EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Artificial Intelligence
Chapter 7: Logical Agents

Andreas Zell

After the Textbook: Artificial Intelligence,
A Modern Approach
by Stuart Russel and Peter Norvig (3rd Edition)

7.1 Knowledge-Based Agents

7.2 The Wumpus World

7.3 Logic

7.4 Propositional Logic

7.5 Propositional Theorem Proving

7.6 Effective Propositional Model Checking

7.1 Knowledge-Based Agents

- Logical agents are always definite – each proposition is either true/false or unknown (agnostic)
- Knowledge representation language – a language used to express knowledge about the world
  - Declarative approach – language is designed to be able to easily express knowledge for the world the language is being implemented for
  - Procedural approach – encodes desired behaviors directly in program code

7.1 Knowledge-Based Agents

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Sentence – a statement expressing a truth about the world in the knowledge representation language
- Knowledge Base (KB) – a set of sentences describing the world
  - Background knowledge – initial knowledge in KB
  - Knowledge level – we only need to specify what the agent knows and what its goals are in order to specify its behavior
  - Tell(P) – function that adds knowledge P to the KB
  - Ask(P) – function that queries the agent about the truth of P

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

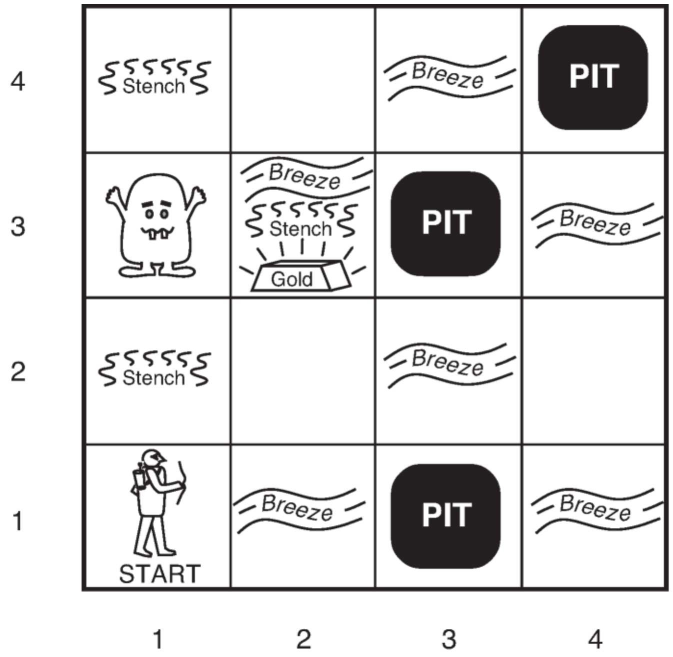
- **Inference** – the process of deriving new sentences from the knowledge base
  - *When the agent draws a conclusion from available information, it is guaranteed to be correct if the available information is correct*

```
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
```

A generic knowledge-based agent

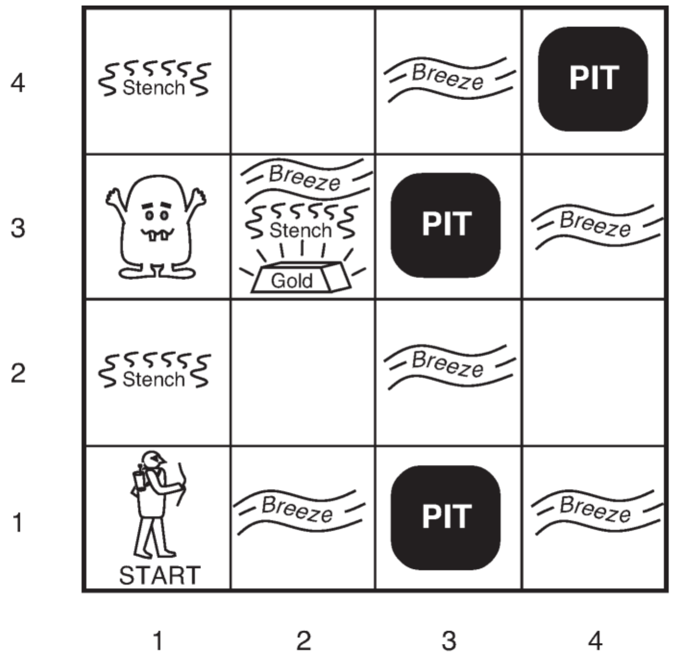
7.2 The Wumpus World

- 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 it
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - Releasing drops the gold in same square



