Chapter 7
Logical Agents

Outline

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

KNOWLEDGE-BASED AGENTS

- The central component of a knowledge-based agent is its **knowledge base, or KB.** A knowledge base is a set of sentences
- Each sentence is expressed in a language called a knowledge representation language and represents some assertion about the world.
- **Axiom :** when the sentence is taken as given without being derived from other sentences.
- There must be a way to add new sentences to the knowledge base and a way to query what is known. The standard names for these operations are TELL and ASK
- Both operations may involve inference—that is, deriving new sentences from old. Inference must obey the requirement that when one ASKs a question of the knowledge base, the answer should follow from what has been told (or TELLed) to the knowledge base previously.
- Each time the agent program is called, it does three things. First, it TELLs the knowledge base what it perceives. Second, it ASKs the knowledge base what action it should perform. In the process of answering this query, extensive reasoning may be done about the current state of the world, about the outcomes of possible action sequences, and so on. Third, the agent program TELLs the knowledge base which action was chosen, and the agent executes the action.

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

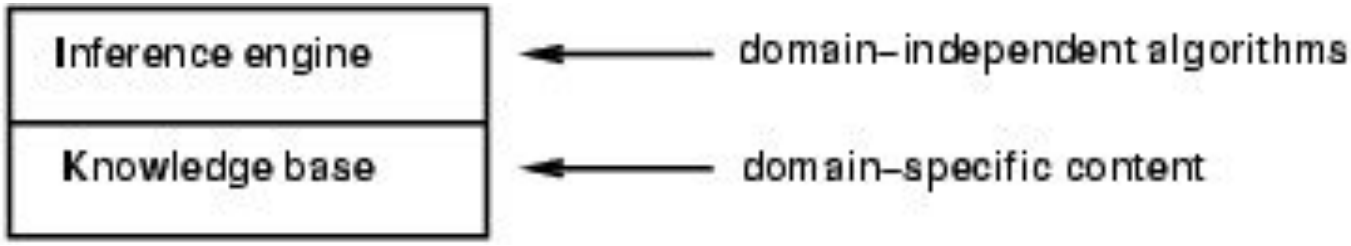
MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time.

MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.

MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed

- A knowledge-based agent can be built simply by TELLing it what it needs to know.
- Starting with an empty knowledge base, the agent designer can TELL sentences one by one until the agent knows how to operate in its environment. This is called the **declarative approach** to system building. In contrast, the **procedural approach** encodes desired behaviors directly as program code.

Knowledge bases



- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - `Tell` it what it needs to know
- Then it can `Ask` itself what to do - answers should follow from the KB
- Agents can be viewed at the knowledge level
  i.e., what they know, regardless of how implemented

- Or at the implementation level
  - i.e., data structures in KB and algorithms that manipulate them

THE WUMPUS WORLD

- The **wumpus world** is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the terrible wumpus, a beast that eats anyone who enters its room.
- The wumpus can be shot by an agent, but the agent has only one arrow. Some rooms contain bottomless pits that will trap anyone who wanders into these rooms The only mitigating feature of this bleak environment is the possibility of finding a heap of gold.

PEAS description

- **Performance measure: +1000 for climbing out of the cave with the gold, –1000 for** falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.
- **Environment:** A 4×4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.

- **Actuators:** The agent can move *Forward, TurnLeft by 90◦, or TurnRight by 90◦. **The*** agent dies a miserable death if it enters a square containing a pit or a live wumpus. If an agent tries to moveforward and bumps into a wall, then the agent does not move. The action *Grab can be* used to pick up the gold if it is in the same square as the agent. The action *Shoot can* be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first *Shoot action has any effect. Finally, the action Climb* can be used to climb out of the cave, but only from square [1,1].
- **Sensors: The agent has five sensors, each of which gives a single bit of information:**

– In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a *Stench.*

– In the squares directly adjacent to a pit, the agent will perceive a *Breeze.*

– In the square where the gold is, the agent will perceive a *Glitter.*

– When an agent walks into a wall, it will perceive a *Bump.*

– When the wumpus is killed, it emits a woeful *Scream that can be perceived anywhere*in the cave.

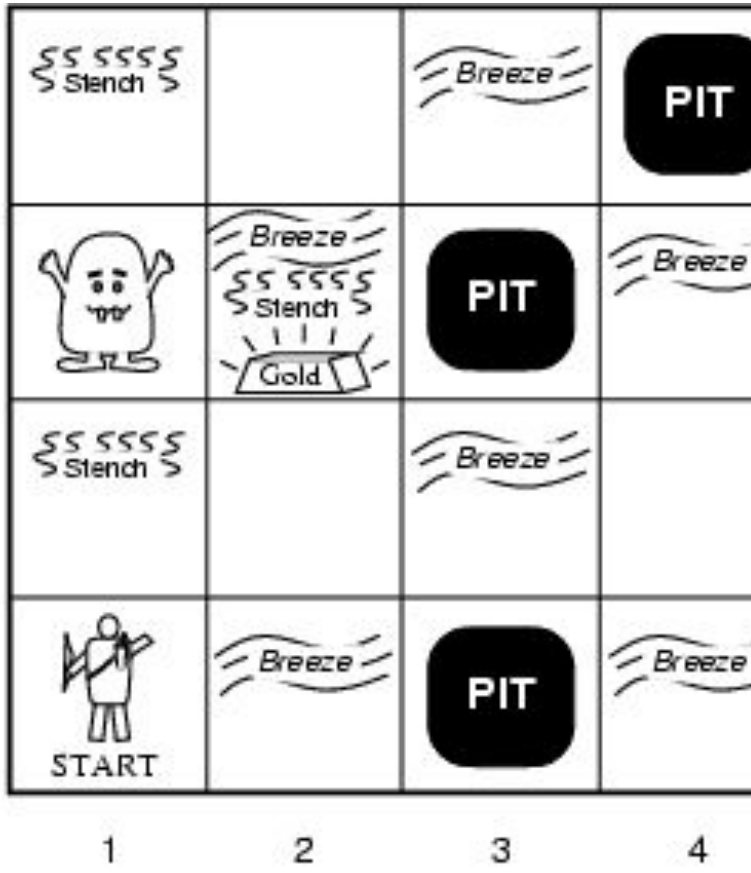
Wumpus World PE

- Performance measure
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow

- Environment

Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter iff gold is in the same square
Shooting kills wumpus if you are facing itShooting uses up the only arrow
Grabbing picks up gold if in same square
Releasing drops the gold in same square

- Sensors: Stench, Breeze, Glitter, Bu…p, Sc…am
  Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot



Wumpus world characterization

- Fully Observable No – only local perception
- Deterministic Yes – outcomes exactly specified
- Episodic No – sequential at the level of actions
- Static Yes – Wumpus and Pits do not move
- Discrete Yes
- Single-agent? Yes – Wumpus is essentially a natural feature

Wumpus world

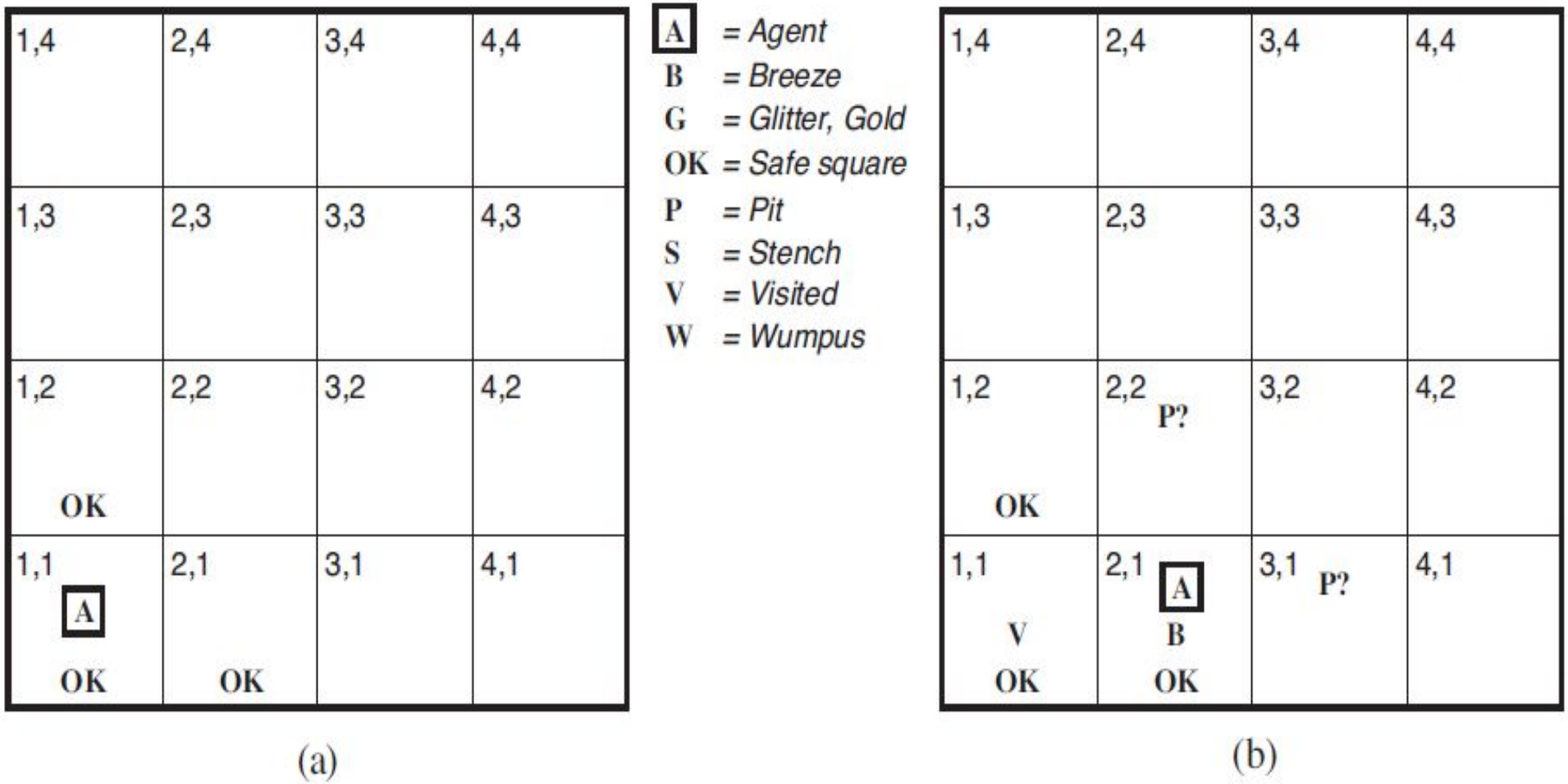


Figure 7.3 The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None*, *None*, *None*, *None*, *None*]. (b) After one move, with percept [*None*, *Breeze*, *None*, *None*, *None*].

