AI

Artificial Intelligence

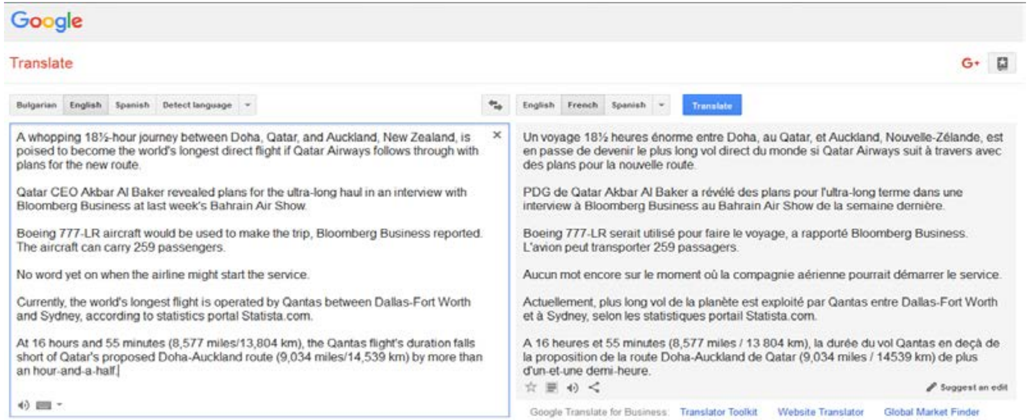
Language and Logic

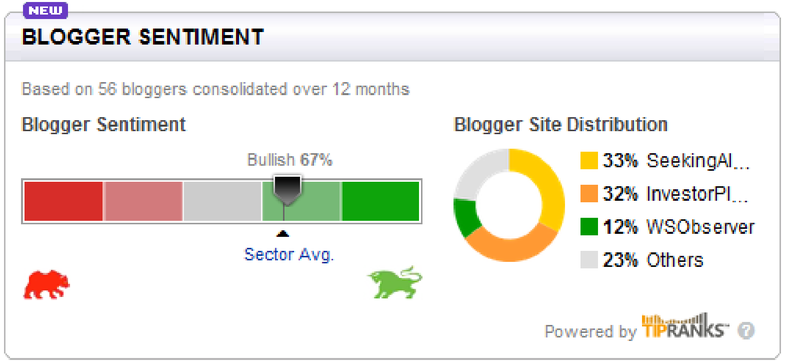
Natural Language Text is Everywhere

- Today's person is subjected to more information in a day than a person in the middle ages in a lifetime
- The search engine market is $94B a year
  - Feb 2016, New York Times
- Siri Gets 1 Billion requests a week
  - Jan 2016, USA Today, citing Apple
- Users send out 168 Million emails every minute
  - 2015, go-globe.com
- Google indexes at least 48 Billion web pages
  - 2016, WorldWideWebSize.com
- Twitter posts 400 Million tweets per day
  - 2012, Dick Costolo, CEO of Twitter
- Google performs 1 Billion automatic translations per day
  - 2016, Cnet.com

NLP Systems


THINK
DENKE
PENSER
PIENSE
$24,000
$77,147
$21,600
Who is Bram Stoker?
$ 17,973


Google
Translate
Bulgarian English Spanish Detect language
English French Spanish
Translate
A whopping 18½-hour journey between Doha, Qatar, and Auckland, New Zealand, is poised to become the world's longest direct flight if Qatar Airways follows through with plans for the new route.
Qatar CEO Akbar Al Baker revealed plans for the ultra-long haul in an interview with Bloomberg Business at last week's Bahrain Air Show.
Boeing 777-LR aircraft would be used to make the trip, Bloomberg Business reported. The aircraft can carry 259 passengers.
No word yet on when the airline might start the service.
Currently, the world's longest flight is operated by Qantas between Dallas-Fort Worth and Sydney, according to statistics portal Statista.com.
At 16 hours and 55 minutes (8,577 miles/13,804 km), the Qantas flight's duration falls short of Qatar's proposed Doha-Auckland route (9,034 miles/14,539 km) by more than an hour-and-a-half.
Un voyage 18½ heures énorme entre Doha, au Qatar, et Auckland, Nouvelle-Zélande, est en passe de devenir le plus long vol direct du monde si Qatar Airways suit à travers avec des plans pour la nouvelle route.
PDG de Qatar Akbar Al Baker a révélé des plans pour l'ultra-long terme dans une interview à Bloomberg Business au Bahrain Air Show de la semaine dernière.
Boeing 777-LR serait utilisé pour faire le voyage, a rapporté Bloomberg Business. L'avion peut transporter 259 passagers.
Aucun mot encore sur le moment où la compagnie aérienne pourrait démarrer le service.
Actuellement, plus long vol de la planète est exploité par Qantas entre Dallas-Fort Worth et à Sydney, selon les statistiques portail Statista.com.
A 16 heures et 55 minutes (8,577 miles / 13 804 km), la durée du vol Qantas en deçà de la proposition de la route Doha-Auckland de Qatar (9,034 miles / 14539 km) de plus d'un-et-une demi-heure.
Suggest an edit
Google Translate for Business: Translator Toolkit Website Translator Global Market Finder

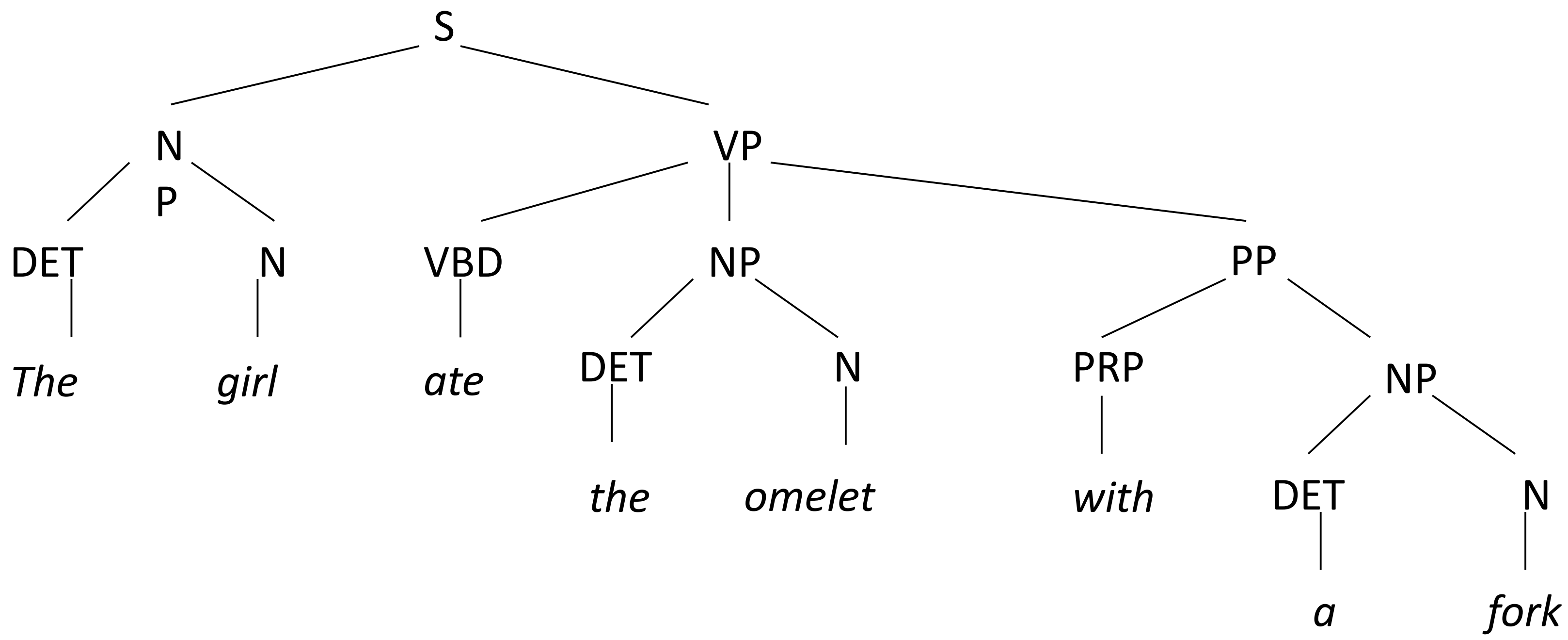

NEW
BLOGGER SENTIMENT
Based on 56 bloggers consolidated over 12 months
Blogger Sentiment
Bullish 67%
Sector Avg.
Blogger Site Distribution
33% SeekingAl...
32% InvestorPl...
12% WSObserver
23% Others
Powered by TIPRANKS