7.2 The Wumpus World

- Performance measure
  - gold +1000,
  - PIT/wumpus -1000
  - -1 per action,
  - -10 for using the arrow
- Actuators:
  - TurnLeft (90°),
  - TurnRight (90°),
  - Forward,
  - Grab (gold),
  - Shoot (arrow),
  - Climb (at 1,1)
- Sensors:
  - Stench, Breeze, Glitter, Bump, Scream



7.2 The Wumpus World

- 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

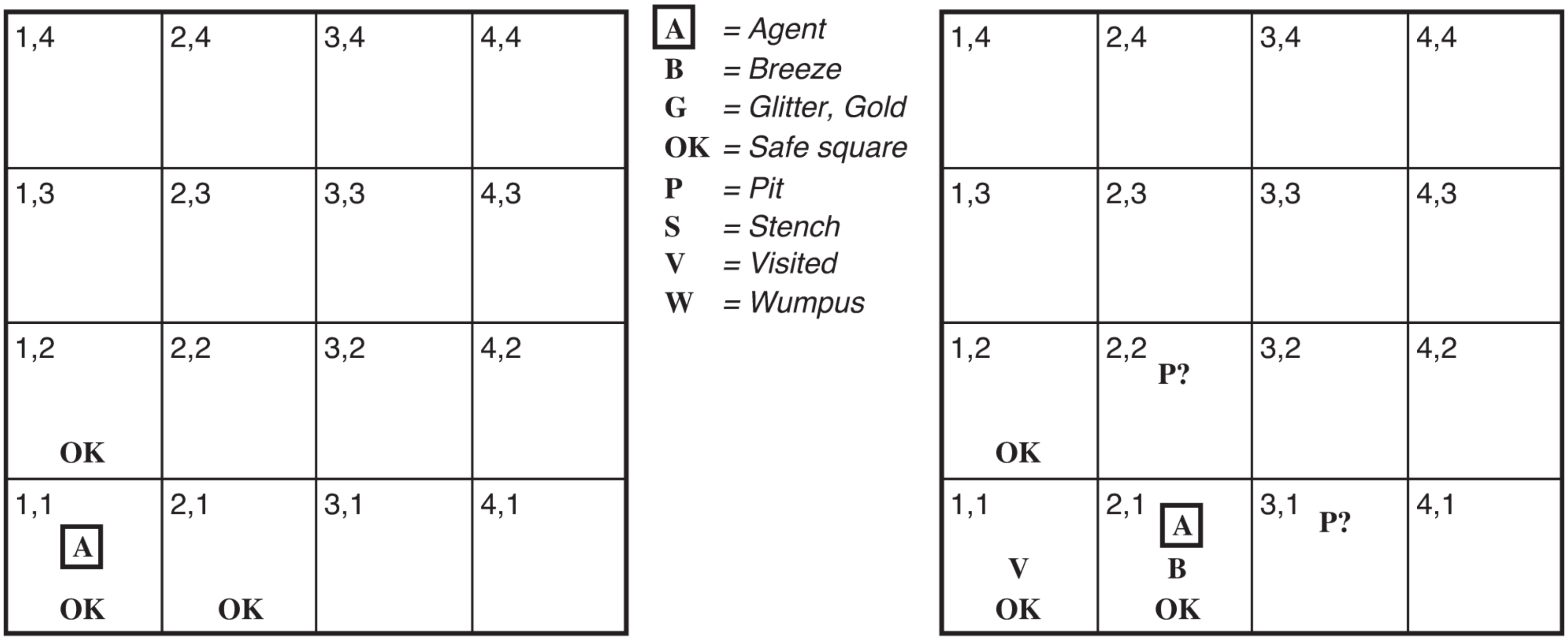
First percept at [1,1]