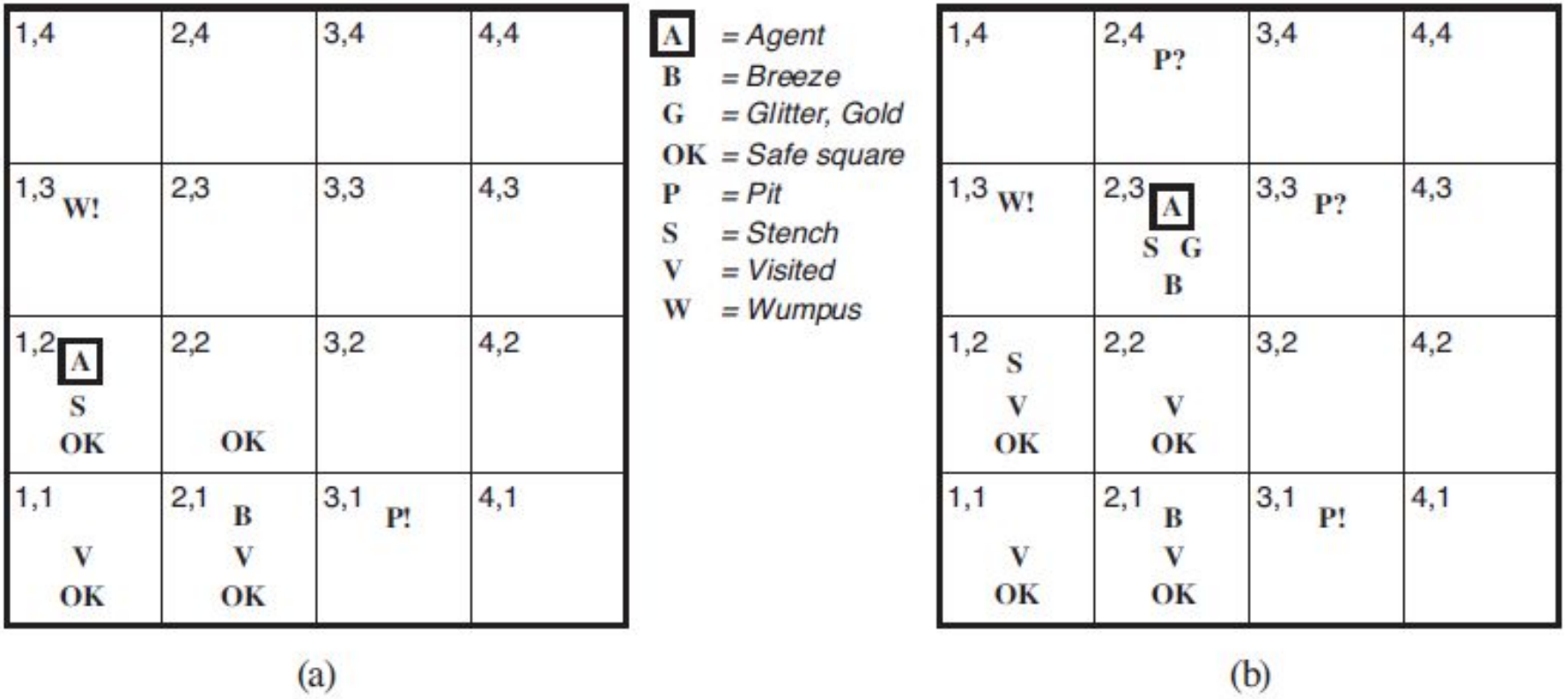
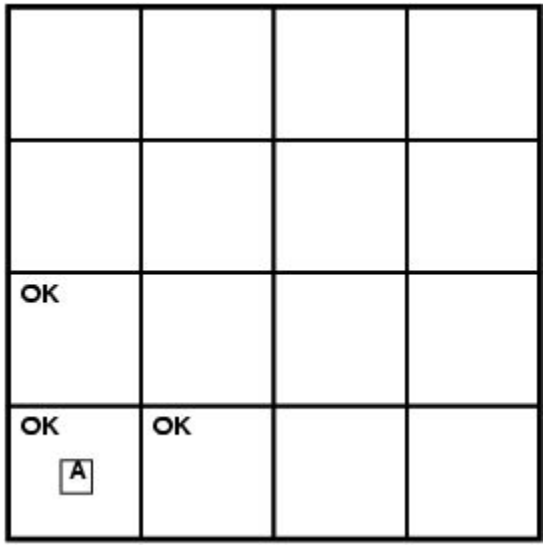
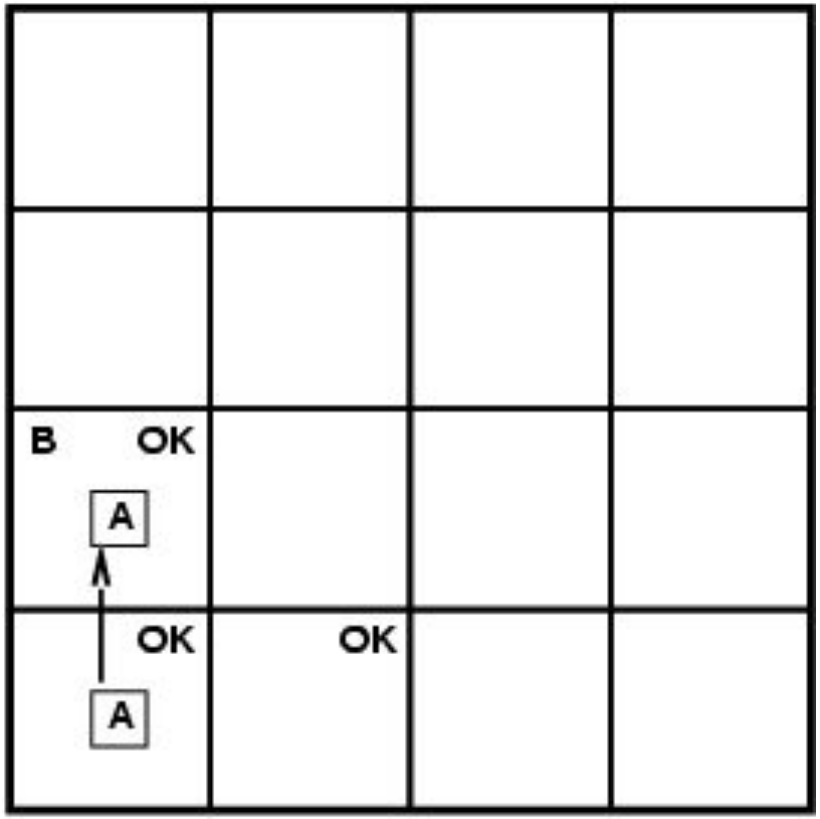


Figure 7.4 Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench*, *None*, *None*, *None*, *None*]. (b) After the fifth move, with percept [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

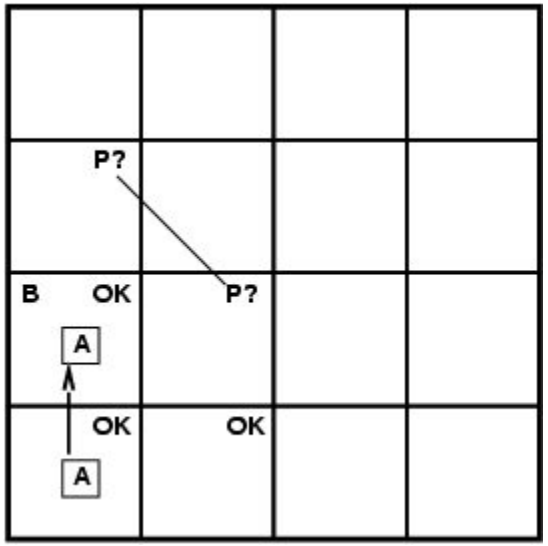
Exploring a wumpus world



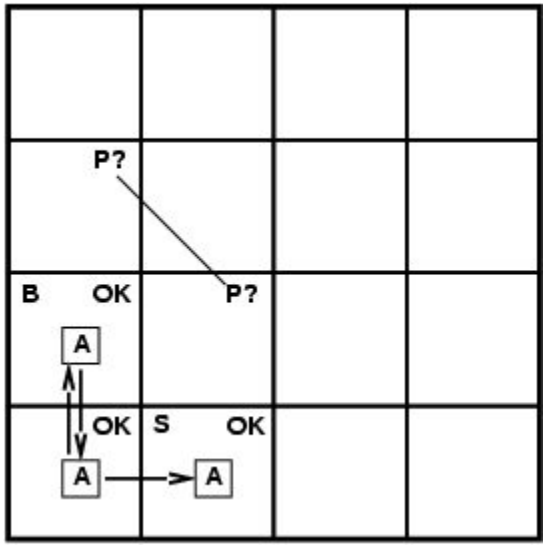
Exploring a wumpus world



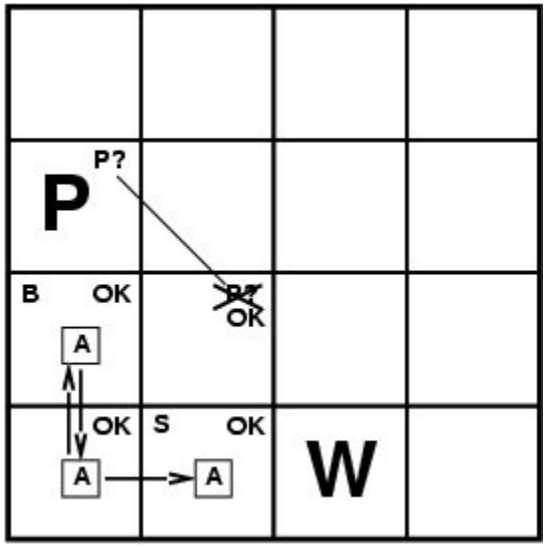
Exploring a wumpus world



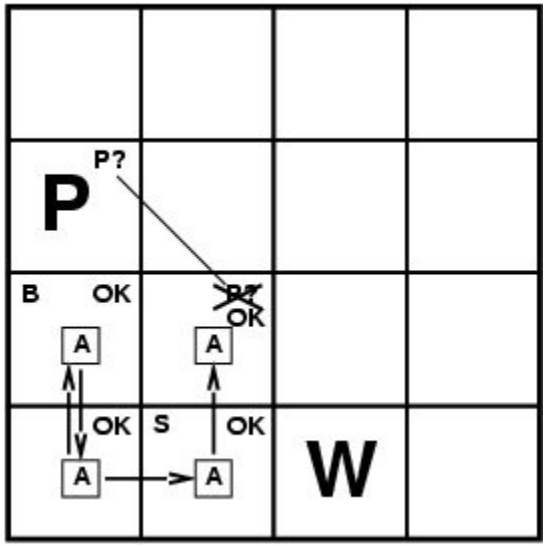
Exploring a wumpus world



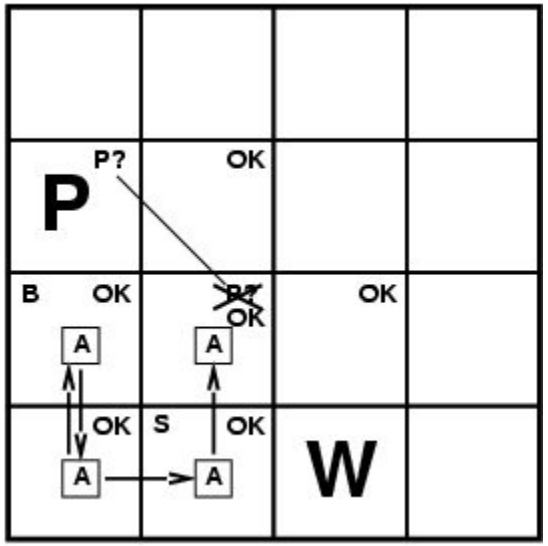
Exploring a wumpus world



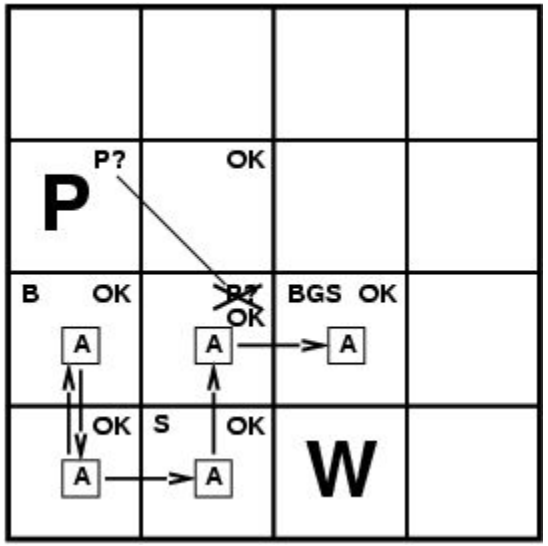
Exploring a wumpus world



Exploring a wumpus world



Exploring a wumpus world



Logic in general

- Logics are formal languages for representing information such that conclusions can be drawn
- Knowledge bases consist of sentences. These sentences

are expressed according to the **syntax** of the representation language, which specifies all the sentences that are well formed.

- Syntax defines the sentences in the language
  A logic must also define the **semantics** or meaning of sentences. The semantics defines the truth of each sentence with respect to each possible world.
  E.g., the semantics for arithmetic specifies that the sentence "x + y =4" is true in a world where x is 2 and y is 2, but false in a world where x is 1 and y is 1.
- In standard logics, every sentence must be either true or false in each possible world—there is no "in between."

Logic

- possible worlds might be thought of as (potentially) real environments that the agent might or might not be in, models are mathematical abstractions, each of which simply fixes the truth or falsehood of every relevant sentence.
- Having x men and y women sitting at a table playing bridge, and the sentence x + y =4 is true when there are four people in total. Formally, the possible models are just all possible assignments of real numbers to the variables x and y. Each such assignment fixes the truth of any sentence of arithmetic whose variables are x and y.

Logic

- If a sentence α is true in model m, then m satisfies α or sometimes m is a model of α.
- M(α) : the set of all models of α.
- The relation of **logical entailment** between sentence gives the idea that a sentence follows logically from another sentence.
- α |= β means sentence α entails the sentence β
- α |= β if and only if, in every model in which α is true, β is also true
- α |= β if and only if $M(\alpha) \subseteq M(\beta)$.
- sentence x = 0 entails the sentence xy = 0.

Model

- Consider the situation in Figure 7.3(b): the agent has detected nothing in [1,1] and a breeze in [2,1]. These percepts, combined with the agent's knowledge of the rules of the wumpus world, constitute the KB.

- The agent is interested (among other things) in whether the adjacent squares [1,2], [2,2], and [3,1] contain pits. Each of the three squares might or might not contain a pit, so (for the purposes of this example) there are $2^3 = 8$ possible models.

