

A percept asserts something only about the current time.

Stench ^{4}: the agent senses stench at time step 4 (in square A).

$\neg$*Stench* ^{3}: the agent senses no stench at time step 3 (in square B).

Associate propositions with time steps for aspects of the world that changes over time.

$L^0_{1,1}$: the agent is in square $[1, 1]$ at time step 0.

For any time step t and any square $[x, y]$,

Representing Percepts

A percept asserts something only about the current time.

Stench ^{4}: the agent senses stench at time step 4 (in square A).

$\neg$*Stench* ^{3}: the agent senses no stench at time step 3 (in square B).

Associate propositions with time steps for aspects of the world that changes over time.

$L^0_{1,1}$: the agent is in square $[1, 1]$ at time step 0.

For any time step t and any square $[x, y]$,

$$L^t_{x,y} \Rightarrow (\mathit{Breeze}^{\,t} \Leftrightarrow B_{x,y})$$

$$L^t_{x,y} \Rightarrow (\mathit{Stench}^{\,t} \Leftrightarrow S_{x,y})$$

Representing Percepts

A percept asserts something only about the current time.

$Stench^4$: the agent senses stench at time step 4 (in square A).

$\neg Stench^3$: the agent senses no stench at time step 3 (in square B).

Associate propositions with time steps for aspects of the world that changes over time.

$L^0_{1,1}$: the agent is in square $[1, 1]$ at time step 0.

For any time step t and any square $[x, y]$,

fluent — $L^t_{x,y} \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})$

(aspect changing with time) $L^t_{x,y} \Rightarrow (Stench^t \Leftrightarrow S_{x,y})$

Describing a Transition Model

Forward^t: the agent executes the forward action at time t.

Describing a Transition Model

Forward^t: the agent executes the forward action at time t.

Effect axioms specify outcome of an action at the next time step.

$$L^0_{1,1} \land \mathit{FacingEast}^0 \land \mathit{Forward}^0 \Rightarrow (L^1_{2,1} \land \neg L^1_{1,1})$$

// if the agent is at [1,1] facing east at time 0 and goes forward,
// the result is that the agent is in [2,1] and no longer in [1,1].

Describing a Transition Model

Forward^t: the agent executes the forward action at time t.

Effect axioms specify outcome of an action at the next time step.

$$L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$$

// if the agent is at [1,1] facing east at time 0 and goes forward,
// the result is that the agent is in [2,1] and no longer in [1,1].

Frame axioms assert all the proportions that remain the same.

Describing a Transition Model

$Forward^t$: the agent executes the forward action at time t.

Effect axioms specify outcome of an action at the next time step.

$$L_{1,1}^0 \land FacingEast^0 \land Forward^0 \Rightarrow (L_{2,1}^1 \land \neg L_{1,1}^1)$$

// if the agent is at [1,1] facing east at time 0 and goes forward,
// the result is that the agent is in [2,1] and no longer in [1,1].

Frame axioms assert all the proportions that remain the same.

$$Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrorw^{t+1})$$

$$Forward^t \Rightarrow (WumpusAlive^t \Leftrightarrow WumpusAlive^{t+1})$$

Describing a Transition Model

$Forward^t$: the agent executes the forward action at time t.

Effect axioms specify outcome of an action at the next time step.

$$L^0_{1,1} \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$$

// if the agent is at [1,1] facing east at time 0 and goes forward,
// the result is that the agent is in [2,1] and no longer in [1,1].

Frame axioms assert all the proportions that remain the same.

$$Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrorw^{t+1})$$

$$Forward^t \Rightarrow (WumpusAlive^t \Leftrightarrow WumpusAlive^{t+1})$$

$O(mn)$ frame axioms for m actions and n fluents

Axioms for Successor States

Successor-state axiom, one for every fluent F, states that

- either the action at t causes F to be true at $t+1$,
- or F was already true at t and the action does not cause it to be false.

$$F^{t+1} \Leftrightarrow \mathit{ActionCausesF}^t \vee (F^t \wedge \neg \mathit{ActionCausesNotF}^t)$$

Axioms for Successor States

Successor-state axiom, one for every fluent F, states that

- either the action at t causes F to be true at $t+1$,
- or F was already true at t and the action does not cause it to be false.

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$$

$$HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$$

Axioms for Successor States

Successor-state axiom, one for every fluent F, states that

- either the action at t causes F to be true at $t+1$,
- or F was already true at t and the action does not cause it to be false.

$$F^{t+1} \Leftrightarrow \mathit{ActionCausesF}^t \vee (F^t \wedge \neg \mathit{ActionCausesNotF}^t)$$

$$\mathit{HaveArrow}^{t+1} \Leftrightarrow (\mathit{HaveArrow}^t \wedge \neg \mathit{Shoot}^t)$$

// no action, e.g., for reloading

Axioms for Successor States

Successor-state axiom, one for every fluent F, states that

- either the action at t causes F to be true at $t+1$,
- or F was already true at t and the action does not cause it to be false.