[*None, None, None, None, None*]

Percept at [2,1]

[*None, Breeze, None, None, None*]

Stench, Breeze, Glitter, Bump, Scream



(a) (b)

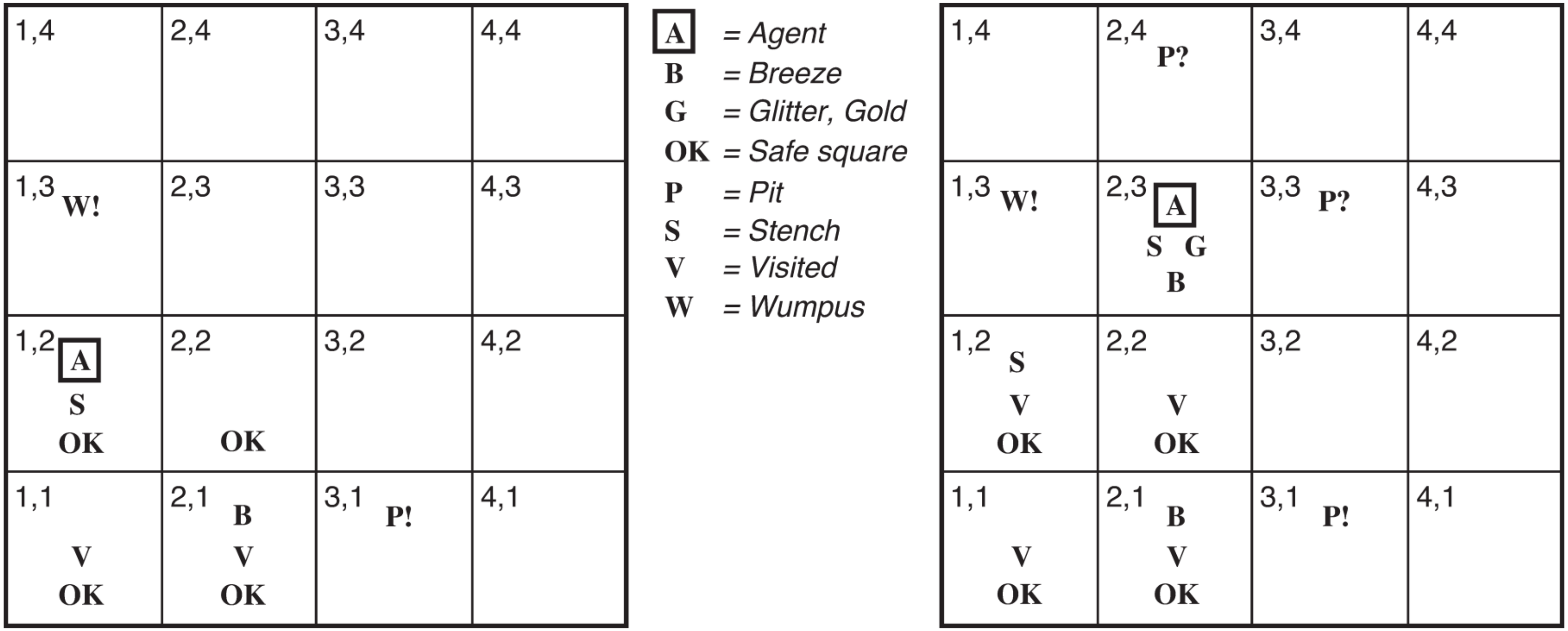
7.2 The Wumpus World

Percept at [1,2]

[Stench, None, None, None, None]

Percept at [2,3]

[Stench, Breeze, Glitter, None, None]