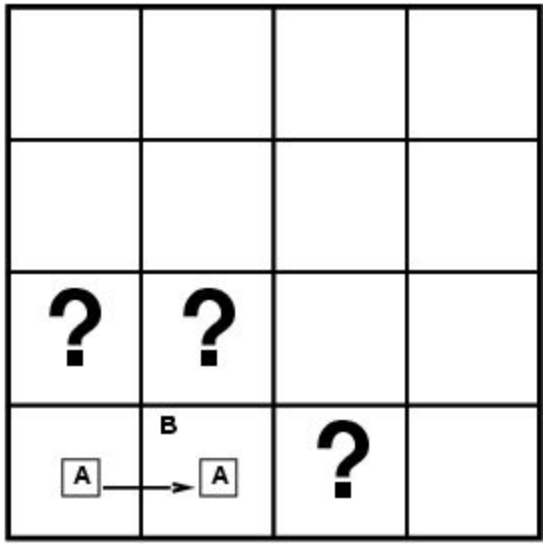
Entailment in the wumpus world

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

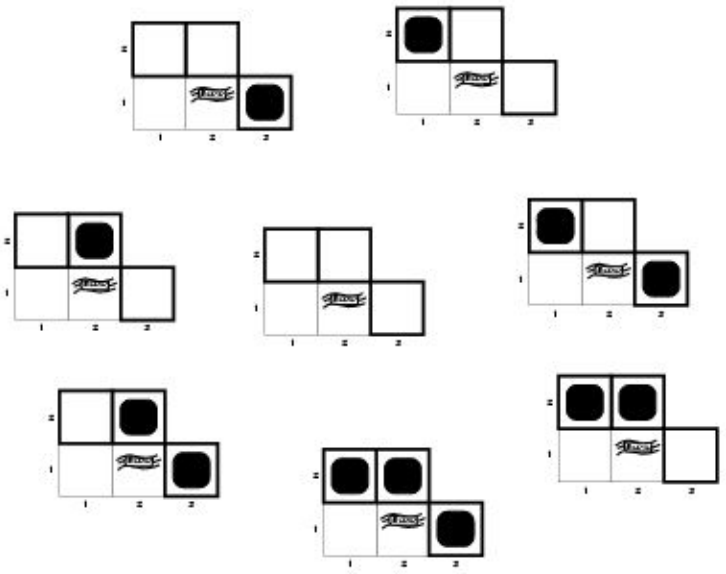
Consider possible models for *KB* assuming only pits

- The agent is interested in whether the adjacent squares [1,2], [2,2], and [3,1] contain pits.

3 Boolean choices ⇒ 8 possible models



Wumpus models



Knowledge Base

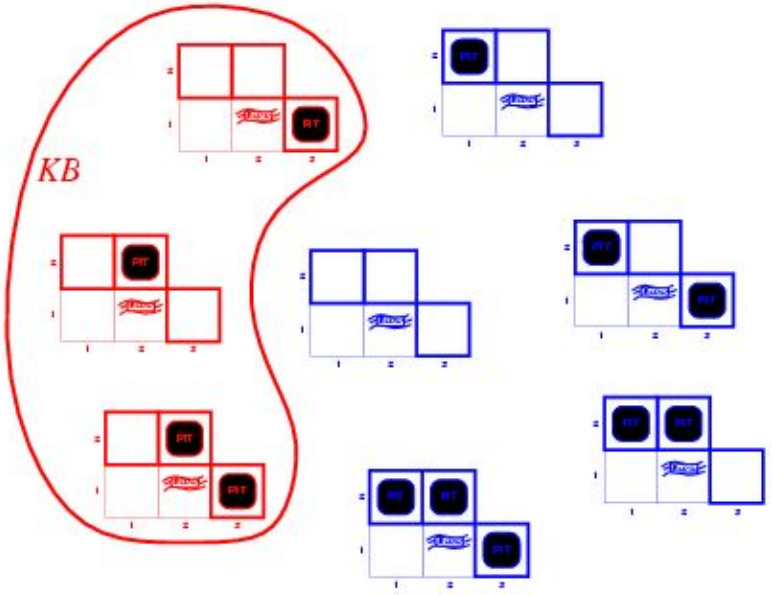
The KB can be thought of as a set of sentences or as a single sentence that asserts all the individual sentences. The KB is false in models that contradict what the agent knows— for example, the KB is false in any model in which [1,2] contains a pit, because there is no breeze in [1,1].

There are in fact just three models in which the KB is true,

- consider two possible conclusions:

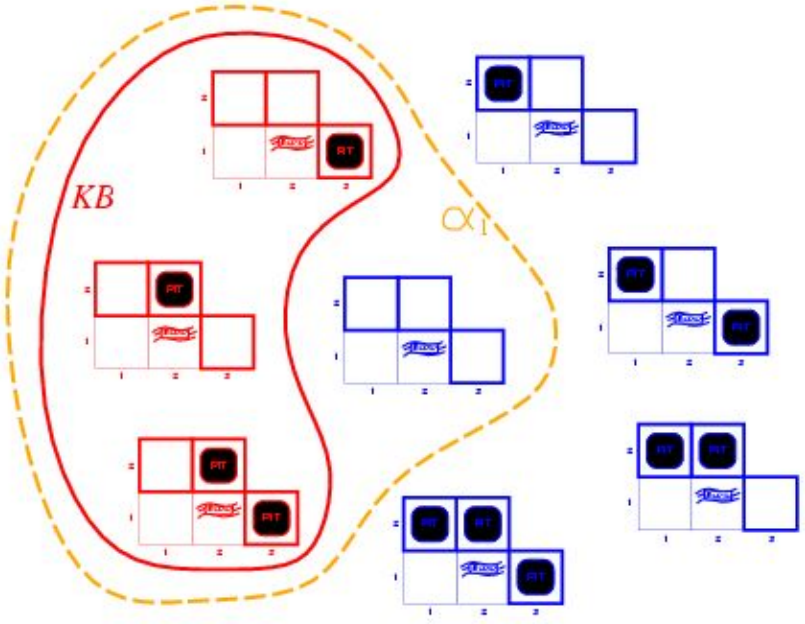
α1 = “There is no pit in [1,2].”
α2 = “There is no pit in [2,2].”

Wumpus models



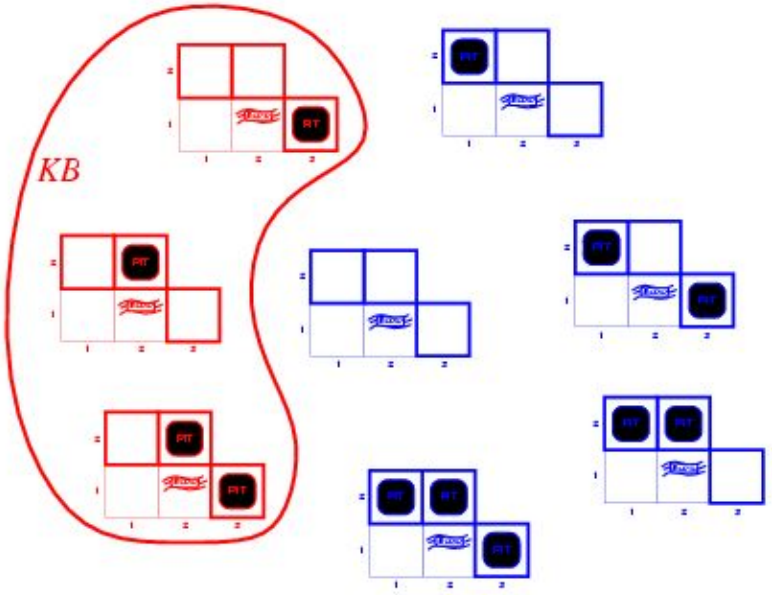
- *KB* = wumpus-world rules + observations

Wumpus models



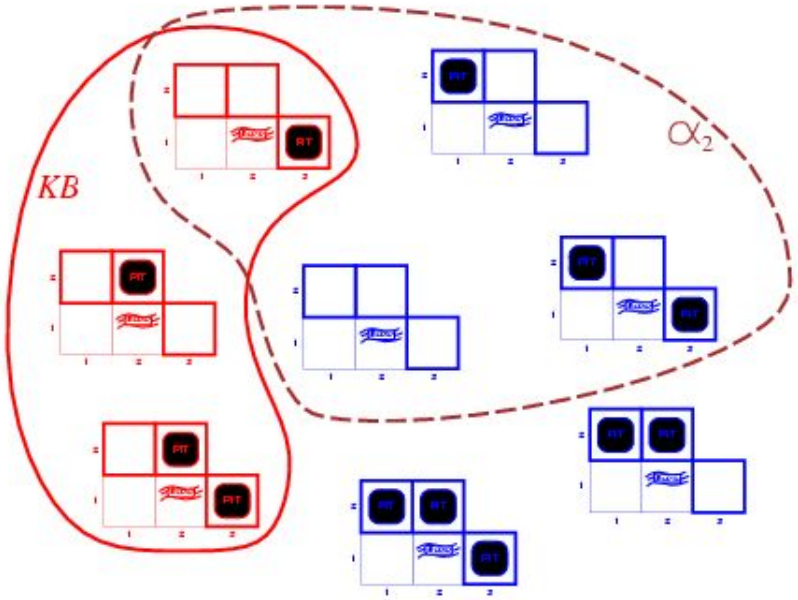
- *KB* = wumpus-world rules + observations
- α1 = “There is no pit in [1,2].” ie α_1 = "[1,2] is safe",
- in every model in which KB is true, α1 is also true.
- *KB* $\models \alpha_1$, proved by model checking
  model checking: enumerates all possible models to check that α is true in all models in which KB is true, that is, $M(KB) \subseteq M(\alpha)$.
- Entailment can be used to carry out **logical inference**

Wumpus models



- *KB* = wumpus-world rules + observations

Wumpus models



- *KB* = wumpus-world rules + observations
- α_2 = "[2,2] is safe", ie α2 = “There is no pit in [2,2].”
- In some models in which KB is true, α2 is false.
- Hence, KB $\not\models$ α2: the agent *cannot conclude that there is no pit in [2,2].*

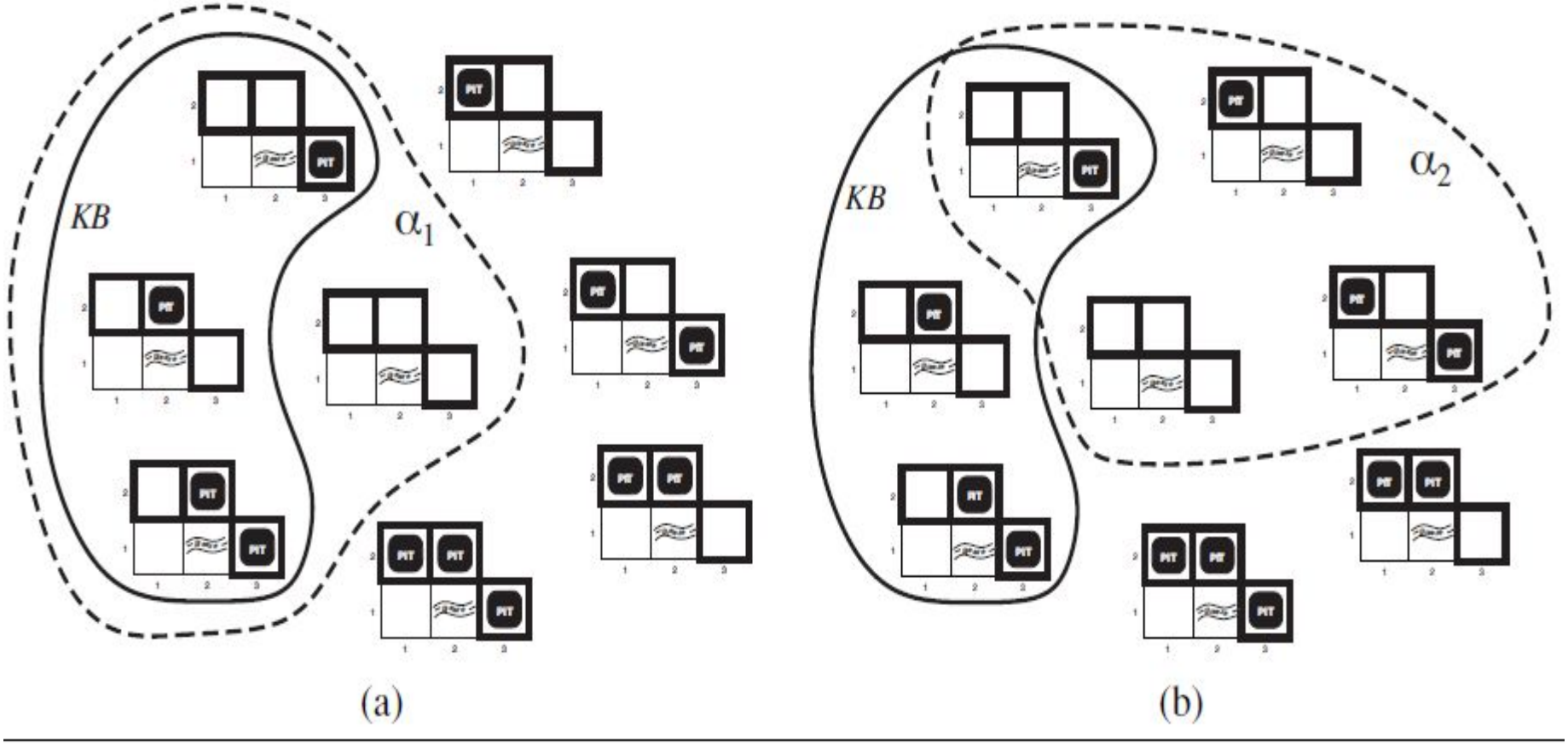


Figure 7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).

Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from *KB* by procedure *I*
  Soundness: An inference algorithm that derives only entailed sentences is called **sound or truth- preserving.**

- *i* is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
  Completeness: an inference algorithm is complete if it can derive any sentence that is entailed. *i* is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$
  if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world.