The NLP Pipeline

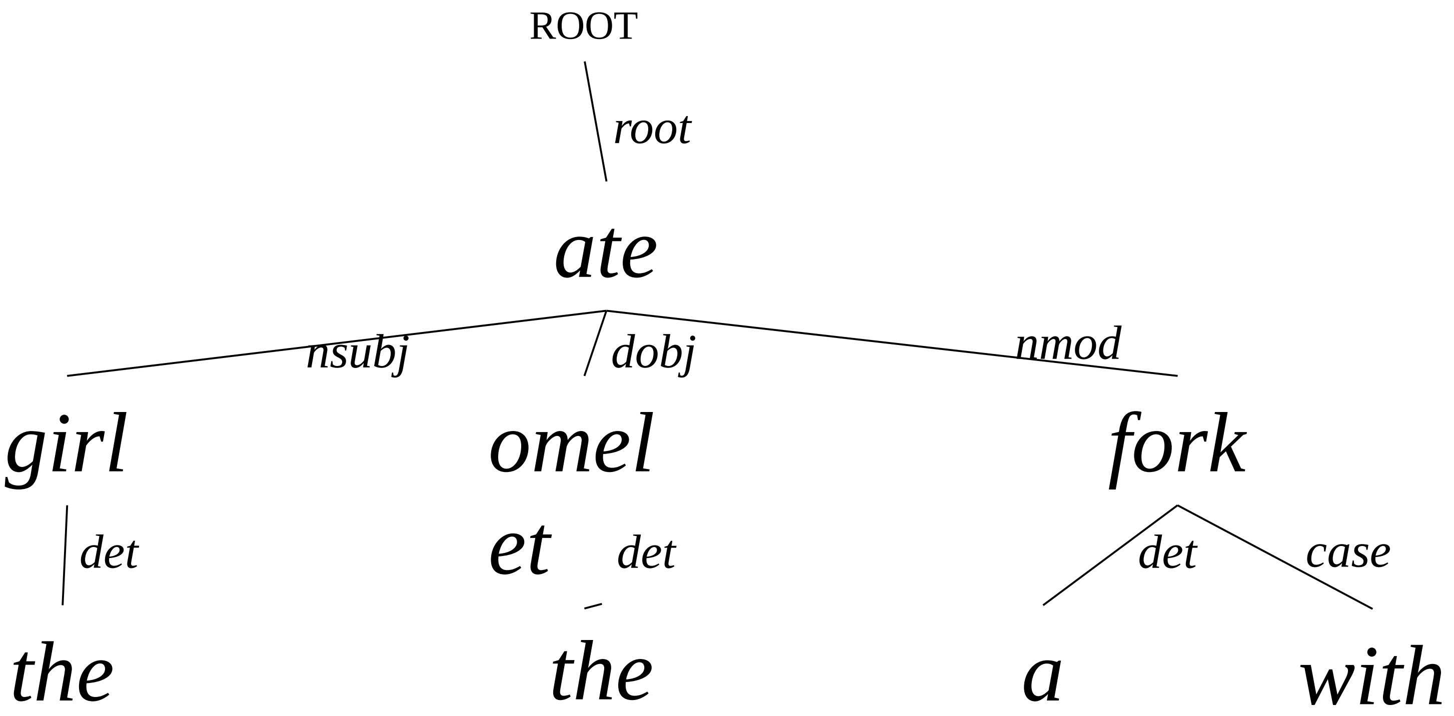
DET N VBD DET N PRP DET N

The girl ate the omelet with a fork.

Constituent Parsing



Dependency Parsing



Language Understanding

- Semantic Analysis

    $Girl\ (g_1)$

    $Omelet\ (o_1)$

    $Fork\ (f_1)$

    $Eating\ (e_1) \wedge Eater\ (e_1,g_1) \wedge Eaten\ (e_1,o_1) \wedge Instrument\ (e_1,f_1)$

- World Knowledge

    $Omelet\ (X) \Rightarrow Food\ (X)$

- Inference

    $Hungry\ (Z,t_0) \wedge Eater\ (e_1,Z) \wedge Eaten\ (e_1,Y) \wedge Time\ (e_1,t_1) \wedge Food\ (Y) \wedge Precedes\ (t_0,t_1) \Rightarrow \neg Hungry\ (Z,t_1)$

- Conclusion

    $\neg Hungry\ (g_1,t_1)$

Modern Methods for NLP

- Vector Semantics
  - Dimensionality Reduction
  - Compositionality
- Supervised Learning
  - Deep Neural Networks
- Learning Architectures
  - RNN, LSTM, CNN
  - Attention-based Models
  - Generative Adversarial Networks
  - Reinforcement Learning
  - Off the shelf libraries

Artificial Intelligence

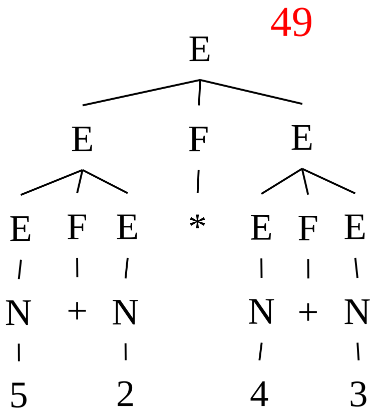
Semantics (from 3.6.1)

Syntax vs. Semantics

- Paraphrases
  - John broke the window
  - The window was broken by John
  - The breaking of the window by John
- Python expression
  - 5/2 = ?

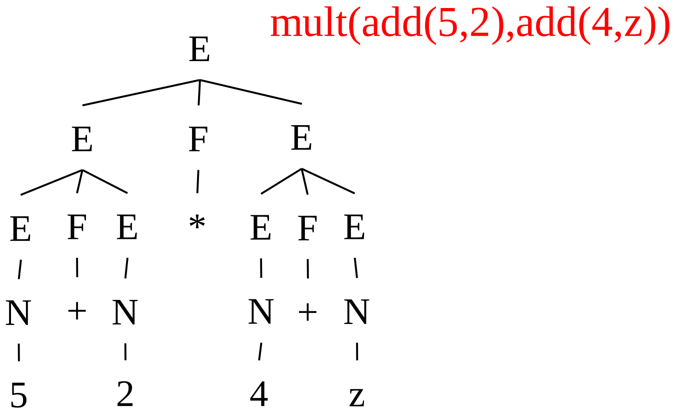
Semantics

- What is the meaning of: (5+2)*(4+3)?
- Parse tree



Semantics

- What if we had (5+2)*(4+z)?



What about (English) sentences?

- Every human is mortal.
- ??

Representing Meaning

- Goal
  - Capturing the meaning of linguistic utterances using formal notation
- Linguistic meaning
  - "It is 8 pm"
- Pragmatic meaning
  - "It is time to leave"
- Semantic analysis:
  - Assign each word a meaning
  - Combine the meanings of words into sentences
- *I bought a book*:

  $\exists\, x,y$: Buying(x) ^ Buyer($speaker,x$) ^ BoughtItem(y,x) ^ Book(y)

  Buying (Buyer=speaker, BoughtItem=book)

Understanding Meaning

- If an agent hears a sentence and can act accordingly, the agent is said to understand it
- Example
  - Leave the book on the table
- Understanding may involve inference
  - Maybe the book is wrapped in paper?
- And pragmatics
  - Which book? Which table?
- So, understanding may involve a procedure

Artificial Intelligence

8.3.2
Propositional Logic
(Chapter 7: part 2)

Models and Formulas

- Variables
  - E.g., A, B
- Models
  - assignment of truth values
- Formulas
  - A->B matches three possible models: (0,0), (0,1), (1,1)
- Knowledge base
  - Tell: add to the knowledge base (e.g., A=1)
  - Ask: query the knowledge base (e.g., A=?)

Clauses

- Definite clauses

  $p \wedge q \wedge r \rightarrow s$ (implication form)

  $\neg p \vee \neg q \vee \neg r \vee s$ (disjunctive form)

- Horn clauses
  - Either definite clauses
  - Or “goal clauses”:

    $p \wedge q \wedge r \rightarrow \mathit{false}$

    $\neg p \vee \neg q \vee \neg r \vee \neg s$

Propositional logic: Syntax

- Propositional logic is the simplest logic – illustrates basic ideas
- The proposition symbols P_1, P_2 etc are sentences
  - If S is a sentence, $\neg S$ is a sentence (negation)
  - If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
  - If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
  - If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
  - If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Rightarrow\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \ \Leftrightarrow\ \mathit{Sentence} \\
 & & \\
\text{OPERATOR PRECEDENCE} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Translating propositions to English

- A = Today is a holiday.
- B = We are going to the zoo.