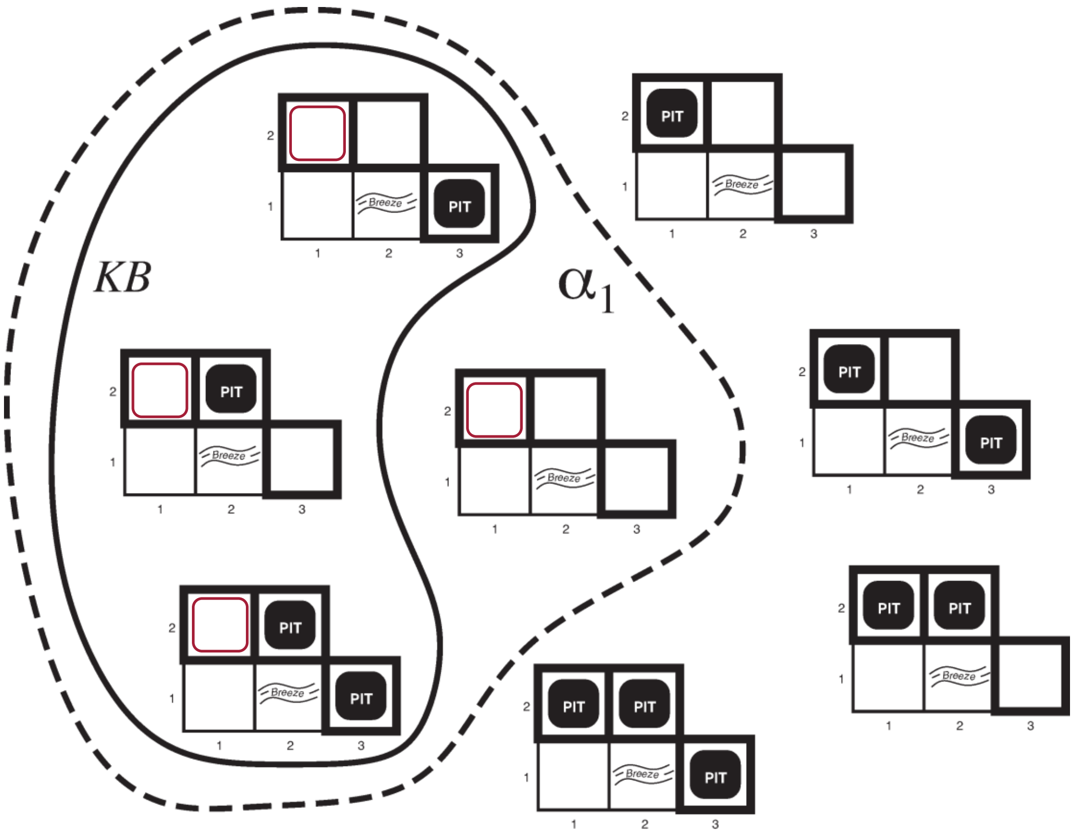


(a) (b)