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas
- The syntax of propositional logic defines the allowable sentences
- The **atomic sentences** consist of a single **proposition symbol.** Each such symbol stands for a proposition that can be true or false.
- W1,3 to stand for the proposition that the wumpus is in [1,3]. The proposition symbols P_1, P_2 etc are sentences
- There are two proposition symbols with fixed meanings: True is the always-true proposition and False is the always-false proposition.
- Complex sentences are constructed from simpler sentences, using parentheses and logical **connectives.**

Connectives

- ¬ (not) If $W_{1,3}$ is a sentence, ¬ $W_{1,3}$ is a sentence (negation) A literal is either an atomic sentence (a positive literal) or a negated atomic sentence (a negative literal).

  $\wedge$ (and) A sentence whose main connective is $\wedge$, such as $W_{1,3} \wedge P_{3,1}$, is called a **conjunction; its parts are the conjuncts.**

- $\vee$ (or). A sentence using $\vee$, such as $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$, is a **disjunction of the disjuncts** $(W_{1,3} \wedge P_{3,1})$ and $W_{2,2}$.

  $\Rightarrow$ (implies) A sentence such as $(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$ is called an **implication (or con**ditional). Its **premise or antecedent** is $(W_{1,3} \wedge P_{3,1})$, and its **conclusion or consequent** is $\neg W_{2,2}$. Implications are also known as **rules or if–then statements.** $\Rightarrow$ symbol is sometimes written in other books as $\supset$ or $\rightarrow$.

  $\Leftrightarrow$ (if and only if). The sentence $W_{\quad} \Leftrightarrow \neg W_{\quad}$ is a

$$
\begin{aligned}
\mathit{Sentence} &\rightarrow \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} &\rightarrow \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} &\rightarrow (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
&\mid \neg\, \mathit{Sentence} \\
&\mid \mathit{Sentence} \wedge \mathit{Sentence} \\
&\mid \mathit{Sentence} \vee \mathit{Sentence} \\
&\mid \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
&\mid \mathit{Sentence} \Leftrightarrow \mathit{Sentence}
\end{aligned}
$$

$$
\text{OPERATOR PRECEDENCE} \quad : \quad \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow
$$

A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Propositional logic: Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model. Each model specifies true/false for each proposition symbol one possible model is m1 = {$P_{1,2}$ =false, $P_{2,2}$ =false, $P_{3,1}$ =true}
- With these symbols, 8 possible models, can be enumerated automatically.
- The models are purely mathematical objects with no necessary connection to wumpus worlds. $P_{1,2}$ is just a symbol; it might mean "there is a pit in [1,2]" or "I'm in Paris today and tomorrow."
- The semantics for propositional logic must specify how to compute the truth value of

Semantics

- True is true in every model and False is false in every model. The truth value of every other proposition symbol must be specified directly in the model.
- For complex sentences, five rules, which hold for any subsentence P and Q in any model m
- ¬P is true iff P is false in m.
- P ∧ Q is true iff both P and Q are true in m.
- P ∨ Q is true iff either P or Q is true in m.
- P ⇒ Q is true unless P is true and Q is false in m.
- P ⇔ Q is true iff P and Q are both true or both false in m. ces P and Q in any model m

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

the sentence
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1})$, evaluated in m1, gives
true $\wedge$ (false $\vee$ true)
=true $\wedge$ true
=true.

Wumpus world sentences

- $P_{x,y}$ is true if there is a pit in [x, y].
- $W_{x,y}$ is true if there is a wumpus in [x, y], dead or alive.
- $B_{x,y}$ is true if the agent perceives a breeze in [x, y].
- $S_{x,y}$ is true if the agent perceives a stench in [x, y].
  - There is no pit in [1,1]:

    $R_1 : \neg P_{1,1}$
- "Pits cause breezes in adjacent squares“- A square is breezy if and only if there is a pit in a neighboring square.

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

- include the breeze percepts for the first two squares visited in the specific world the agent is in, leading upto the situation in Figure 7.3(b).

R4: $\neg B_{1,1}$

R5 : $B_{2,1}$

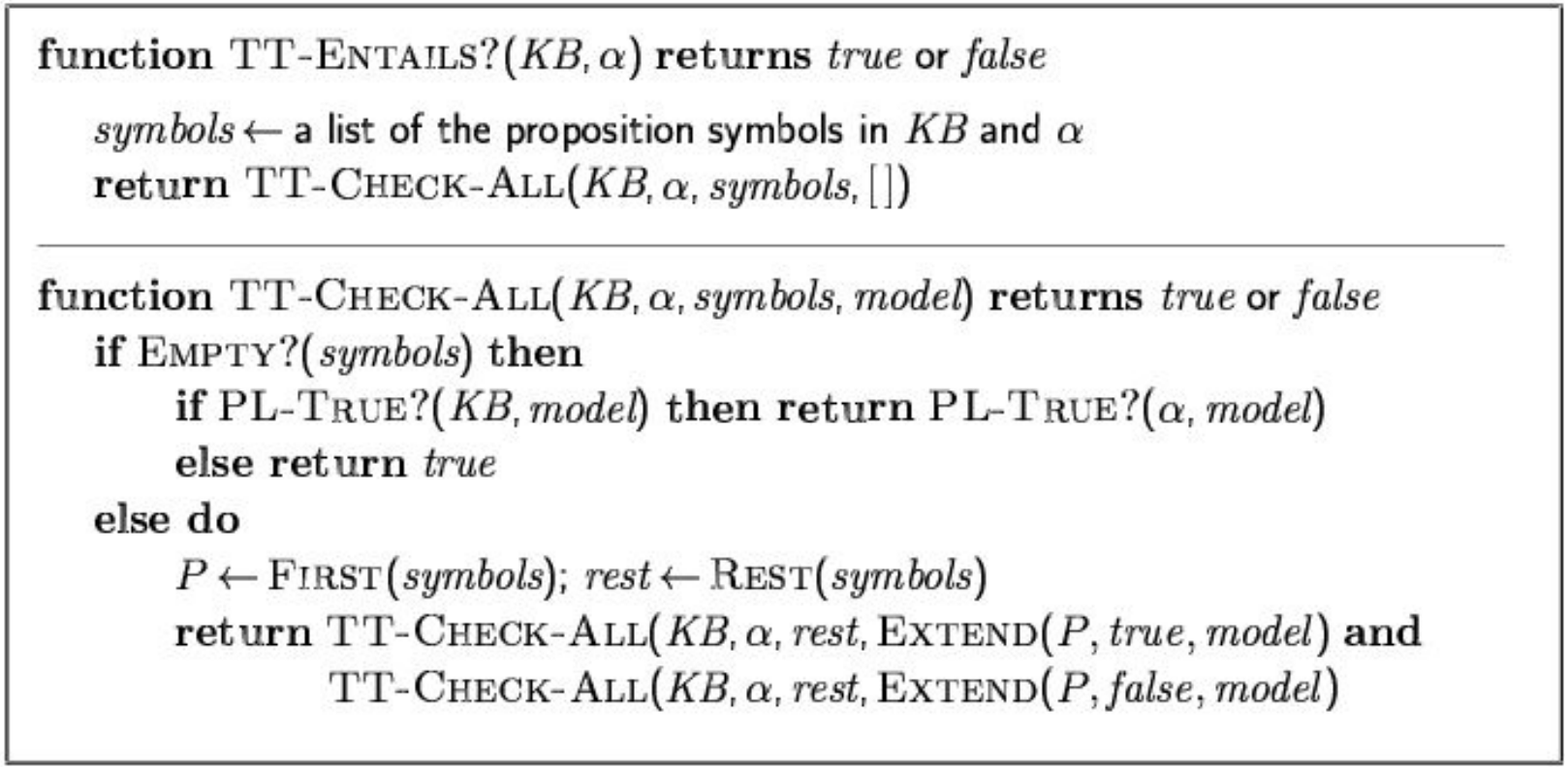
Inference

- goal now is to decide whether KB |= α for some sentence α. For example, is ¬P1,2entailed by our KB? Our first algorithm for inference is a model-checking approach that is a direct implementation of the definition of entailment: enumerate the models, and check thatα is true in every model in which KB is true. Models are assignments of true or false to every proposition symbol.
- $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$. With seven symbols, there are 2^7 =128 possible models; in three of these, KB is true (Figure 7.9).

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	α_1
false	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*false*	*false*	*false*	*false*	*false*	*true*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	*true*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

A truth table constructed for the knowledge base given in the text. *KB is true* if R1 through R5 are true, which occurs in just 3 of the 128 rows (the ones underlined in the right-hand column). In all 3 rows, $P_{1,2}$ is false, so there is no pit in [1,2]. On the other hand, there might (or might not) be a pit in [2,2].



```
function TT-ENTAILS?(KB, α) returns true or false
    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
    else do
        P ← FIRST(symbols); rest ← REST(symbols)
        return TT-CHECK-ALL(KB, α, rest, EXTEND(P, true, model)) and
                TT-CHECK-ALL(KB, α, rest, EXTEND(P, false, model))
```

Figure 7.10 A truth-table enumeration algorithm for deciding propositional entailment.(TT stands for truth table.) PL-TRUE? returns *true if a sentence holds within a model. The* variable *model represents a partial model—an assignment to some of the symbols. The keyword* "**and" is used here as a logical operation on its two arguments, returning true or false.**

- The algorithm is **sound because it** implements directly the definition of entailment, and **complete because it works for any KB** and α and always terminates—there are only finitely many models to examine.

PROPOSITIONAL THEOREM PROVING

- Entailment can be done by **theorem** proving—applying rules of inference directly to the sentences in knowledge base to construct a proof of the desired sentence without consulting models.
- If the number of models is large but the length of the proof is short, then theorem proving can be more efficient than model checking.

Logical equivalence

- **logical equivalence:** two sentences α and β are logically equivalent if they are true in the same set of models. Two sentences are logically equivalent iff true in same models:
- α ≡ ß iff α ⊨ β and β ⊨ α

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

Validity and satisfiability

A sentence is valid if it is true in all models,

- Valid sentences are also known as **tautologies.** the sentence True is true in all models, every valid sentence is logically equivalent to True
  e.g., *True*,A $\vee \neg$A, A $\Rightarrow$ A, (A $\wedge$ (A $\Rightarrow$ B)) $\Rightarrow$ B

Validity is connected to inference via the Deduction Theorem:
For any sentences α and β, α |= β if and only if the sentence (α ⇒ β) is valid.

- we can decide if α |= β by checking that (α ⇒ β) is true in every model—which is essentially what the inference algorithm in Figure 7.10 does—or by proving that (α ⇒ β) is equivalent to True. the deduction theorem states that every valid implication sentence describes a legitimate inference.
- A sentence is satisfiable if it is true in or satisfied by, some model
  e.g., A $\vee$ B, C

- $(R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5)$, is satisfiable because there are three models in which it is true,
- Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence. The problem of determining the satisfiability of sentences in propositional logic—the SAT problem

A sentence is unsatisfiable if it is true in no models

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

- α is valid iff ¬α is unsatisfiable; contrapositively, α is satisfiable iff ¬α is not valid.
- α |= β *if and only if the sentence (α ∧ ¬β) is unsatisfiable.*
- Proving β from α by checking the unsatisfiability of (α ∧ ¬β) corresponds exactly to the standard mathematical proof technique of *reductio ad absurdum* It is also called proof by **refutation or proof by contradiction.** One assumes a sentence β to be false and shows that this leads to a contradiction with known axioms α. This contradiction is exactly what is meant by saying that the sentence (α ∧ ¬β) is unsatisfiable.

Inference and proofs

- Proof methods divide into (roughly) two kinds:
- Application of inference rules : Legitimate (sound) generation of new sentences from old
  Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search algorithm
- **Modus Ponens**

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

- **And-Elimination**

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

- if (WumpusAhead $\wedge$ WumpusAlive) $\Rightarrow$ Shoot

and (WumpusAhead $\wedge$ WumpusAlive) are given, then Shoot can be inferred.

from (WumpusAhead $\wedge$ WumpusAlive), WumpusAlive can be inferred.

Modus Ponens and And-Elimination are sound

- All of the logical equivalences can be used as inference rules. For example, the equivalence for biconditional elimination yields the two inference rules

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \qquad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Prove ¬P1,2

biconditional elimination to R_2 to obtain

$$R_6 : \quad (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

Then we apply And-Elimination to R_6 to obtain

$$R_7 : \quad ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}) .$$

Logical equivalence for contrapositives gives

$$R_8 : \quad (\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})) .$$

Now we can apply Modus Ponens with R_8 and the percept R_4 (i.e., $\neg B_{1,1}$), to obtain

$$R_9 : \quad \neg(P_{1,2} \vee P_{2,1}) .$$

Finally, we apply De Morgan's rule, giving the conclusion

$$R_{10} : \quad \neg P_{1,2} \wedge \neg P_{2,1} .$$

That is, neither [1,2] nor [2,1] contains a pit.

One final property of logical systems is **monotonicity, which says that the set of entailed** sentences can only *increase as information is added to the knowledge base.8 For any* sentences α and β, if KB |= α then KB ∧ β |= α .