- $B \Rightarrow A$
- $A \wedge \neg B$
- $\neg A \Rightarrow B$
- $\neg B \Rightarrow A$
- $B \Rightarrow A$

Translating propositions to English

A = Today is a holiday.
B = We are going to the zoo.

$B \Rightarrow A$

If we are going to the zoo, then today is a holiday.

$A \wedge \neg B$

Today is a holiday and we are not going to the zoo.

$\neg A \Rightarrow \neg B$

If today is not a holiday, then we are not going to the zoo.

$\neg B \Rightarrow \neg A$

If we are not going to the zoo, then today is not a holiday.

$B \Rightarrow A$

If we are going to the zoo, then today is a holiday.

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

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

With these symbols, 8 possible models, can be enumerated automatically.
Rules for evaluating truth with respect to a model *m*:

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

Simple recursive process evaluates an arbitrary sentence, e.g.,

$$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$$

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*

Wumpus world sentences

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

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

"Pits cause breezes in adjacent squares"

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{3,1})$$

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

$P_{x,y}$ 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 $[x, y]$.

The sentences we write will suffice to derive $\neg P_{1,2}$ (there is no pit in [1,2]), as was done informally in Section 7.3. We label each sentence R_i so that we can refer to them:

- There is no pit in [1,1]:

$$R_1 : \quad \neg P_{1,1} .$$

- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:

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

- The preceding sentences are true in all wumpus worlds. Now we include the breeze percepts for the first two squares visited in the specific world the agent is in, leading up to the situation in Figure 7.3(b).

$$R_4 : \quad \neg B_{1,1} .$$
$$R_5 : \quad B_{2,1} .$$

Truth tables for inference

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Inference by enumeration

- Depth-first enumeration of all models is sound and complete

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

   symbols ← a list of the proposition symbols in KB and α
   return TT-CHECK-ALL(KB, α, symbols, [])
________________________________________________________________________

function TT-CHECK-ALL(KB, α, symbols, model) returns true or false
   if EMPTY?(symbols) then
      if PL-TRUE?(KB, model) then return PL-TRUE?(α, model)
      else return true
   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))
```

- For *n* symbols, time complexity is $O(2^n)$, space complexity is $O(n)$

Logical equivalence

- Two sentences are logically equivalent iff true in same models: α ≡ ß iff α ╞ β and β ╞ α

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

Validity and satisfiability

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

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

Validity is connected to inference via the Deduction Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in some model

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

A sentence is unsatisfiable if it is true in no models

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

Satisfiability is connected to inference via the following:

$KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable

Proof methods

- Proof methods divide into (roughly) two kinds:
  - Natural Deduction: 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
    - Typically require transformation of sentences into a normal form
  - Model checking
    - truth table enumeration (always exponential in *n*)
    - improved backtracking, e.g., Davis--Putnam-Logemann-Loveland (DPLL)
    - heuristic search in model space (sound but incomplete)
      e.g., min-conflicts-like hill-climbing algorithms

This section covers **inference rules** that can be applied to derive a **proof**—a chain of conclusions that leads to the desired goal. The best-known rule is called **Modus Ponens** (Latin for *mode that affirms*) and is written

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

The notation means that, whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given, then the sentence β can be inferred. For example, if $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$ and $(WumpusAhead \wedge WumpusAlive)$ are given, then $Shoot$ can be inferred.

Another useful inference rule is **And-Elimination**, which says that, from a conjunction, any of the conjuncts can be inferred:

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

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

By considering the possible truth values of α and β, one can show easily that Modus Ponens and And-Elimination are sound once and for all. These rules can then be used in any particular instances where they apply, generating sound inferences without the need for enumerating models.

All of the logical equivalences in Figure 7.11 can be used as inference rules. For example, the equivalence for biconditional elimination yields the two inference rules

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

Not all inference rules work in both directions like this. For example, we cannot run Modus Ponens in the opposite direction to obtain $\alpha \Rightarrow \beta$ and α from β.

Let us see how these inference rules and equivalences can be used in the wumpus world. We start with the knowledge base containing R_1 through R_5 and show how to prove $\neg P_{1,2}$, that is, there is no pit in [1,2]. First, we apply 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.

We found this proof by hand, but we can apply any of the search algorithms in Chapter 3 to find a sequence of steps that constitutes a proof. We just need to define a proof problem as follows:

- 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.

7.4 Which of the following are correct?

a. $\mathit{False} \models \mathit{True}$.

b. $\mathit{True} \models \mathit{False}$.

c. $(A \land B) \models (A \Leftrightarrow B)$.

d. $A \Leftrightarrow B \models A \lor B$.

e. $A \Leftrightarrow B \models \neg A \lor B$.

f. $(A \land B) \Rightarrow C \models (A \Rightarrow C) \lor (B \Rightarrow C)$.

g. $(C \lor (\neg A \land \neg B)) \equiv ((A \Rightarrow C) \land (B \Rightarrow C))$.

h. $(A \lor B) \land (\neg C \lor \neg D \lor E) \models (A \lor B)$.

i. $(A \lor B) \land (\neg C \lor \neg D \lor E) \models (A \lor B) \land (\neg D \lor E)$.

j. $(A \lor B) \land \neg(A \Rightarrow B)$ is satisfiable.

k. $(A \Leftrightarrow B) \land (\neg A \lor B)$ is satisfiable.

l. $(A \Leftrightarrow B) \Leftrightarrow C$ has the same number of models as $(A \Leftrightarrow B)$ for any fixed set of proposition symbols that includes A, B, C.

7.4 In all cases, the question can be resolved easily by referring to the definition of entailment.

a. $\mathit{False} \models \mathit{True}$ is true because False has no models and hence entails every sentence AND because True is true in all models and hence is entailed by every sentence.

b. $\mathit{True} \models \mathit{False}$ is false.

c. $(A \wedge B) \models (A \;\Leftrightarrow\; B)$ is true because the left-hand side has exactly one model that is one of the two models of the right-hand side.

d. $A \;\Leftrightarrow\; B \models A \vee B$ is false because one of the models of $A \;\Leftrightarrow\; B$ has both A and B false, which does not satisfy $A \vee B$.

e. $A \;\Leftrightarrow\; B \models \neg A \vee B$ is true because the RHS is $A \;\Rightarrow\; B$, one of the conjuncts in the definition of $A \;\Leftrightarrow\; B$.

f. $(A \wedge B) \;\Rightarrow\; C \models (A \;\Rightarrow\; C) \vee (B \;\Rightarrow\; C)$ is true because the RHS is false only when both disjuncts are false, i.e., when A and B are true and C is false, in which case the LHS is also false. This may seem counterintuitive, and would not hold if $\Rightarrow$ is interpreted as "causes."

g. $(C \vee (\neg A \wedge \neg B)) \equiv ((A \Rightarrow C) \wedge (B \Rightarrow C))$ is true; proof by truth table enumeration, or by application of distributivity (Fig 7.11).

h. $(A \vee B) \wedge (\neg C \vee \neg D \vee E) \models (A \vee B)$ is true; removing a conjunct only allows more models.

i. $(A \lor B) \land (\neg C \lor \neg D \lor E) \models (A \lor B) \land (\neg D \lor E)$ is false; removing a disjunct allows fewer models.

j. $(A \lor B) \land \neg(A \Rightarrow B)$ *is* satisfiable; model has A and $\neg B$.

k. $(A \Leftrightarrow B) \land (\neg A \lor B)$ *is* satisfiable; RHS is entailed by LHS so models are those of $A \Leftrightarrow B$.

l. $(A \Leftrightarrow B) \Leftrightarrow C$ does have the same number of models as $(A \Leftrightarrow B)$; half the models of $(A \Leftrightarrow B)$ satisfy $(A \Leftrightarrow B) \Leftrightarrow C$, as do half the non-models, and there are the same numbers of models and non-models.

7.10 Decide whether each of the following sentences is valid, unsatisfiable, or neither. Verify your decisions using truth tables or the equivalence rules of Figure 7.11 (page 249).

a. $Smoke \Rightarrow Smoke$

b. $Smoke \Rightarrow Fire$

c. $(Smoke \Rightarrow Fire) \Rightarrow (\neg Smoke \Rightarrow \neg Fire)$

d. $Smoke \lor Fire \lor \neg Fire$

e. $((Smoke \land Heat) \Rightarrow Fire) \Leftrightarrow ((Smoke \Rightarrow Fire) \lor (Heat \Rightarrow Fire))$

f. $(Smoke \Rightarrow Fire) \Rightarrow ((Smoke \land Heat) \Rightarrow Fire)$

g. $Big \lor Dumb \lor (Big \Rightarrow Dumb)$

7.10

- **a**. Valid.
- **b**. Neither.
- **c**. Neither.
- **d**. Valid.
- **e**. Valid.
- **f**. Valid.
- **g**. Valid.

7.22 Minesweeper, the well-known computer game, is closely related to the wumpus world. A minesweeper world is a rectangular grid of N squares with M invisible mines scattered among them. Any square may be probed by the agent; instant death follows if a mine is probed. Minesweeper indicates the presence of mines by revealing, in each probed square, the *number* of mines that are directly or diagonally adjacent. The goal is to probe every unmined square.

a. Let $X_{i,j}$ be true iff square $[i, j]$ contains a mine. Write down the assertion that exactly two mines are adjacent to [1,1] as a sentence involving some logical combination of $X_{i,j}$ propositions.

b. Generalize your assertion from (a) by explaining how to construct a CNF sentence asserting that k of n neighbors contain mines.

c. Explain precisely how an agent can use DPLL to prove that a given square does (or does not) contain a mine, ignoring the global constraint that there are exactly M mines in all.

d. Suppose that the global constraint is constructed from your method from part (b). How does the number of clauses depend on M and N? Suggest a way to modify DPLL so that the global constraint does not need to be represented explicitly.

e. Are any conclusions derived by the method in part (c) invalidated when the global constraint is taken into account?

f. Give examples of configurations of probe values that induce *long-range dependencies* such that the contents of a given unprobed square would give information about the contents of a far-distant square. (*Hint*: consider an $N \times 1$ board.)

7.22

a. This is a disjunction with 28 disjuncts, each one saying that two of the neighbors are true and the others are false. The first disjunct is

$$X_{2,2} \land X_{1,2} \land \neg X_{0,2} \land \neg X_{0,1} \land \neg X_{2,1} \land \neg X_{0,0} \land \neg X_{1,0} \land \neg X_{2,0}$$

The other 27 disjuncts each select two different $X_{i,j}$ to be true.

b. There will be $\binom{n}{k}$ disjuncts, each saying that k of the n symbols are true and the others false.

c. For each of the cells that have been probed, take the resulting number n revealed by the game and construct a sentence with $\binom{n}{8}$ disjuncts. Conjoin all the sentences together. Then use DPLL to answer the question of whether this sentence entails $X_{i,j}$ for the particular i, j pair you are interested in.

d. To encode the global constraint that there are M mines altogether, we can construct a disjunct with $\binom{M}{N}$ disjuncts, each of size N. Remember, $\binom{M}{N=M!/(M-N)!}$. So for a Minesweeper game with 100 cells and 20 mines, this will be morre than 10^{39}, and thus cannot be represented in any computer. However, we can represent the global constraint within the DPLL algorithm itself. We add the parameter *min* and *max* to the DPLL function; these indicate the minimum and maximum number of unassigned symbols that must be true in the model. For an unconstrained problem the values 0 and N will be used for these parameters. For a mineseeper problem the value M will be used for both *min* and *max*. Within DPLL, we fail (return false) immediately if *min* is less than the number of remaining symbols, or if *max* is less than 0. For each recursive call to DPLL, we update *min* and *max* by subtracting one when we assign a true value to a symbol.

e. No conclusions are invalidated by adding this capability to DPLL and encoding the global constraint using it.

f. Consider this string of alternating 1's and unprobed cells (indicated by a dash):