7.3 Logic

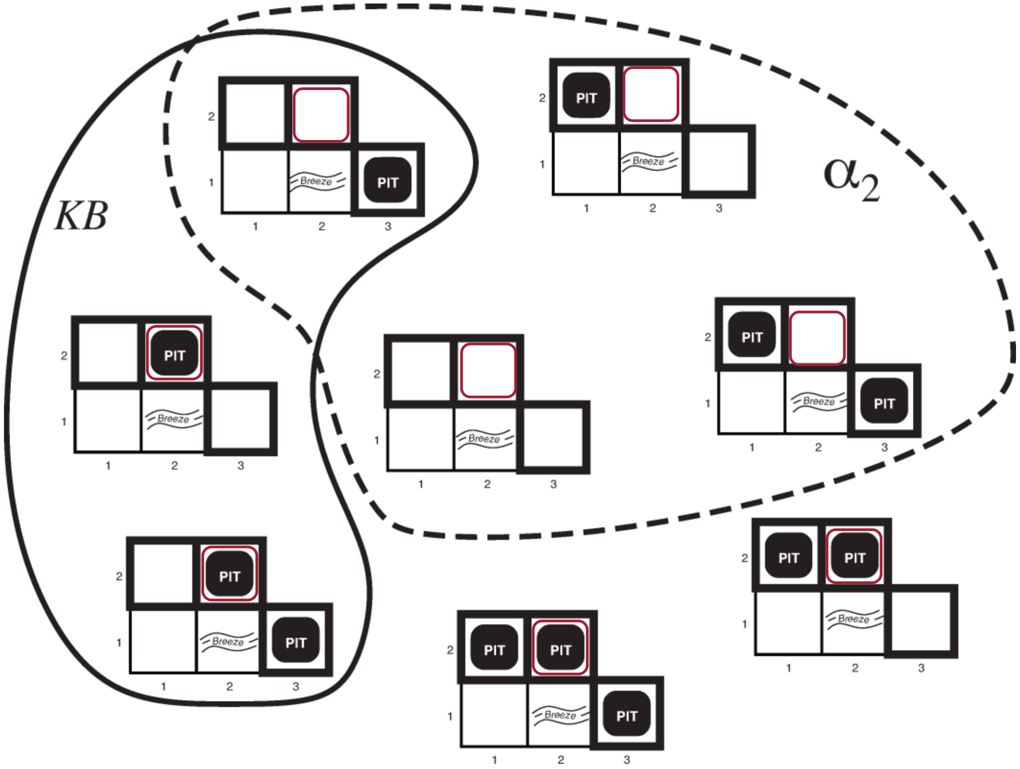
- Logics – formal languages for representing information such that conclusions can be drawn
- Syntax – description of a representative language in terms of well-formed sentences of the language
- Semantics – defines the “meaning” (truth) of a sentence in the representative language w.r.t. each possible world
- Model – the world being described by a KB
- Satisfaction – model *m* satifies a sentence *α*, if *α* is true in *m*

- Entailment – the concept that a sentence follows from another sentence:
  - $\alpha \models \beta$ if α is true, then β must also be true.
- Logical inference – the process of using entailment to derive conclusions
- Model checking – enumeration of all possible models to ensure that a sentence *α* is true in all models in which KB is true
- $M(\alpha)$ is the set of all models of α



KB = wumpus-world rules + observations

α_1 = “[1,2] is safe”, KB $\models \alpha_1$, proved by model checking



KB = wumpus-world rules + observations

α_2 = “[2,2] is safe”, KB $\nvDash$ α_2

7.3 Logic

- If an inference algorithm *i* can derive α from KB we write KB $\vdash_i$ α.
- Sound (truth-preserving) inference – an inference algorithm that derives only entailed sentences
  - *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*
- Complete inference procedure – an inference proc. that can derive any sentence that is entailed
- Grounding – the connection between logical reasoning processes and the real environment in which the agent exists

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Atomic sentence – consists of a single propositional symbol, which is *True* or *False*
- Complex sentence – sentence constructed from simpler sentences using parentheses and logical connectives:
  - ¬ (not) – negation
  - ∧ (and) – conjunction
  - ∨ (or) – disjunction
  - ⇒ (implies) – implication (premise=>conclusion)
  - ⇔ (if and only if) – biconditional

Highest priority

Lowest priority

7.4 Propositional Logic

- **Truth table** – a (simple) representation of a complex sentence by enumerating its truth in terms of the possible values of each of its symbols.
- Truth table for connectives:

P	Q	¬P	P ∧ Q	P ∨ Q	P=>Q	P<=>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

7.4 Propositional Logic

- Wumpus World Symbols:
  - $P_{x,y}$ is true if there is a pit in [x,y]
  - $W_{x,y}$ is true if there is a wumpus in [x,y]
  - $B_{x,y}$ is true if there is a breeze in [x,y]
  - $S_{x,y}$ is true if there is a stench in [x,y]

- Sentences R_i:
  - No pit in [1,1]
    - R_1: $\neg P_{1,1}$
  - Pits cause breezes in adjacent squares
    - R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
    - R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
  - For first two squares
    - R_4: $\neg B_{1,1}$
    - R_5: $B_{2,1}$

7.4 Propositional Logic by Model Checking

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
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*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Fig. 7.8: Truth Table for Wumpus World KB, consisting of 2^7 = 128 rows, one each for the different assignments of truth values to the 7 proposition symbols $B_{1,1}$, …, $P_{3,1}$. KB is true if R_1 through R_5 are true, which occurs just in 3 rows.