Proof

- INITIAL STATE: the initial knowledge base.
- ACTIONS: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.
- RESULT: the result of an action is to add the sentence in the bottom half of the inference rule.
- GOAL: the goal is a state that contains the sentence we are trying to prove.

Property of logical systems is**monotonicity,** which says that the set of entailed sentences can only *increase as information is added to the knowledge base.8 For any* sentences α and β, if KB |= α then KB ∧ β |= α .

- the agent returns from [2,1] to [1,1] and then goes to [1,2], where it perceives a stench, but no breeze.

$$R_{11}: \quad \neg B_{1,2} .$$
$$R_{12}: \quad B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3}) .$$
$$R_{13}: \quad \neg P_{2,2} .$$
$$R_{14}: \quad \neg P_{1,3} .$$

Apply biconditional elimination to R_3, followed by Modus Ponens with R_5, to obtain the fact that there is a pit in [1,1], [2,2], or [3,1]:

$$R_{15}: \quad P_{1,1} \vee P_{2,2} \vee P_{3,1} .$$

Resolution

- ¬$P_{2,2}$ in R_{13} *resolves with* the literal $P_{2,2}$ in R_{15} to give the **resolvent**

$$R_{16}: \quad P_{1,1} \vee P_{3,1} .$$

In English; if there's a pit in one of [1,1], [2,2], and [3,1] and it's not in [2,2], then it's in [1,1] or [3,1].

- Literal ¬$P_{1,1}$ in R_1 resolves with the literal $P_{1,1}$ in R_{16} to give

$$R_{17}: \quad P_{3,1} .$$

- In English: if there's a pit in [1,1] or [3,1] and it's not in [1,1], then it's in [3,1].

- **unit resolution inference rule,**

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k},$$

where each is a literal and li and m are **complementary literals** (i.e., one is the negation of the other). Thus, the unit resolution rule takes a **clause—a disjunction of literals—and a** literal and produces a new clause. Note that a single literal can be viewed as a disjunction of one literal, also known as a **unit clause.**

Resolution rule

- clause—a disjunction of literals
- single literal can be viewed as a disjunction of one literal, also known as a **unit clause.**

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n},$$

$$\frac{P_{1,1} \vee P_{3,1}, \qquad \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}.$$

The removal of multiple copies of literals is called **factoring.** Resolve (A ∨ B) with (A ∨ ¬B), we obtain (A ∨ A), which is reduced to just A.

Resolution is Sound

- The *soundness of the resolution rule can be seen easily by considering the literal li that* is complementary to literal mj in the other clause. If li is true, then mj is false, and hence $m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$ must be true, because $m_1 \vee \cdots \vee m_n$ is given. If l_i is false, then $l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k$ must be true because $l_1 \vee \cdots \vee l_k$ is given. Now li is either true or false, so one or other of these conclusions holds—exactly as the resolution rule states.

CNF **-Conjunctive normal form**

- *every sentence of propositional logic is logically equivalent to a conjunction of clauses. A* sentence expressed as a conjunction of clauses is said to be in **conjunctive normal form or**

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Apply distributive law ($\wedge$ over $\vee$) and flatten:

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution

- To show that KB |= α, show that (KB ∧ ¬α) is unsatisfiable. by proving a contradiction.
- First, (KB ∧ ¬α) is converted intoCNF. Then, the resolution rule is applied to the resulting clauses.
- Each pair that contains complementary literals is resolved to produce a new clause, which is added to the set if it is not already present.
- The process continues until one of two things happens:

there are no new clauses that can be added, in which case KB does not entail α; or two clauses resolve to yield the *empty clause, in which case KB entails α.*

- The empty clause—a disjunction of no disjuncts—is equivalent to False because a disjunction is true only if at least one of its disjuncts is true.

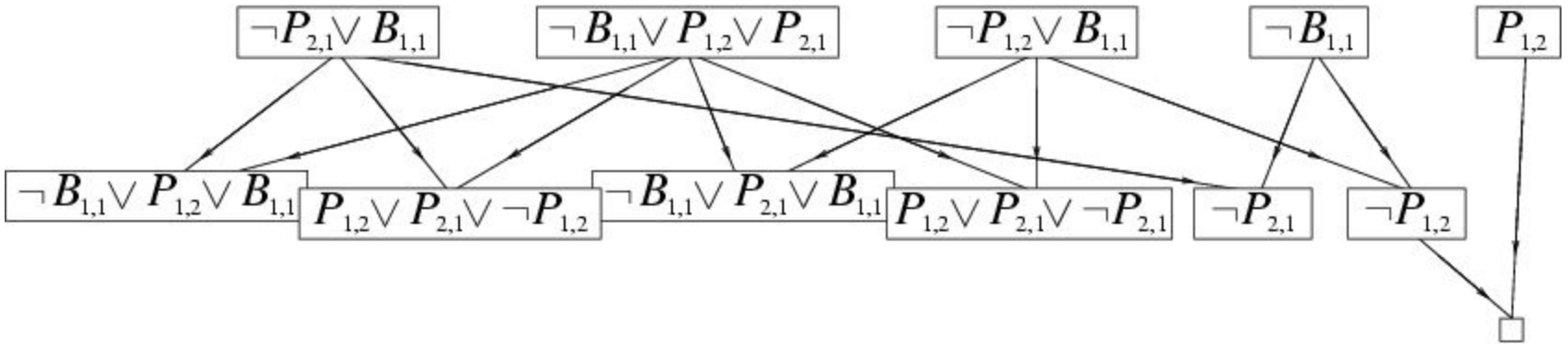
Resolution algorithm

- Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Resolution example proove $\alpha = \neg P_{1,2}$

- *KB* = $(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ $\alpha = \neg P_{1,2}$



Completeness of resolution

The **resolution closure** RC(S) of a set of clauses S, which is the set of all clauses derivable by repeated application of the resolution rule to clauses in S or their derivatives.

- The completeness theorem for resolution in propositional logic is called the **ground resolution theorem**: If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

- **DEFINITE CLAUSE is a clause, which is a disjunction of literals of** which *exactly one is positive. For example, the clause (¬L1,1 ∨ ¬Breeze ∨ B1,1) is a definite* clause, whereas (¬B1,1 ∨ P1,2 ∨ P2,1) is not.
- HORN CLAUSE **which is a disjunction of literals of which *at most one is positive. So all definite clauses are Horn clauses, as are clauses with no positive*** literals; these are called **goal clauses.** Horn clauses are closed under resolution: if you resolve two Horn clauses, you get back a Horn clause.

$$
\begin{aligned}
\mathit{CNFSentence} &\rightarrow \mathit{Clause}_1 \wedge \cdots \wedge \mathit{Clause}_n \\
\mathit{Clause} &\rightarrow \mathit{Literal}_1 \vee \cdots \vee \mathit{Literal}_m \\
\mathit{Literal} &\rightarrow \mathit{Symbol} \mid \neg \mathit{Symbol} \\
\mathit{Symbol} &\rightarrow P \mid Q \mid R \mid \ldots \\
\mathit{HornClauseForm} &\rightarrow \mathit{DefiniteClauseForm} \mid \mathit{GoalClauseForm} \\
\mathit{DefiniteClauseForm} &\rightarrow (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{Symbol} \\
\mathit{GoalClauseForm} &\rightarrow (\mathit{Symbol}_1 \wedge \cdots \wedge \mathit{Symbol}_l) \Rightarrow \mathit{False}
\end{aligned}
$$

Figure 7.14 A grammar for conjunctive normal form, Horn clauses, and definite clauses.

Forward and backward chaining

- Horn Form (restricted)

  KB = conjunction of Horn clauses

  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) ⇒ symbol
  - E.g., C ∧ (B ⇒ A) ∧ (C ∧ D ⇒ B)

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

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

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

Converting Arbitrary wffs to Conjunctions of Clauses

- Any wff in propositional calculus can be converted to an equivalent CNF.
- Ex: $\neg(P \supset Q) \vee (R \supset P)$

1. $\neg(\neg P \vee Q) \vee (\neg R \vee P)$ Equivalent Form Using $\vee$
2. $(P \wedge \neg Q) \vee (\neg R \vee P)$ DeMorgan
3. $(P \vee \neg Q \vee P) \wedge (\neg Q \vee \neg R \vee P)$ Distributive Rule
4. $(P \vee \neg R) \wedge (\neg Q \vee \neg R \vee P)$ Associative Rule

- Usually expressed as $\{(P \vee \neg R), (\neg Q \vee \neg R \vee P)\}$

Example

- Given a set, Δ, of wffs: {P, R, P ⊃ Q}, {P, P ⊃ Q, Q, R, Q ∧ R} is a proof of Q ∧ R.
- The concept of proof can be based on a partial order.

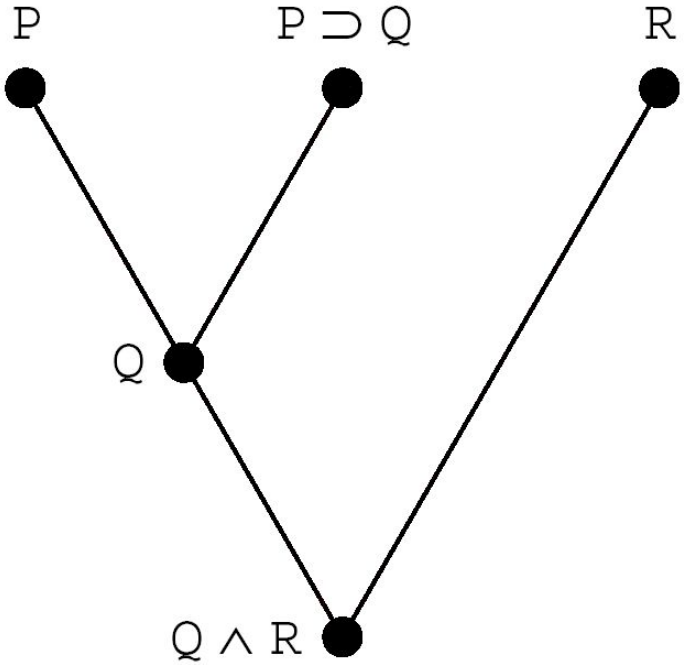


Figure 13.1 A Sample Proof Tree

A Resolution Refutation Tree

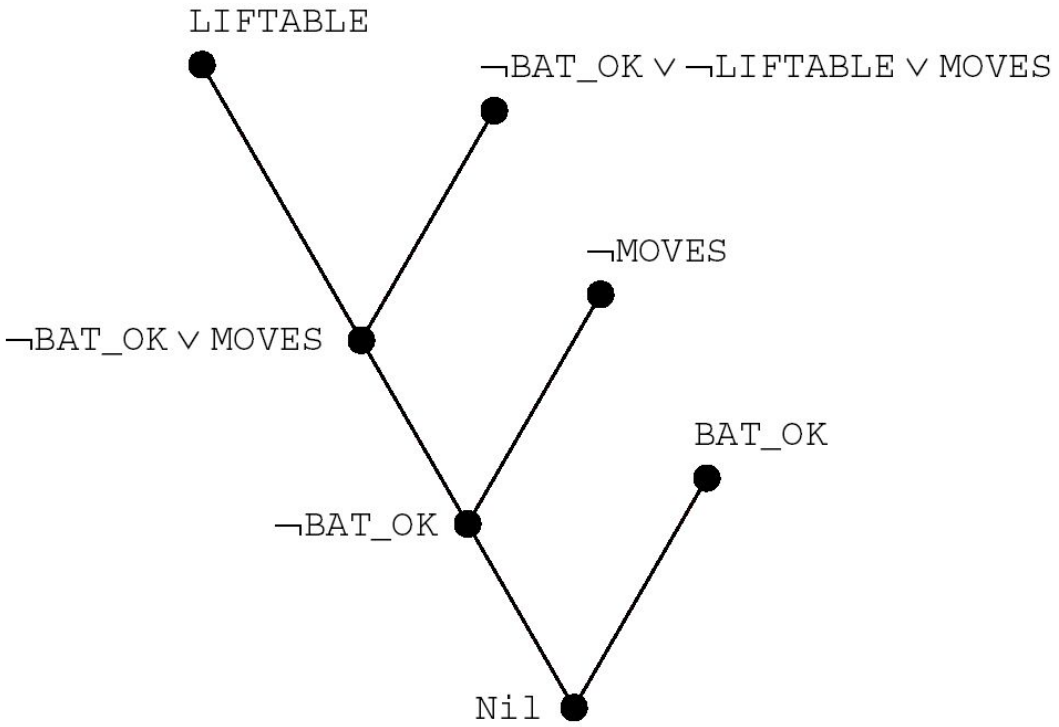


Figure 14.1 A Resolution Refutation Tree

Given:

1. BAT_OK
2. ¬MOVES
3. BAT_OK ∧ LIFTABLE ⊃ MOVES

Clause form of 3:

4. ¬BAT_OK ∨ ¬LIFTABLE ∨ MOVES

Negation of goal:

5. LIFTABLE

Perform resolution:

6. ¬BAT_OK ∨ MOVES (from resolving 5 with 4)
7. ¬BAT_OK (from 6, 2)
8. Nil (from 7, 1)

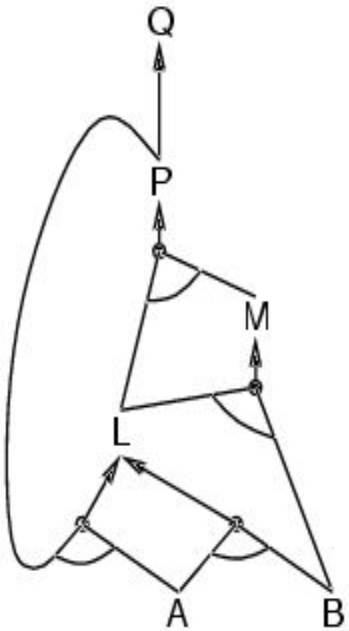
forward-chaining

- The forward-chaining algorithm PL-FC-ENTAILS?(KB, q) determines if a single proposition symbol q—the query—is entailed by a knowledge base of definite clauses. It begins from known facts (positive literals) in the knowledge base. If all the premises of an implication are known, then its conclusion is added to the set of known facts. For example, if L1,1 and Breeze are known and (L1,1 ∧ Breeze) ⇒ B1,1 is in the knowledge base, then B1,1 can be added. This process continues until the query q is added or until no further inferences can be made.