```
|-|1|-|1|-|1|-|1|-|1|-|1|-|1|-|
```

There are two possible models: either there are mines under every even-numbered dash, or under every odd-numbered dash. Making a probe at either end will determine whether cells at the far end are empty or contain mines.

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

Figure 7.14 A grammar for conjunctive normal form, Horn clauses, and definite clauses. A clause such as $A \wedge B \Rightarrow C$ is still a definite clause when it is written as $\neg A \vee \neg B \vee C$, but only the former is considered the canonical form for definite clauses. One more class is the k-CNF sentence, which is a CNF sentence where each clause has at most k literals.

definite clause – exactly one literal is positive

Horn clause – at most one literal is positive

Resolution

Conjunctive Normal Form (CNF)

conjunction of disjunctions of literals

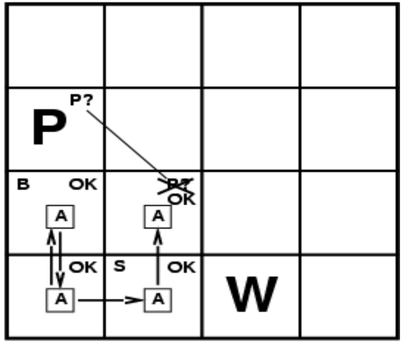
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

- Resolution inference rule (for CNF):

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

where ℓ_i and m_j are complementary literals.

E.g., $\dfrac{P_{1,3} \vee P_{2,2}, \quad \neg P_{2,2}}{P_{1,3}}$



- Resolution is sound and complete for propositional logic

Conversion to CNF

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

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:

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

4. Apply distributivity 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

Soundness of resolution inference rule:

$$\frac{\neg(l_i \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k) \Rightarrow l_i \qquad \neg m_j \Rightarrow (m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}{\neg(l_i \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k) \Rightarrow (m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n)}$$

Resolution algorithm

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

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

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

Figure 7.12 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

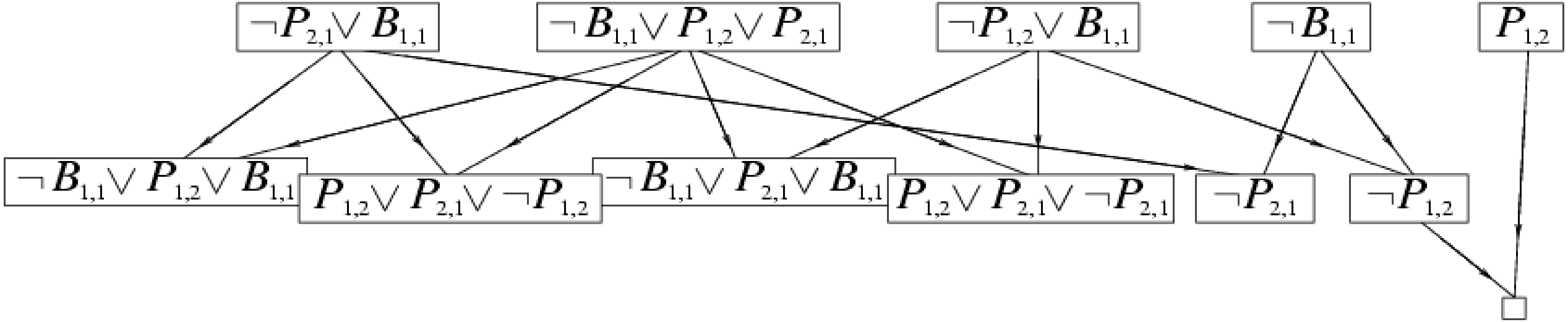
We can apply the resolution procedure to a very simple inference in the wumpus world. When the agent is in [1,1], there is no breeze, so there can be no pits in neighboring squares. The relevant knowledge base is

$$KB = R_2 \wedge R_4 = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$$

and we wish to prove α which is, say, $\neg P_{1,2}$. When we convert $(KB \wedge \neg\alpha)$ into CNF, we obtain the clauses shown at the top of Figure 7.13. The second row of the figure shows clauses obtained by resolving pairs in the first row. Then, when $P_{1,2}$ is resolved with $\neg P_{1,2}$, we obtain the empty clause, shown as a small square. Inspection of Figure 7.13 reveals that many resolution steps are pointless. For example, the clause $B_{1,1} \vee \neg B_{1,1} \vee P_{1,2}$ is equivalent to $True \vee P_{1,2}$ which is equivalent to $True$. Deducing that $True$ is true is not very helpful. Therefore, any clause in which two complementary literals appear can be discarded.

Resolution example

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



Forward and backward chaining

- Horn Form (restricted)

  KB = conjunction of Horn clauses

  - Horn clause =
    - proposition symbol; or
    - (conjunction of symbols) $\Rightarrow$ symbol
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- 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

Knowledge bases containing only definite clauses are interesting for three reasons:

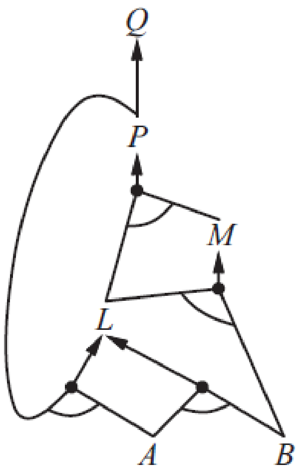
1. Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. (See Exercise 7.13.) For example, the definite clause $(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1})$ can be written as the implication $(L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$. In the implication form, the sentence is easier to understand: it says that if the agent is in [1,1] and there is a breeze, then [1,1] is breezy. In Horn form, the premise is called the **body** and the conclusion is called the **head**. A sentence consisting of a single positive literal, such as $L_{1,1}$, is called a **fact**. It too can be written in implication form as $True \Rightarrow L_{1,1}$, but it is simpler to write just $L_{1,1}$.
2. Inference with Horn clauses can be done through the **forward-chaining** and **backward-chaining** algorithms, which we explain next. Both of these algorithms are natural, in that the inference steps are obvious and easy for humans to follow. This type of inference is the basis for **logic programming**, which is discussed in Chapter 9.
3. Deciding entailment with Horn clauses can be done in time that is *linear* in the size of the knowledge base—a pleasant surprise.

Forward chaining

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

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

(a)



(b)

Figure 7.16 (a) A set of Horn clauses. (b) The corresponding AND–OR graph.

Forward chaining algorithm

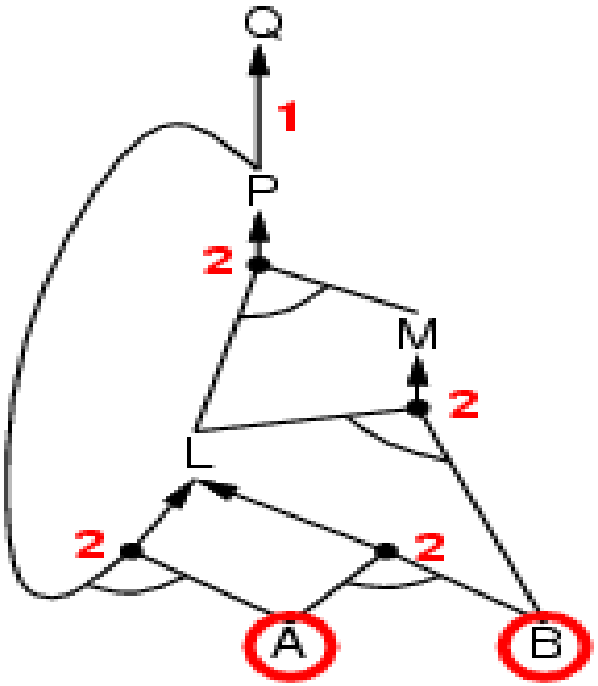
```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially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