7.4.4 Propositional Model Checking

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

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

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

Figure 7.8 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*.

7.5 Propositional Theorem Proving

- Knowledge Base can be represented as a conjunction of all its statements since it asserts that all statements are true.
- *Every known inference algorithm for propositional logic has a worst-case complexity exponential in the size of the input.*
- Logical equivalence – two sentences α and β are logically equivalent if they are true in the same set of models.
- Validity – a sentence is valid if it is true in *all* models.
- Valid sentences are also called tautologies – sentences that are necessarily true.

7.5 Propositional Theorem Proving

- Deduction Theorem – *For any sentences α and β, $\alpha \models \beta$ if and only if the sentence $(\alpha \Rightarrow \beta)$ is valid.*
- Satisfiablility – a sentence is satisfiable if it is true in *some* model.
  - *Determining satisfiablity in propositional logic is NP-complete.*
  - Proof by contradiction: *$\alpha \models \beta$ if and only if the sentence* $\neg(\alpha \Rightarrow \beta)$ or rather $(\alpha \wedge \neg\beta)$ is unsatisfiable.
- Inferentially equivalent – two sentences α and β are inferentially equivalent if the satisfiablity of α implies the satisfiablity of β and vice versa.

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

Fig. 7.11 Standard logical equivalences. The symbols α,β and γ stand for arbitrary sentences of propositional logic

7.5 Propositional Theorem Proving

- Inference rules used to derive a proof
- Common Patterns:
  - Modus Pones $$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$
  - And-Elimination $$\frac{\alpha \wedge \beta}{\alpha}$$
- *Finding a proof can be efficient since irrelevant propositions can be ignored.*
- Monotonicity says that the set of entailed sentences can only increase as information is added to KB

7.5 Propositional Theorem Proving

Example to prove $\neg P_{1,2}$ from R_1 through R_5:

- Applying biconditional elimination to R_2 to obtain
  R_6: $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$
- Applying And-Elimination to obtain
  R_7: $((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$
- Contraposition gives
  R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \lor P_{2,1}))$
- Modus Ponens with R_8 and the percept $\neg B_{1,1}$ gives
  R_9: $\neg(P_{1,2} \lor P_{2,1})$
- De Morgan's rule gives
  R_{10}: $\neg P_{1,2} \land \neg P_{2,1}$
  that is, neither P_{12} nor P_{21} contains a pit.

Conjunctive Normal Form (CNF) – every sentence of propositional logic is *logically equivalent* to a conjunction of clauses. E.g. Convert $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ to CNF:

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 distributivity law ($\vee$ over $\wedge$) and flatten

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

7.5 Propositional Theorem Proving

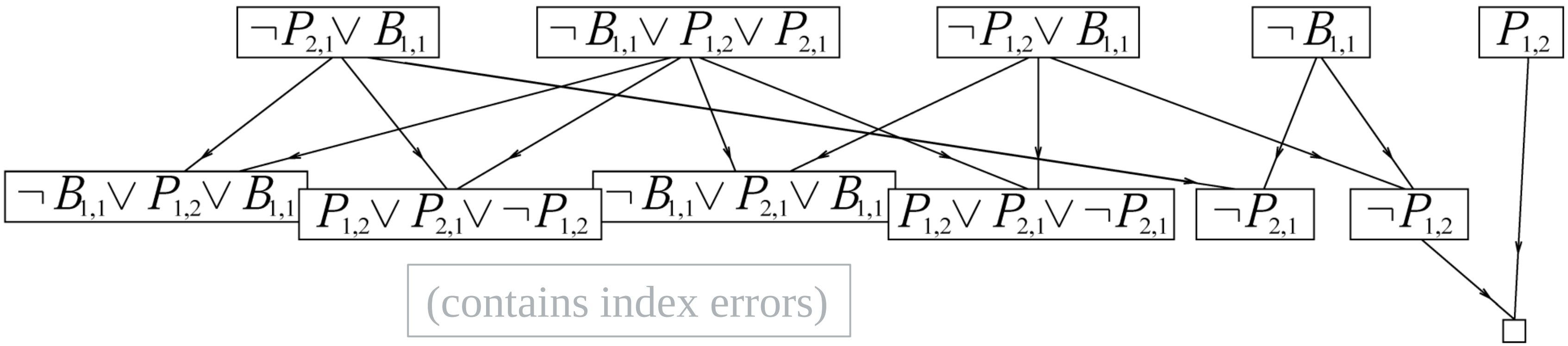
function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \land \neg\alpha$
  $new \leftarrow \{\ \}$
  loop do
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      $resolvents \leftarrow$ PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      $new \leftarrow new \cup resolvents$
    if $new \subseteq clauses$ **then return** *false*
    $clauses \leftarrow clauses \cup new$