Forward chaining

- Idea: fire any rule whose premises are satisfied in the *KB*,
  - add its conclusion to the *KB*, until query is found

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



a simple knowledge base of Horn clauses with A and B as known facts. In AND–OR graphs, multiple links joined by an arc indicate a conjunction—every link must be proved—while multiple links without an arc indicate a disjunction—any link can be proved. The known leaves (here, A and B) are set, and inference propagates up the graph as far as possible. Wherever a conjunction appears, the propagation waits until all the conjuncts are known before proceeding.

Forward chaining algorithm

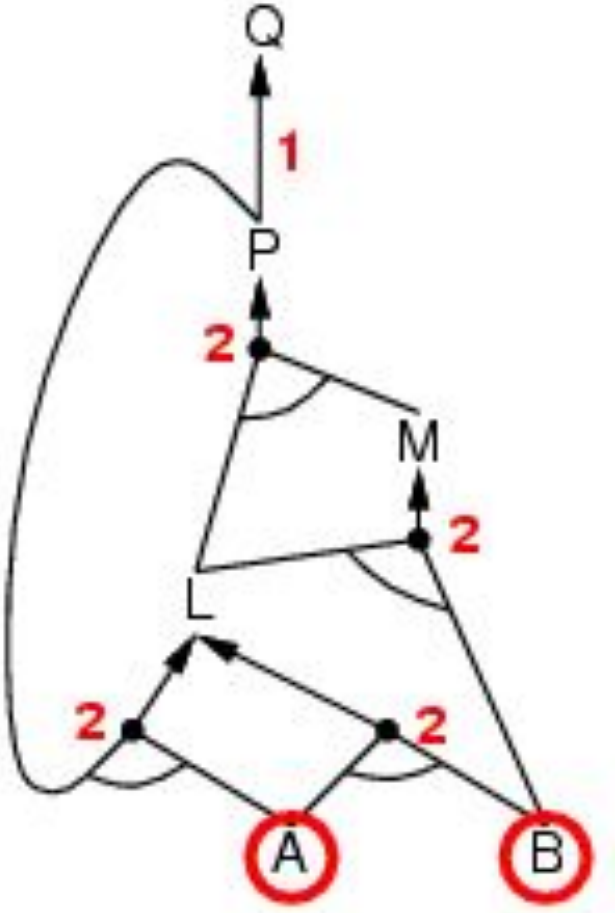
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  local variables: count, a table, indexed by clause, initially the number of premises
                   inferred, a table, indexed by symbol, each entry initially false
                   agenda, a list of symbols, initially the symbols known 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
```

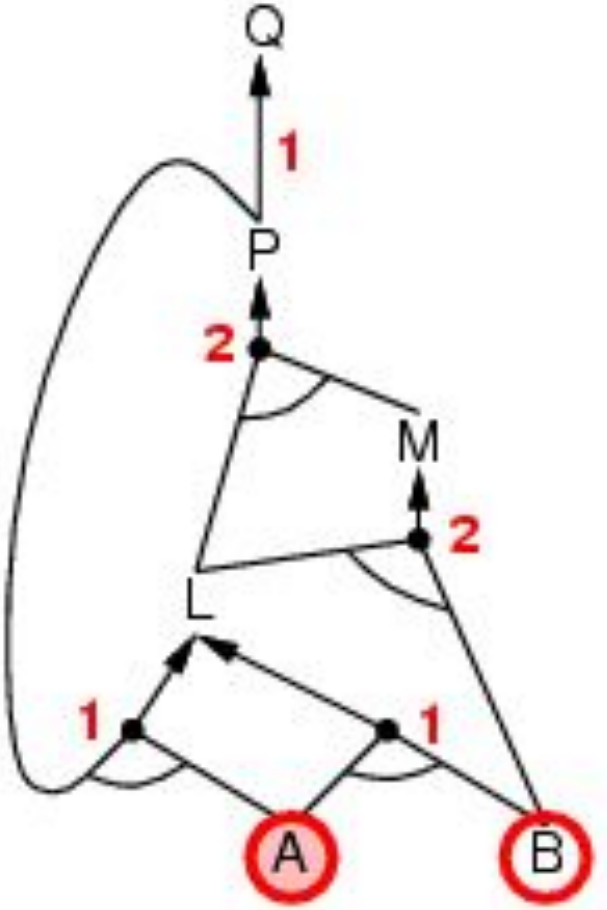
- Forward chaining is sound and complete for Horn KB

The forward-chaining algorithm for propositional logic. The agenda keeps track of symbols known to be true but not yet "processed." The count table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as P ⇒ Q and Q ⇒ P.

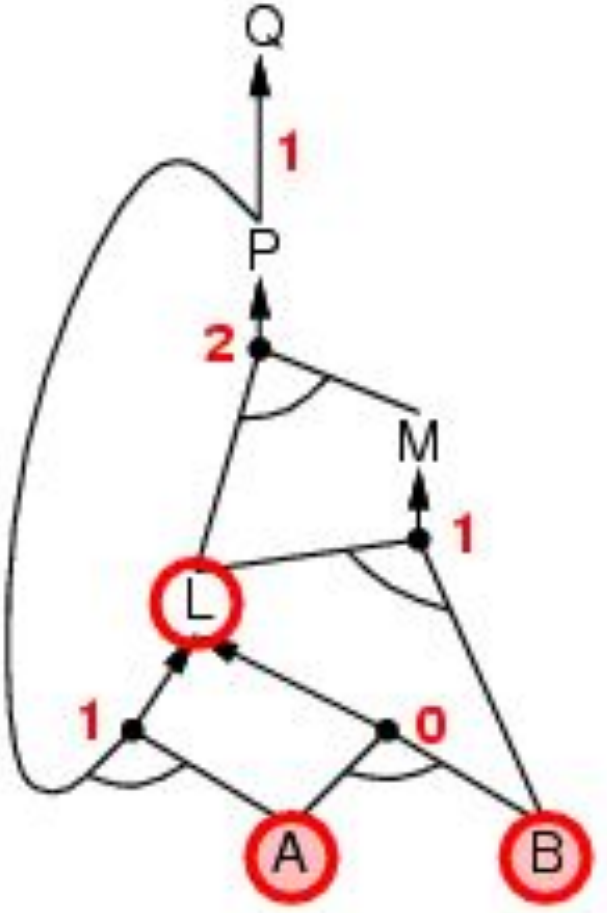
Forward chaining example



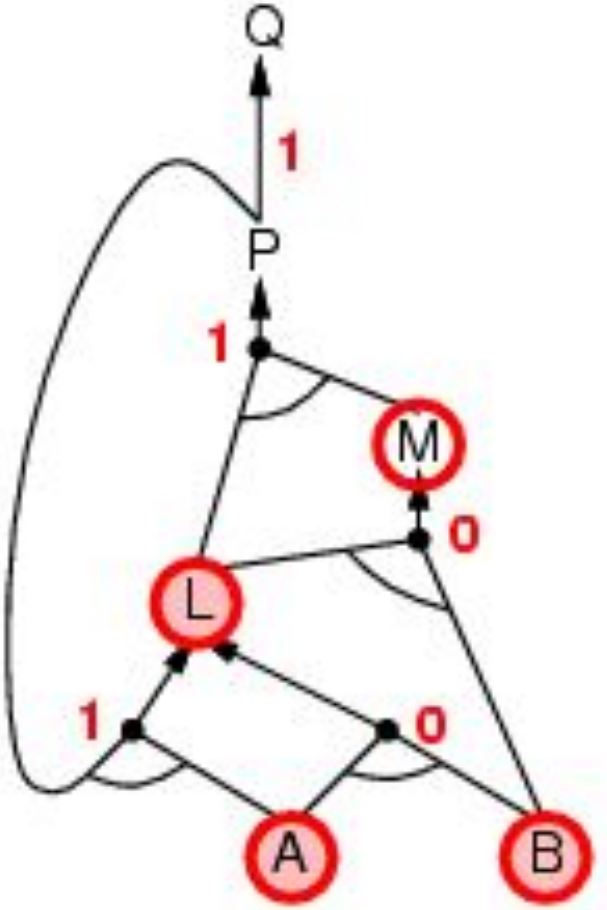
Forward chaining example



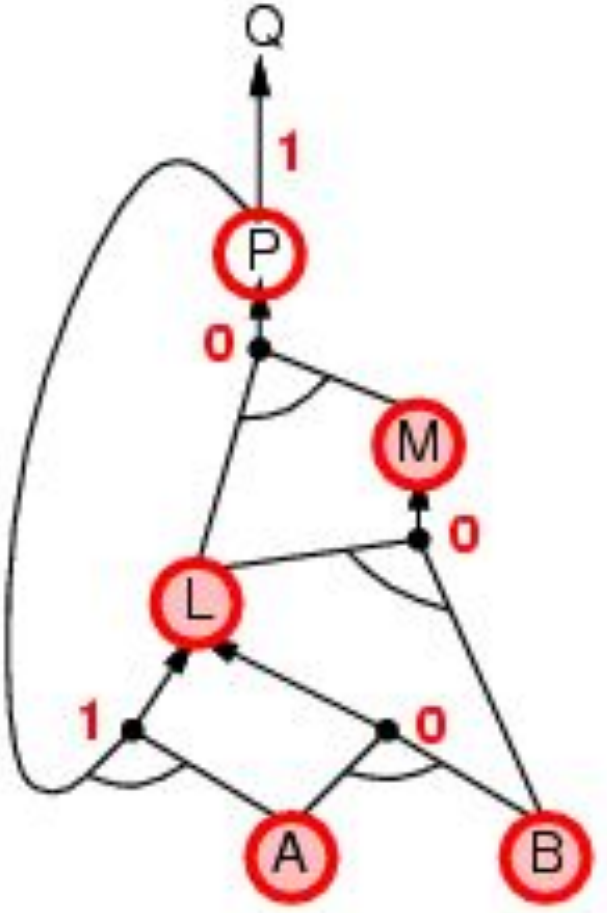
Forward chaining example



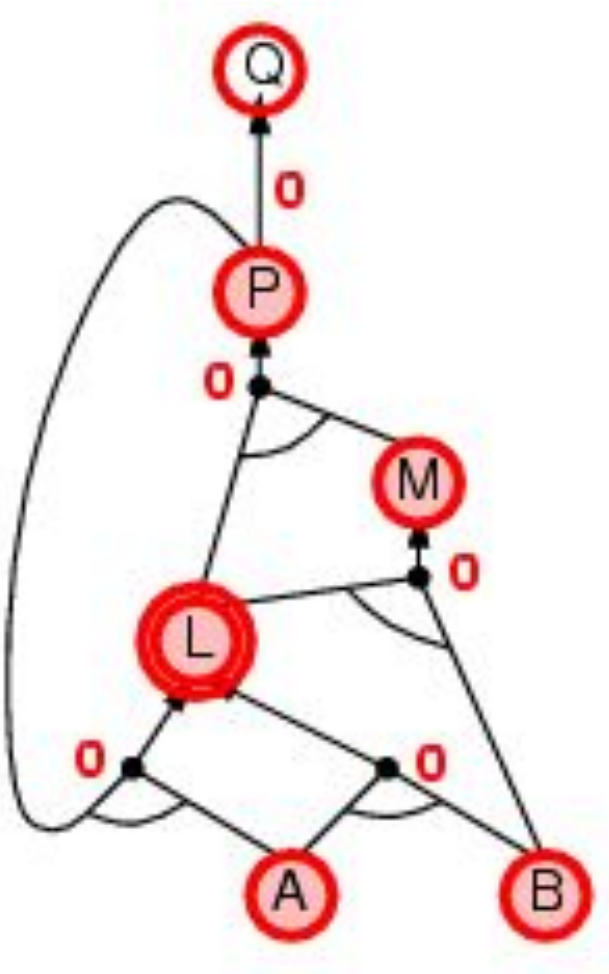
Forward chaining example



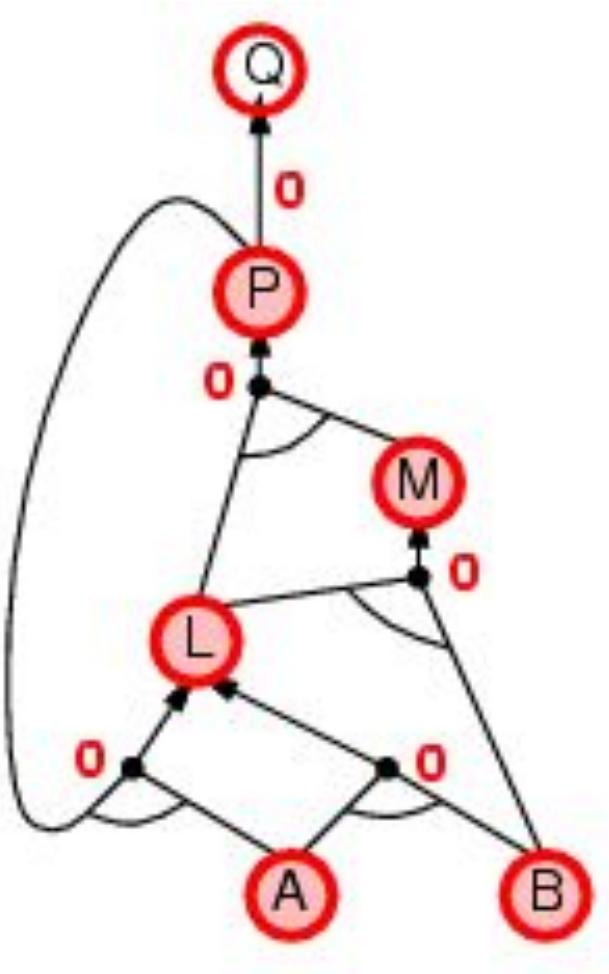
Forward chaining example



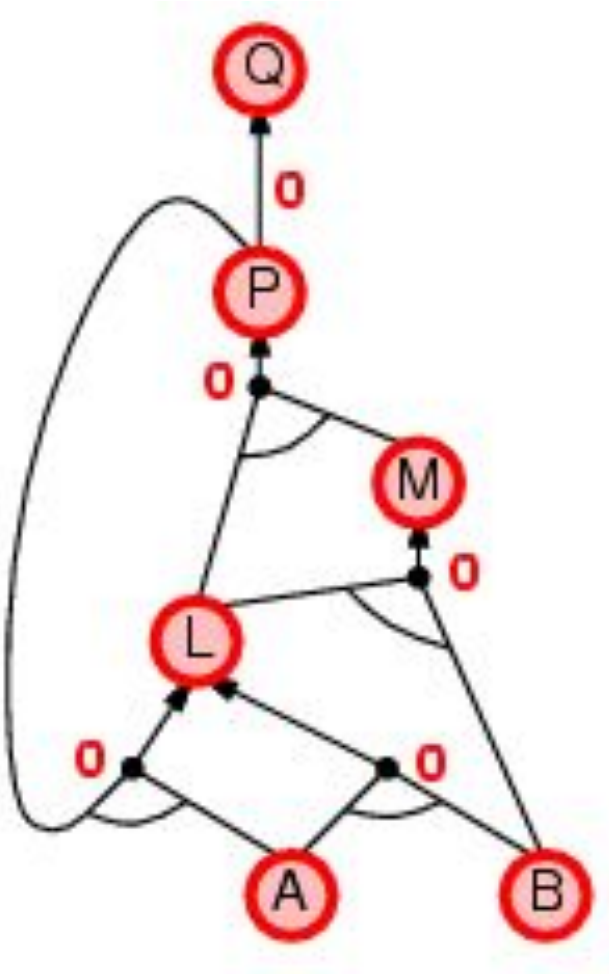
Forward chaining example



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
   Consider the final state as a model *m*, assigning true/false to symbols
   Every clause in the original *KB* is true in *m*