  while agenda is not empty do
     p ← POP(agenda)
     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 agenda
  return false
```

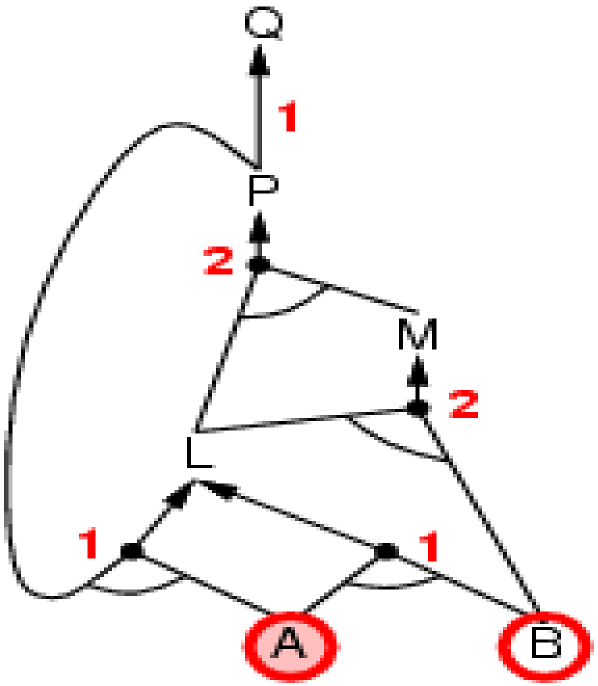
Figure 7.15 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count 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 $P \Rightarrow Q$ and $Q \Rightarrow P$.

- Forward chaining is sound and complete for Horn KB

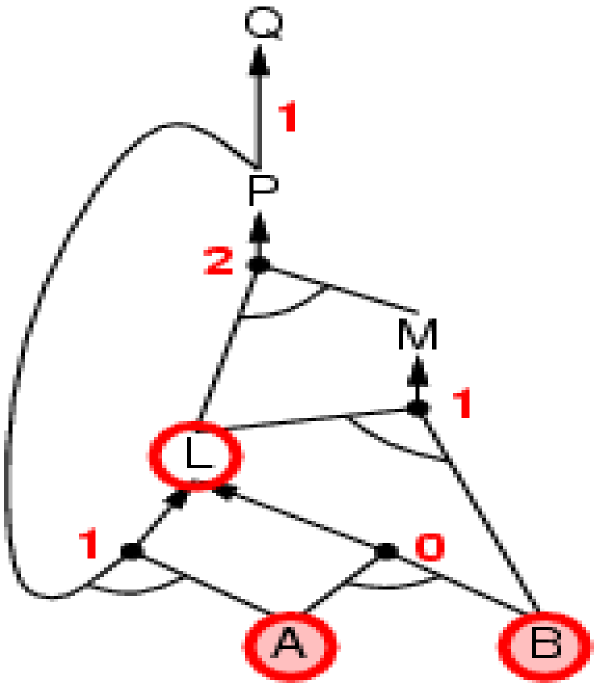
Forward chaining example



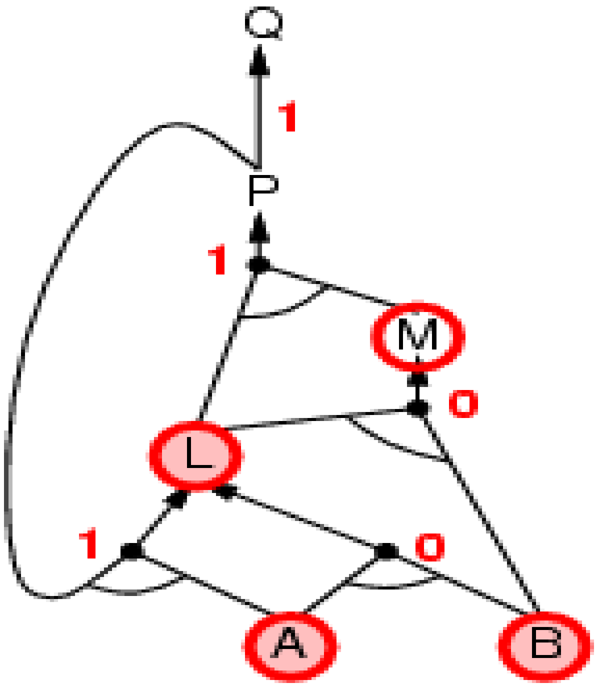
Forward chaining example



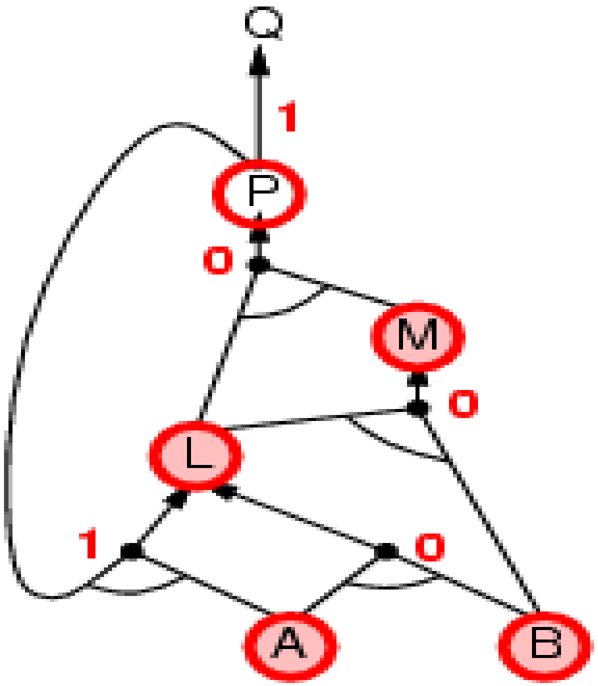
Forward chaining example



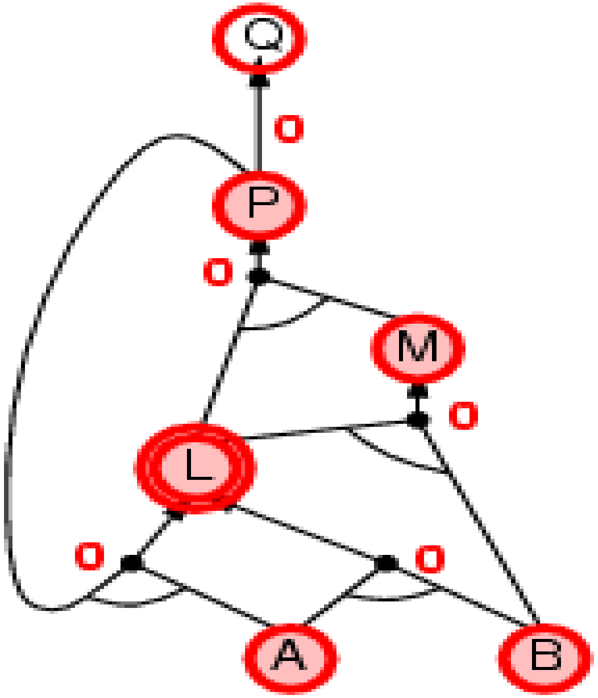
Forward chaining example



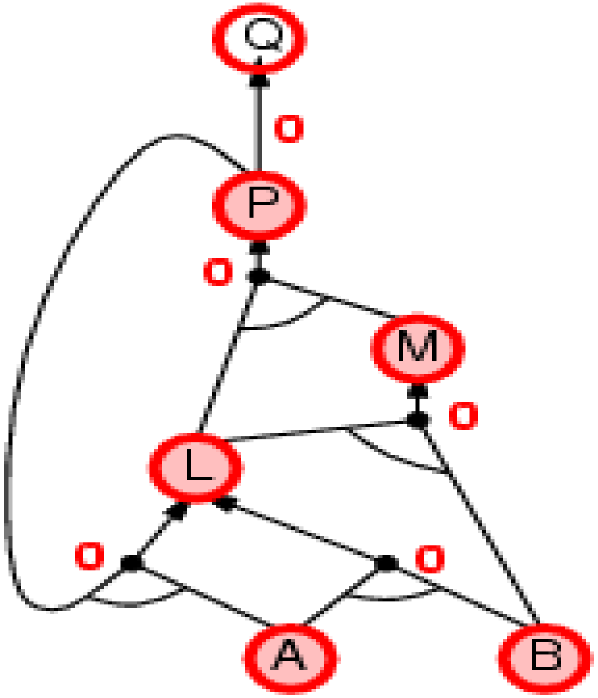
Forward chaining example



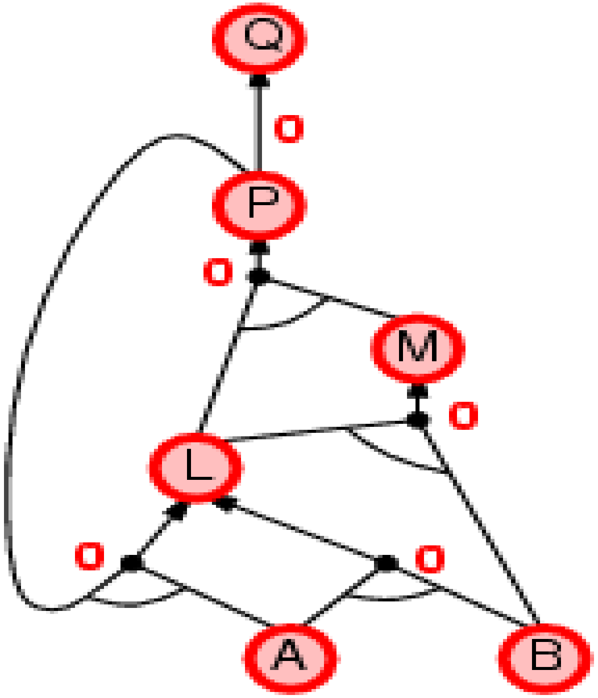
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

- FC derives every atomic sentence that is entailed by *KB*
  1. FC reaches a fixed point where no new atomic sentences are derived
  2. Consider the final state as a model *m*, assigning true/false to symbols
  3. Every clause in the original *KB* is true in *m*
  4. $a_1 \wedge \ldots \wedge a_k \Rightarrow b$
  5. Hence *m* is a model of *KB*
  6. If $KB \models q$, *q* is true in every model of *KB*, including *m*

Backward chaining

Idea: work backwards from the query *q*:

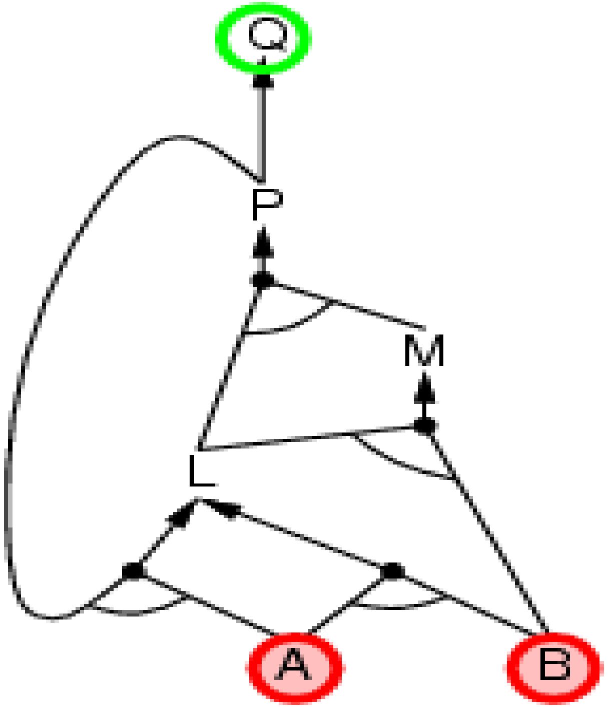
- to prove *q* by BC,
  - check if *q* is known already, or
  - prove by BC all premises of some rule concluding *q*

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

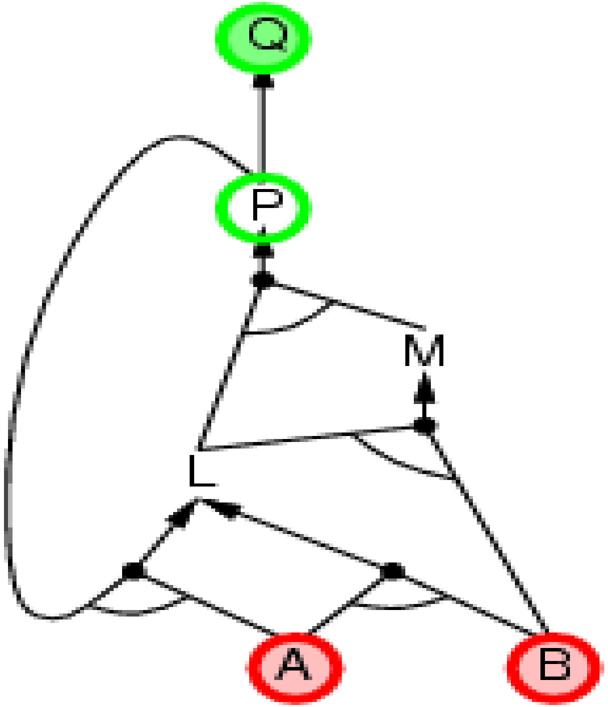
Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

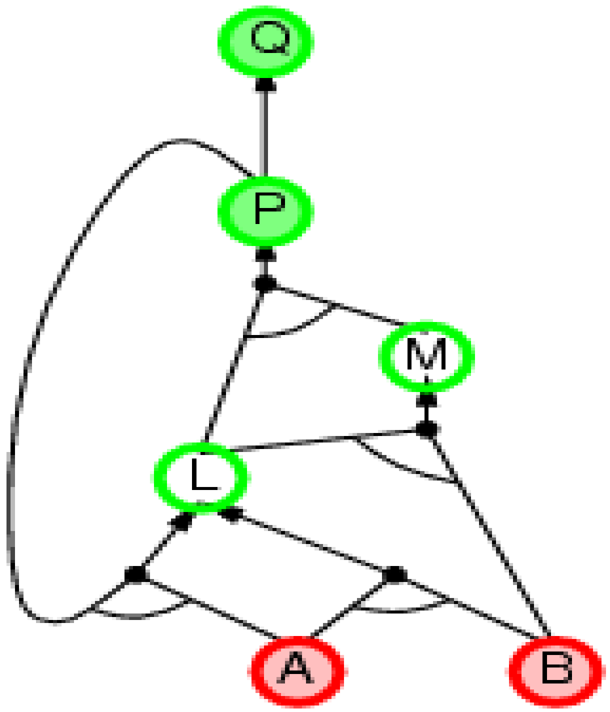
Backward chaining example



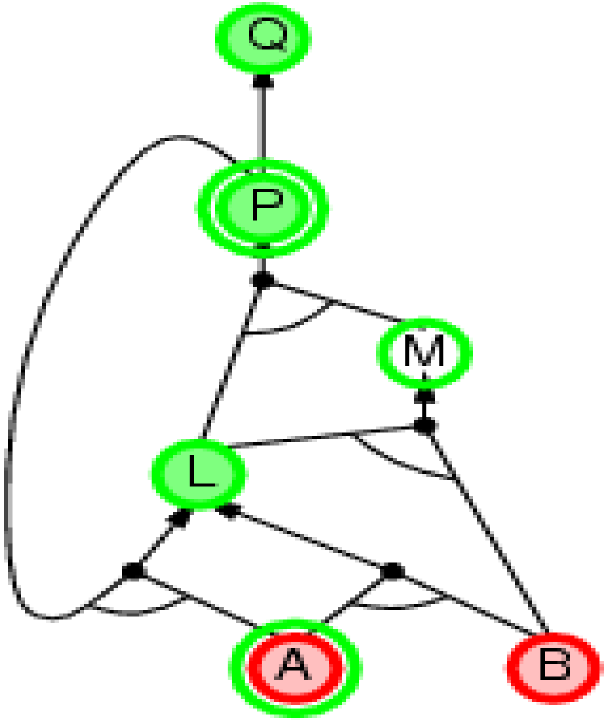
Backward chaining example



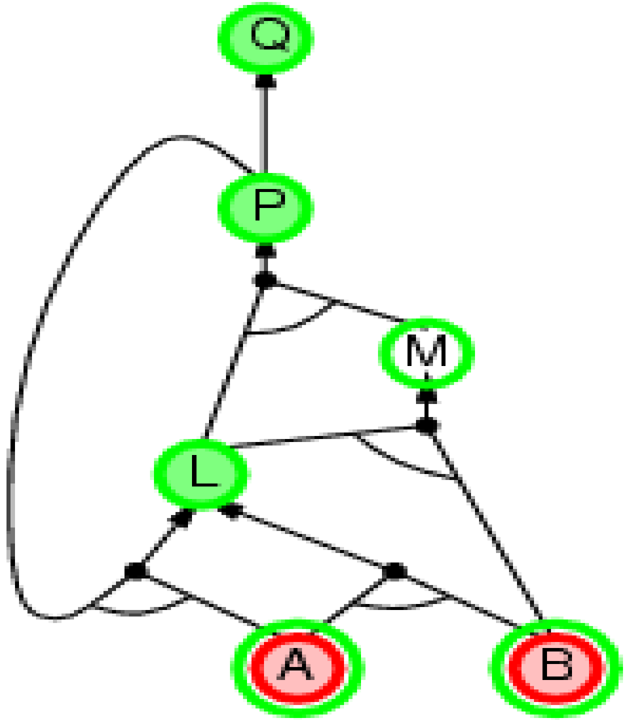
Backward chaining example



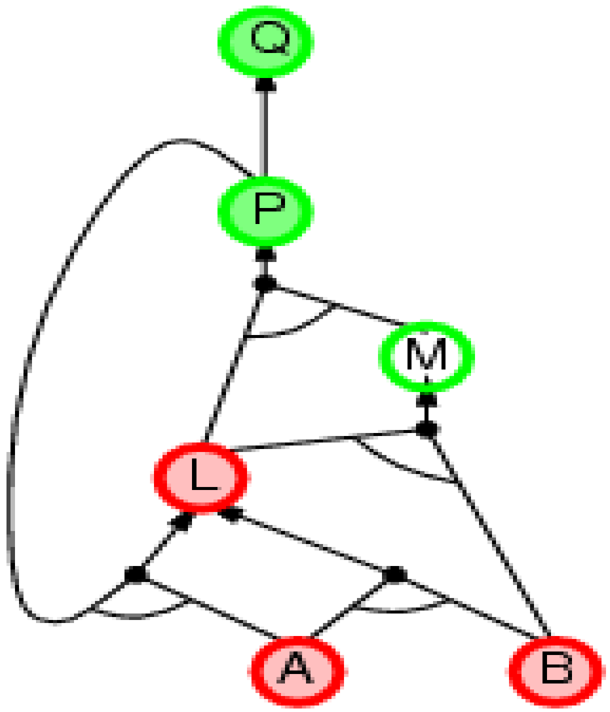
Backward chaining example



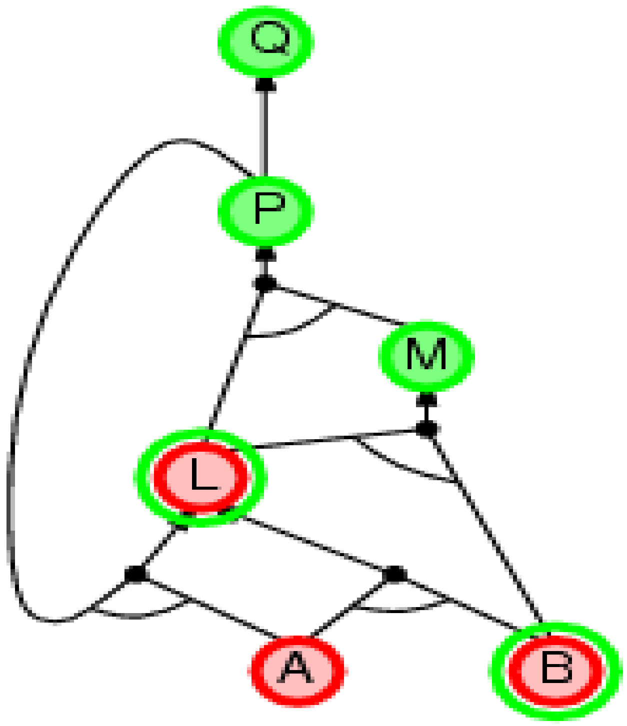
Backward chaining example



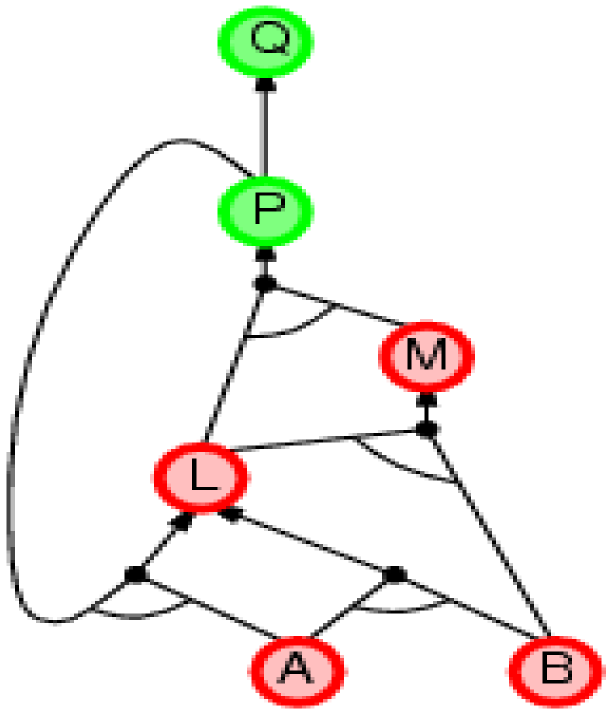
Backward chaining example



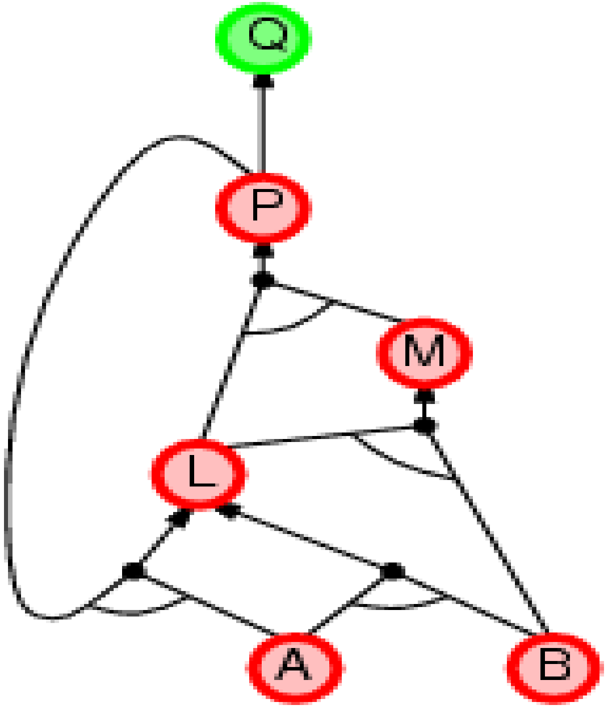
Backward chaining example



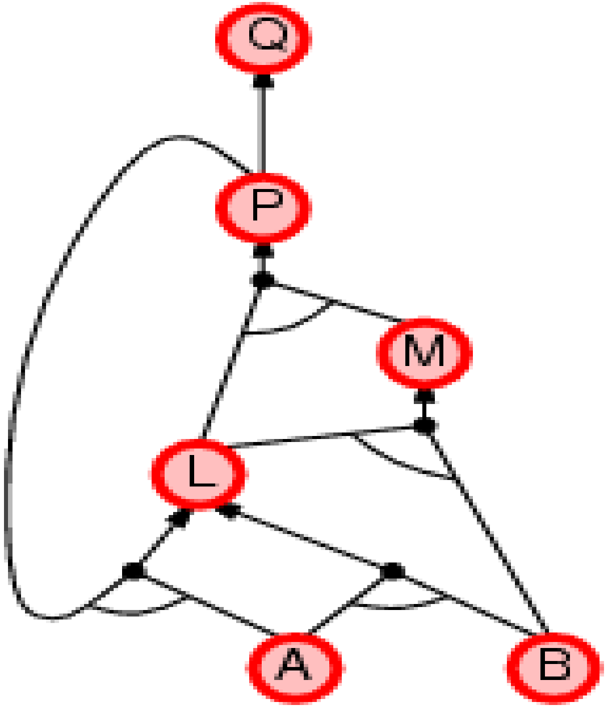
Backward chaining example



Backward chaining example



Backward chaining example



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

Efficient propositional inference

Two families of efficient algorithms for propositional inference:

Complete backtracking search algorithms

- DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - `WalkSAT` algorithm

The DPLL algorithm

Determine if an input propositional logic sentence (in CNF) is satisfiable.

Improvements over truth table enumeration:

1. Early termination

   A clause is true if any literal is true.

   A sentence is false if any clause is false.

2. Pure symbol heuristic

   Pure symbol: always appears with the same "sign" in all clauses.

   e.g., In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.

   Make a pure symbol literal true.

3. Unit clause heuristic

   Unit clause: only one literal in the clause

   The only literal in a unit clause must be true.

The DPLL algorithm