Figure 7.9 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

Resolution algorithm: Proof by contradiction, i.e., show $KB \land \neg\alpha$ unsatisfiable

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



Resolution Example from Wumpus World

7.5 Propositional Theorem Proving

- Definite clause – disjunction of literals, of which exactly one is positive e.g. $\neg P_1 \lor \neg P_2 \lor \neg P_3 \lor P_4$
- Horn clause – a disjunction of literals at most one of which is positive e.g. $\neg P_1 \lor \neg P_2$, or $\neg P_3 \lor P_4$
  - Can be used with forward chaining or backward chaining
  - Deciding entailment is *linear* in the size of KB
- Goal clause – a clause with no positive literals, $\neg P_1 \lor \neg P_2$
- Forward chaining – a *sound* and *complete* inference algorithm that is essentially Modus Ponens
  - *data-driven reasoning*; reasoning which starts from known data
- Backward chaining – *goal-directed reasoning*; reasoning that works backward from goal
  - Often works in less than linear time as it avoids redundant facts

7.5 Propositional Theorem Proving

```
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
```

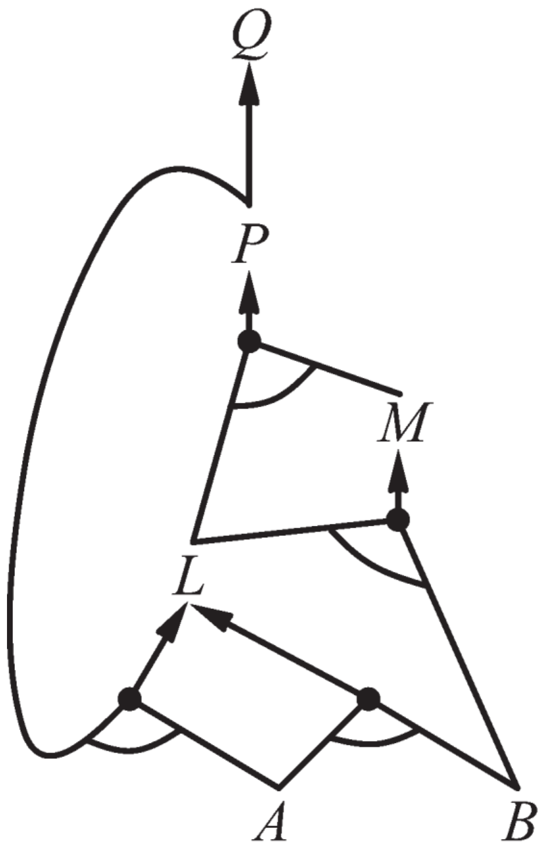
Figure 7.12 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count 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 $Q \Rightarrow P$.

7.5 Propositional Theorem Proving

A set of Horn Clauses

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

And the corresponding AND-OR graph:



EBERHARD KARLS UNIVERSITÄT TÜBINGEN

KB = ¬P∨Q, ¬L∨¬M∨P, ¬B∨¬L∨M, ¬A∨¬P∨L, ¬A∨¬B∨L, A, B

Question: KB ⊨ Q ?