   $a_1 \wedge \ldots \wedge a_k \Rightarrow b$
   Hence *m* is a model of *KB*
   If $KB \models q$, *q* is true in every model of *KB*, including *m*

Forward chaining is an example of the general concept of **data-driven** reasoning—that is, reasoning in which the focus of attention starts with the known data. It can be used within an agent to derive conclusions from incoming percepts, often without a specific query in mind. For example, the wumpus agent might TELL its percepts to the knowledge base using an incremental forward-chaining algorithm in which new facts can be added to the agenda to initiate new inferences. In humans, a certain amount of data-driven reasoning occurs as new information arrives. For example, if I am indoors and hear rain starting to fall, it might occur to me that the picnic will be canceled. Yet it will probably not occur to me that the seventeenth petal on the largest rose in my neighbor's garden will get wet; humans keep forward chaining under careful control, lest they be swamped with irrelevant consequences.

Backward chaining

Idea: work backwards from the query *q*:

to prove *q* by BC,

check if *q* is known already, or

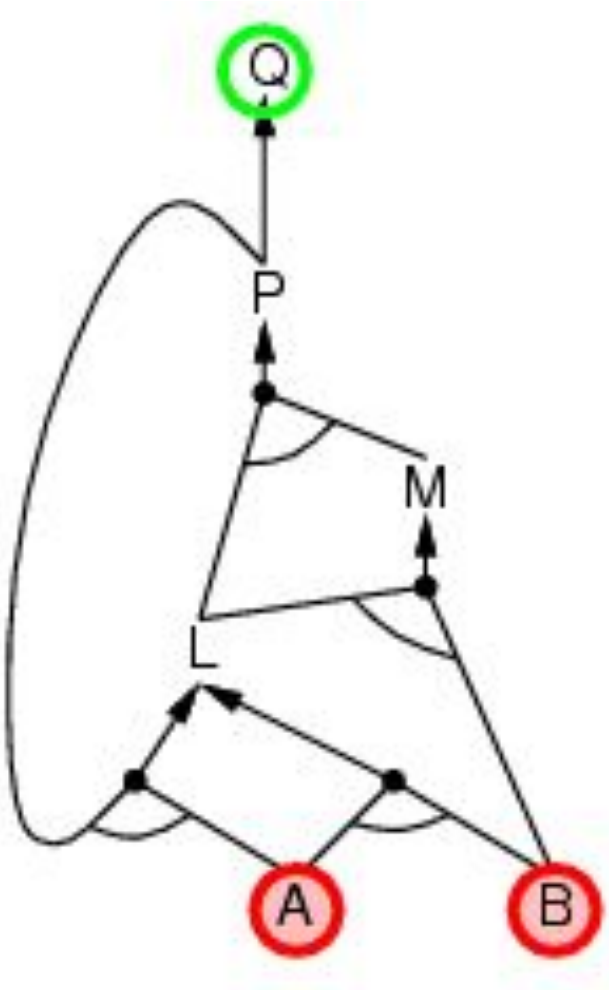
prove by BC all premises of some rule concluding *q*

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

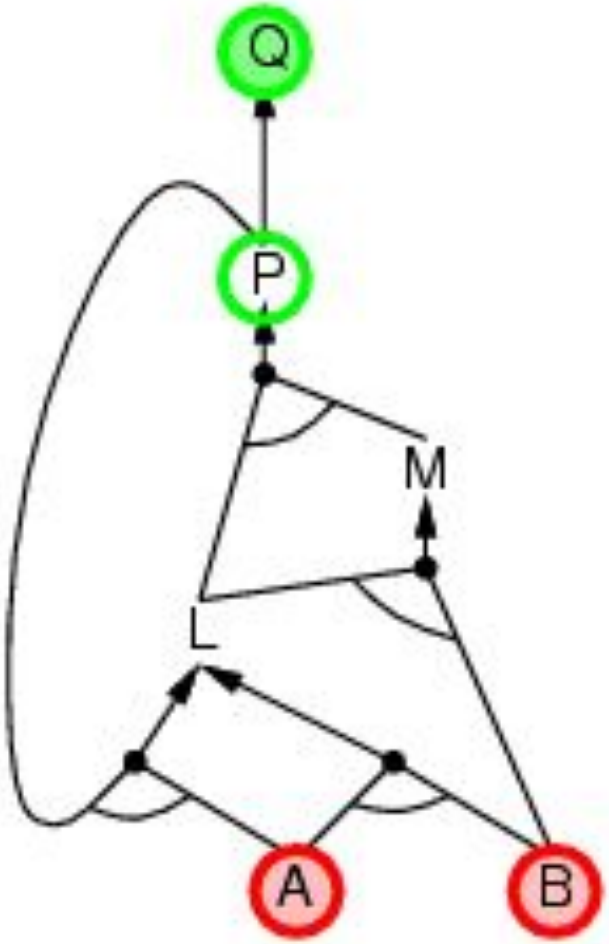
Avoid repeated work: check if new subgoal has already been proved true, or has already failed

The backward-chaining algorithm, as its name suggests, works backward from the query. If the query q is known to be true, then no work is needed. Otherwise, the algorithm finds those implications in the knowledge base whose conclusion is q. If all the premises of one of those implications can be proved true (by backward chaining), then q is true.

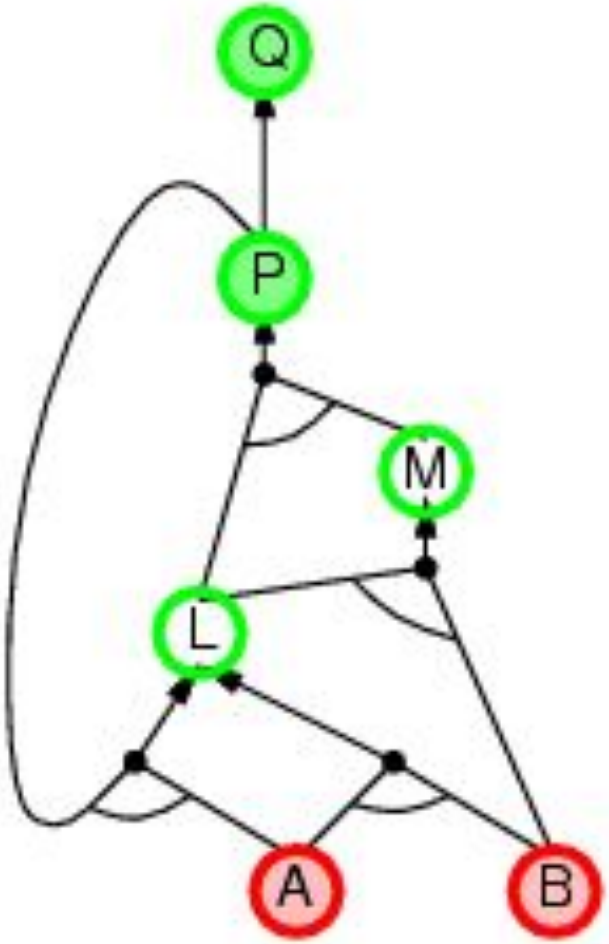
Backward chaining example



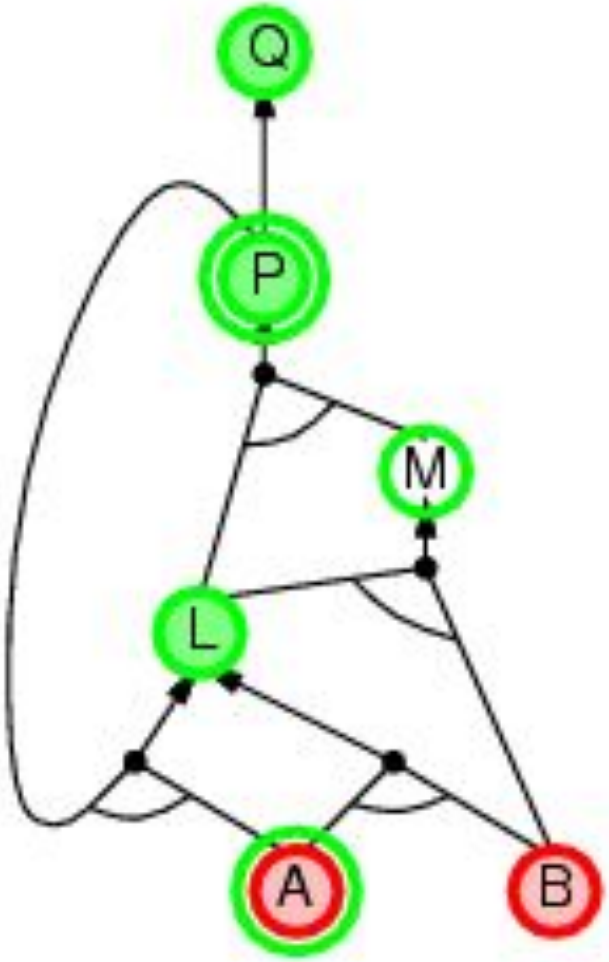
Backward chaining example



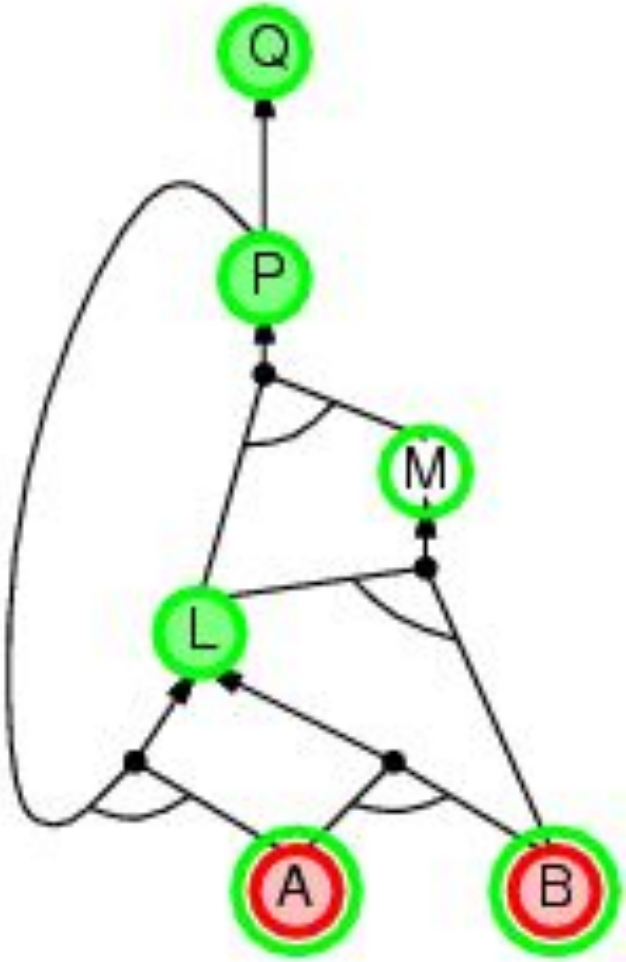