```
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}))
```

Figure 7.17 The DPLL algorithm for checking satisfiability of a sentence in propositional logic. The ideas behind FIND-PURE-SYMBOL and FIND-UNIT-CLAUSE are described in the text; each returns a symbol (or null) and the truth value to assign to that symbol. Like TT-ENTAILS?, DPLL operates over partial models.

The WalkSAT algorithm

- Incomplete, local search algorithm
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses
- 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 flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for $i = 1$ **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    with probability *p* 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*

Figure 7.18 The WALKSAT algorithm for checking satisfiability by randomly flipping the values of variables. Many versions of the algorithm exist.

Hard satisfiability problems

- Consider random 3-CNF sentences. e.g.,

  $(\neg D \lor \neg B \lor C) \land (B \lor \neg A \lor \neg C) \land (\neg C \lor \neg B \lor E) \land (E \lor \neg D \lor B) \land (B \lor E \lor \neg C)$

  m = number of clauses

  n = number of symbols

  – Hard problems seem to cluster near $m/n = 4.3$ (critical point)

Hard satisfiability problems

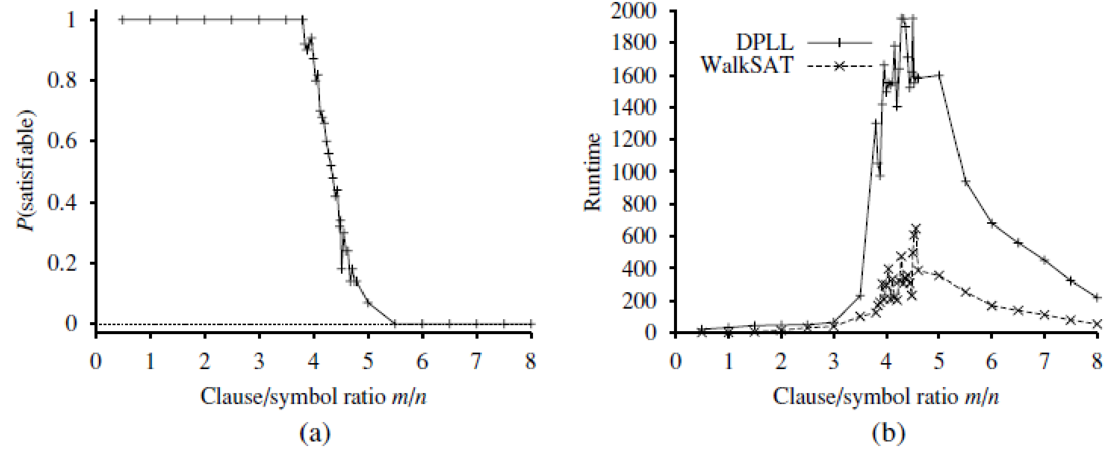


Figure 7.19 (a) Graph showing the probability that a random 3-CNF sentence with $n = 50$ symbols is satisfiable, as a function of the clause/symbol ratio m/n. (b) Graph of the median run time (measured in number of recursive calls to DPLL, a good proxy) on random 3-CNF sentences. The most difficult problems have a clause/symbol ratio of about 4.3.

- Median runtime for 100 satisfiable random 3-CNF sentences, $n = 50$

Inference-based agents in the wumpus world

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$
$\neg W_{1,1}$
$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$
$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$
$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$
$\neg W_{1,1} \vee \neg W_{1,2}$
$\neg W_{1,1} \vee \neg W_{1,3}$
…

Exactly one wumpus = at least one wumpus AND at most one wumpus

$\Rightarrow$ 64 distinct proposition symbols, 155 sentences