KB ⊨ Q if and only if KB, ¬Q ⊨ false

So ¬Q, ¬P∨Q, ¬L∨¬M∨P, ¬B∨¬L∨M, ¬A∨¬P∨L, ¬A∨¬B∨L, A, B

¬P

¬L∨¬M

¬L∨¬B∨¬L (factoring, elimination of duplicate Literals)

¬A∨¬B

¬B

false (empty clause)

Unit resolution (l_i are literals):

$$\frac{l_1 \vee l_2 \vee \ldots l_i \vee \ldots \vee l_n, \quad \neg l_i}{l_1 \vee l_2 \vee \ldots \vee \ldots \vee l_n}$$

(l_i deleted)

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Full Resolution:

Implicit "and"

$$\frac{l_1 \vee l_2 \vee \ldots l_i \vee \ldots \vee l_k \, , \;\; m_1 \vee m_2 \vee \ldots m_j \vee \ldots \vee m_n}{l_1 \vee l_2 \vee \ldots \quad \vee \ldots \vee l_n \vee m_1 \vee m_2 \vee \ldots \quad \vee \ldots \vee m_n}$$

"or"

where the l_i and m_j are complementary literals.
Multiple copies of a literal are reduced to one (factoring).

Examples:

- ¬P∨Q, ¬L∨¬M∨P
  Q∨¬L∨¬M

  ¬B∨¬L∨M, ¬B∨¬P∨L
  ¬B∨M∨¬B∨¬P
  factoring

- ¬A∨B, ¬A∨C
  cannot be resolved

7.6 Effective Propositional Model Checking

- Davis-Putnam algorithm (DPLL) – an algorithm for checking satisfiability based on the fact that satisfiability is commutative. Essentially, it is a DFS method of *model checking*.
- Fundamental algorithm:

  DP(*clauses, symbols, model*)

  - If (all *clauses* are *true in model*) return true;
  - If (there is a *false* clause in model) return false;
  - *P* = next unassigned symbol in *symbols*;
  - return DP (*clauses, symbols, model* + *{P / true}*) OR
    DP (*clauses, symbols, model* + *{P / false}*);

7.6 Effective Propositional Model Checking

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

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 7.14 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.

7.6 Effective Propositional Model Checking

- Heuristics in the Davis-Putnam algorithm:
  - Early termination – short-circuit logical evaluations.
    A clause is true if *any* literal in it is true.
    A sentence is false if *any* clause in it is false.
  - Pure symbol heuristic – a symbol that appears with the same sign in all clauses of a sentence (all positive literals or negative ones).
    - Making these literals *true* can never make a clause *false*. Hence, pure symbols are fixed respectively.
  - Unit clause heuristic – assignment of true to unit clauses.
    - unit clause – a clause in which all literals but one have been assigned false.
    - unit propagation – assigning one unit clause creates another causing a cascade of forced assignments.

7.6 Effective Propositional Model Checking

- Tricks to scale up to large SAT problems:
  - Component Analysis (and working on each component separately)
  - Variable and value ordering (choosing the variable that appears most often in remaining clauses)
  - Intelligent backtracking (backing up all the way to the relevant conflict)
  - Random restarts (reduces the variance on the time to solution)
  - Clever indexing (with dynamic indexing structures).

7.6 Effective Propositional Model Checking

- WalkSAT – a local search algorithm based on the idea of a random walk.
  - Initial assignment is chosen randomly.
  - Repeat until satisfied or “exhausted”.
  - A *min-conflicts* heuristic (as with CSPs) is used to minimize the number of unsatisfied clauses.
  - A random walk step chooses the symbol to flip.
- If a satisfying assignment exists, it will be found, eventually.
- WalkSAT can not guarantee a sentence is unsatisfiable except with high probability.

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

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Hard Satisfiablility

- Let *m* be the number of clauses and *n* be the number of symbols.
- The ratio *m*/*n* is indicative of the difficulty of the problem.
- The probability for satisfiability drops sharply around *m*/*n* = 4.3.
- underconstrained – relatively small *m/n* thus making the expected number of satisfying assignments high.
- overconstrained – relatively high *m/n* thus making the expected number of satisfying assignments low.
- critical point – value of *m/n* such that the problem is nearly satisfiable and nearly unsatisfiable. Thus, the most difficult cases for satisfiability algorithms