Backward chaining example



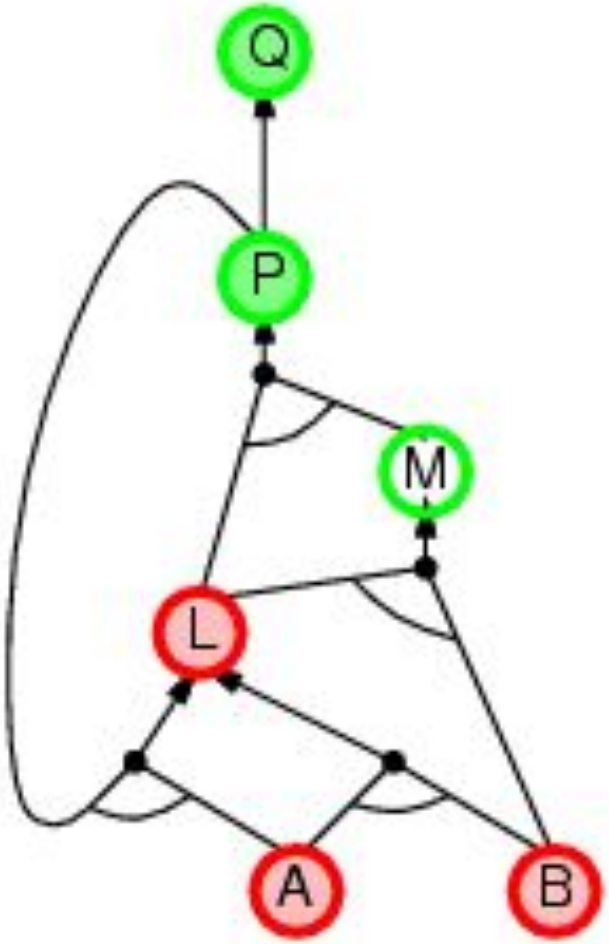
Backward chaining example



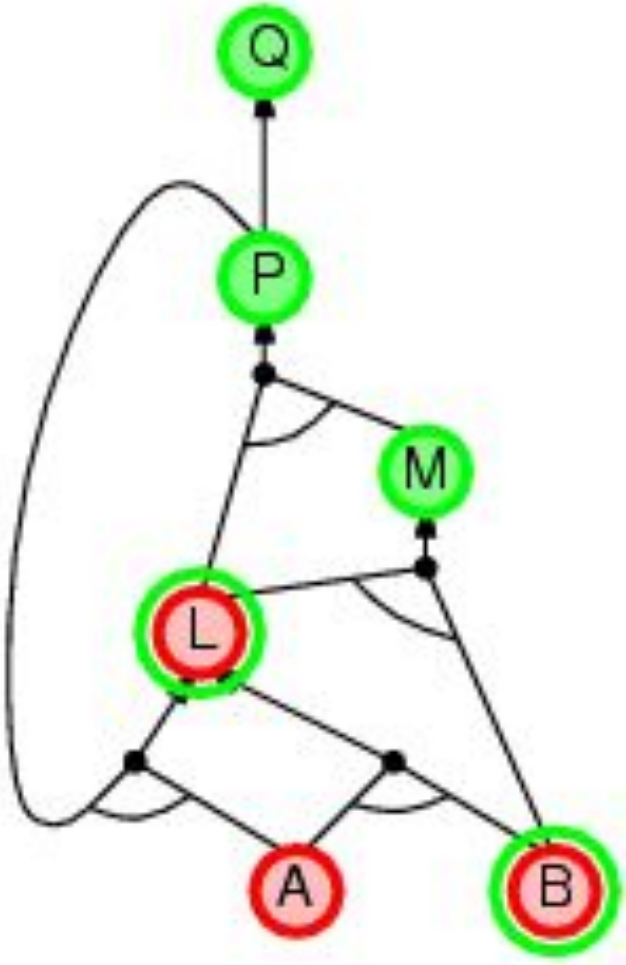
Backward chaining example



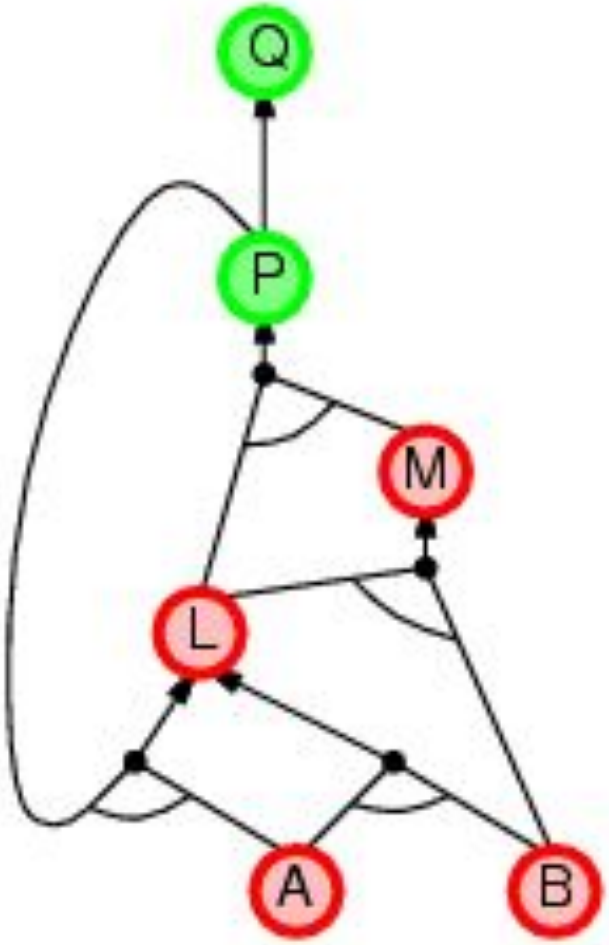
Backward chaining example



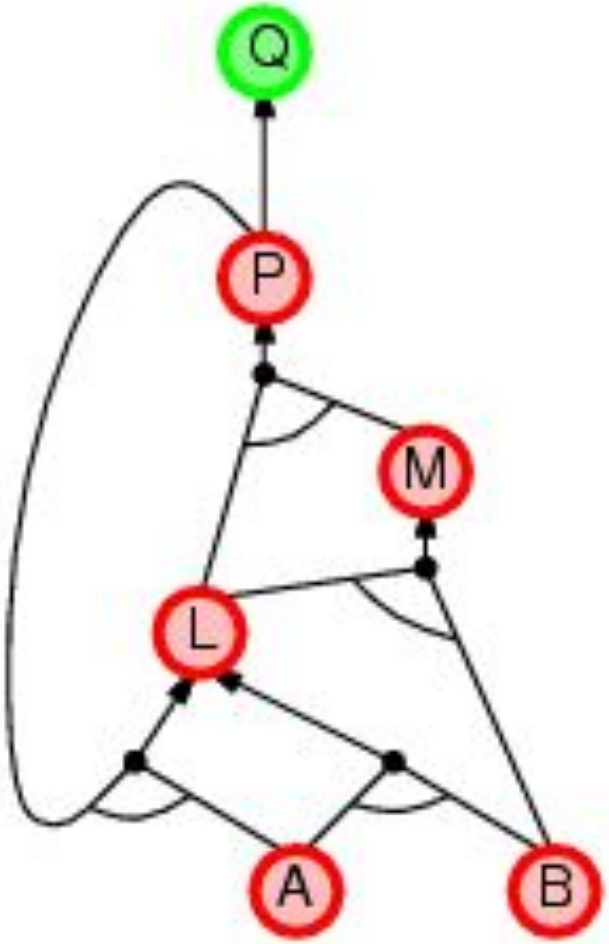
Backward chaining example



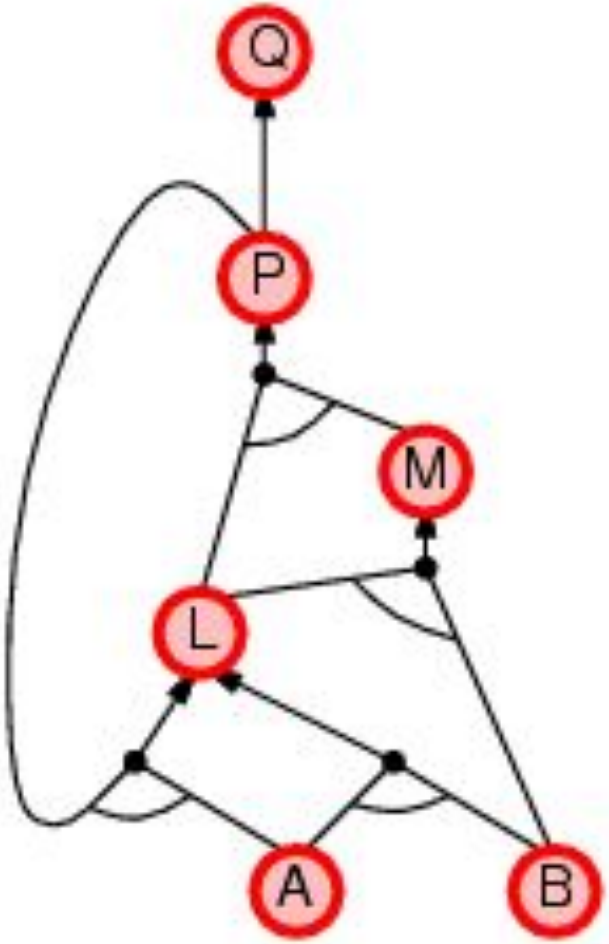
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

- FC is data-driven, 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

(Smoke ⇒ Fire) ⇒ (¬Smoke ⇒ ¬Fire) is valid

smoke	$\sim smoke$	fire	$\sim fire$	$smoke \Rightarrow fire$	$\sim smoke \Rightarrow \sim fire$	$smoke \Rightarrow fire = \sim smoke \Rightarrow \sim fire$
T	F	T	F	T	T	T
T	F	F	T	F	F	T
F	T	T	F	T	T	T
F	T	F	T	T	T	T

((Smoke ∧ Heat) ⇒ Fire) ⇔ ((Smoke ⇒ Fire) ∨ (Heat ⇒ Fire)) is valid

Smoke	Heat	$(Smoke \wedge Heat)$	Fire	$((Smoke \wedge Heat) \Rightarrow Fire)$	$Smoke \Rightarrow Fire$	$Heat \Rightarrow Fire$	$\sim (Heat \Rightarrow Fire)$	$((Smoke \wedge Heat) \Rightarrow Fire) \Leftrightarrow ((Smoke \Rightarrow Fire) \vee (Heat \Rightarrow Fire))$
F	F	F	F	T	T	T	F	T
F	T	F	F	T	T	T	F	T
T	F	F	T	T	F	F	T	T
T	T	T	T	T	T	T	F	T

According to some political pundits, a person who is radical (R) is electable (E) if
he/she is conservative (C), but otherwise is not electable.

a. Which of the following are correct representations of this assertion?

(i) $(R \wedge E) \Leftarrow\Rightarrow C$

(**ii) $R \Rightarrow (E \Leftarrow\Rightarrow C)$**

(iii) $R \Rightarrow ((C \Rightarrow E) \vee \neg E)$

7.17 A propositional *2-CNF* expression is a conjunction of clauses, each containing *exactly* 2 literals, e.g.,

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee D) \wedge (\neg C \vee G) \wedge (\neg D \vee G) .$$

a. Prove using resolution that the above sentence entails G.

- (A ∨ B) with (¬ A ∨ C) to get (BVC)
- Step 2. Now combine (B ∨ C) with (¬B ∨ D) to get (D ∨ C)
- (D ∨ C) with (¬C ∨ G) TO GET (DVG)
- (DVG) WITH (¬DVG) TO GET (GVG) WHICH IS G