```
function PL-WUMPUS-AGENT(percept) returns an action
    inputs: percept, a list, [stench,breeze,glitter]
    static: KB, initially containing the "physics" of the wumpus world
            x, y, orientation, the agent's position (init. [1,1]) and orient. (init. right)
            visited, an array indicating which squares have been visited, initially false
            action, the agent's most recent action, initially null
            plan, an action sequence, initially empty

    update x,y,orientation, visited based on action
    if stench then TELL(KB, S_x,y) else TELL(KB, ¬ S_x,y)
    if breeze then TELL(KB, B_x,y) else TELL(KB, ¬ B_x,y)
    if glitter then action ← grab
    else if plan is nonempty then action ← POP(plan)
    else if for some fringe square [i,j], ASK(KB, (¬ P_i,j ∧ ¬ W_i,j)) is true or
            for some fringe square [i,j], ASK(KB, (P_i,j ∨ W_i,j)) is false then do
        plan ← A*-GRAPH-SEARCH(ROUTE-PB([x,y], orientation, [i,j],visited))
        action ← POP(plan)
    else action ← a randomly chosen move
    return action
```

Additional considerations

- What if there was no stench in step 3 but there is stench in step 4?
- We cannot assert “stench” to the KB at step 3 since “~stench” was already asserted at step 4.
- What solution is there?
- Stench^4

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

- For every time t and every location [x,y],

$L_{x,y}^{t} \wedge FacingRight^{t} \wedge Forward^{t} \Rightarrow L_{x+1,y}^{t}$

- Rapid proliferation of clauses

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - syntax: formal structure of sentences
  - semantics: truth of sentences wrt models
  - entailment: necessary truth of one sentence given another
  - inference: deriving sentences from other sentences
  - soundness: derivations produce only entailed sentences
  - completeness: derivations can produce all entailed sentences
- Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.
- Resolution is complete for propositional logic
  Forward, backward chaining are linear-time, complete for Horn clauses
- Propositional logic lacks expressive power

AI