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$$

$$HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$$

// no action, e.g., for reloading

$$L_{1,1}^{t+1} \Leftrightarrow (L_{1,1}^t \wedge (\neg Forward^t \vee Bump^{t+1})) \vee (L_{1,2}^t \wedge (FacingSouth^t \vee Forward^t))$$
$$\vee (L_{2,1}^t \wedge (FacingWest^t \vee Forward^t))$$

Axioms for Successor States

Successor-state axiom, one for every fluent F, states that

- either the action at t causes F to be true at $t+1$,
- or F was already true at t and the action does not cause it to be false.

$$F^{t+1} \Leftrightarrow \mathit{ActionCausesF}^t \vee (F^t \wedge \neg \mathit{ActionCausesNotF}^t)$$

$$\mathit{HaveArrow}^{t+1} \Leftrightarrow (\mathit{HaveArrow}^t \wedge \neg \mathit{Shoot}^t)$$

// no action, e.g., for reloading

$$L_{1,1}^{t+1} \Leftrightarrow (L_{1,1}^t \wedge (\neg \mathit{Forward}^t \vee \mathit{Bump}^{t+1})) \vee (L_{1,2}^t \wedge (\mathit{FacingSouth}^t \vee \mathit{Forward}^t)) \vee (L_{2,1}^t \wedge (\mathit{FacingWest}^t \vee \mathit{Forward}^t))$$

Square-OK axiom asserts that a square is free of a pit or live Wumpus.

$$OK_{x,y}^t \Leftrightarrow \neg P_{x,y} \wedge \neg (W_{x,y} \wedge \mathit{WumpusAlive}^t)$$

Initial Percepts and Actions

$\neg Stench^0 \wedge \neg Breeze^0 \wedge \neg Glitter^0 \wedge \neg Bump^0 \wedge \neg Scream^0$; $Forward^0$

$\neg Stench^1 \wedge Breeze^1 \wedge \neg Glitter^1 \wedge \neg Bump^1 \wedge \neg Scream^1$; $TurnRight^1$

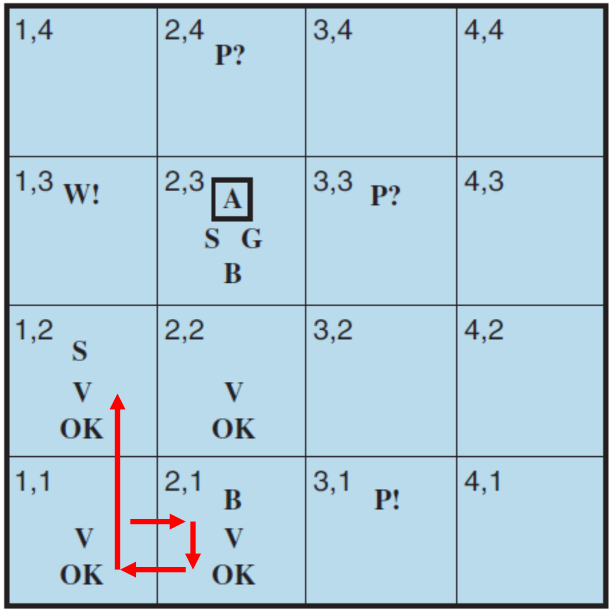
$\neg Stench^2 \wedge Breeze^2 \wedge \neg Glitter^2 \wedge \neg Bump^2 \wedge \neg Scream^2$; $TurnRight^2$

$\neg Stench^3 \wedge Breeze^3 \wedge \neg Glitter^3 \wedge \neg Bump^3 \wedge \neg Scream^3$; $Forward^3$

$\neg Stench^4 \wedge \neg Breeze^4 \wedge \neg Glitter^4 \wedge \neg Bump^4 \wedge \neg Scream^4$; $TurnRight^4$

$\neg Stench^5 \wedge \neg Breeze^5 \wedge \neg Glitter^5 \wedge \neg Bump^5 \wedge \neg Scream^5$; $Forward^5$

$Stench^6 \wedge \neg Breeze^6 \wedge \neg Glitter^6 \wedge \neg Bump^6 \wedge \neg Scream^6$



Query the knowledge base:

$$\text{ASK}(KB, L^6_{1,2}) = true$$

$$\text{ASK}(KB, W_{1,3}) = true$$

$$\text{ASK}(KB, P_{3,1}) = true$$

$$\text{ASK}(KB, OK^6_{2,2}) = true$$

// the square [2,2] is OK to move into.