Knowledge and Reasoning

- Logical Agents

Outline

- Knowledge-based agents
- Wumpus world
- Logic in general - models and entailment
- Propositional (Boolean) logic
  - Syntax and Semantics
  - Reasoning
    - Resolution
    - Forward/backward chaining
    - DPLL and local search

2

Knowledge-Based Agents

- Intelligent agents need knowledge about the world in order to reach good decisions
- The agent must be able to:
  - Represent states, actions, etc.
  - Incorporate new percepts
  - Update internal representations of the world
  - Deduce hidden properties of the world
  - Deduce appropriate actions
- Knowledge representation and reasoning

3

Generic K-Based Agent

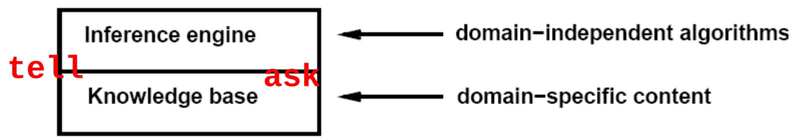
```
function KB-AGENT(percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   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
```

4

Knowledge Base

Knowledge Base (KB): set of *sentences* represented in a *knowledge representation language* and represent assertions about the world.



Tell: add new sentences to the KB

Ask: query the KB

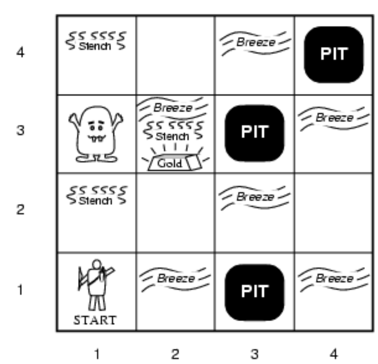
Inference rule: when one ASKs questions of the KB, the answer should *follow* from what has been TELLed to the KB previously.

5

Wumpus World PEAS description

The goal is to find the gold and bring it back to the start as quickly as possible, without getting killed

- Performance measure
  - gold +1000, death -1000
  - -1 per step, -10 for using the arrow
- Environment
  - nxn grid of rooms
  - The agent dies if it enters a square containing a pit or a live wumpus
  - 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
  - When killed, wumpus gives out a scream
  - Shooting uses up the only arrow
  - Grabbing picks up gold if in same square
  - When an agent walks into a wall, it will perceive a bump
- Sensors: Stench, Breeze, Glitter, Bump, Scream
- Actuators: Left turn, Right turn, Forward, Grab, Shoot



6

Wumpus World Characterization

- Observable?
- Deterministic?
- Static?
- Discrete?
- Single-agent?

7

Wumpus World Characterization

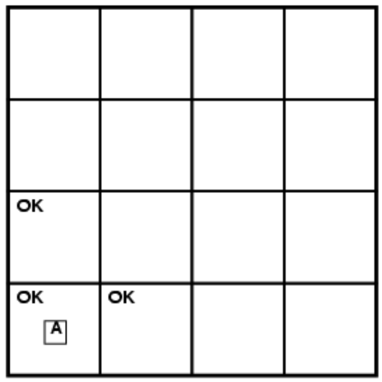
- Observable? Partially, only local perception
- Deterministic? Yes, outcome exactly specified
- Static? Yes, Wumpus and pits do not move
- Discrete? Yes
- Single-agent? Yes, Wumpus is essentially a natural feature.

8

Exploring a wumpus world

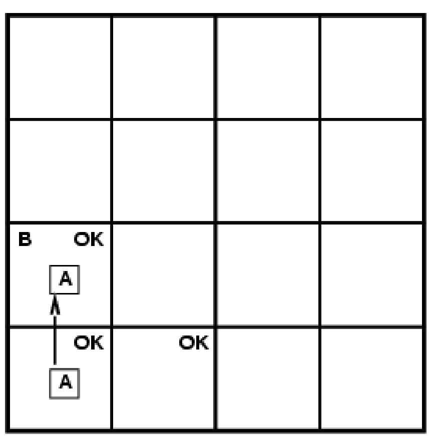
Initial KB: the rules of the environment

Percepts at (1,1): none



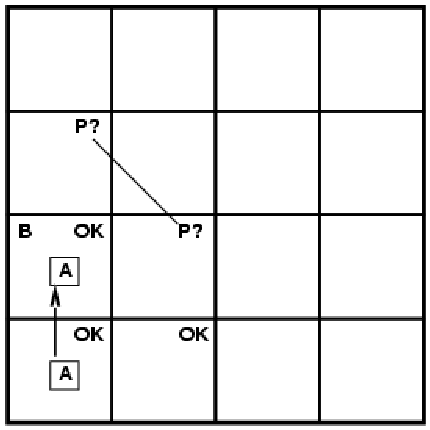
9

Exploring a wumpus world

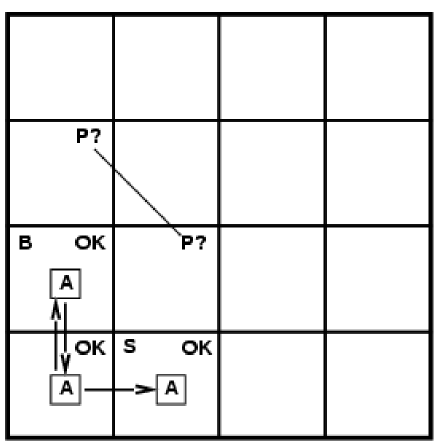


10

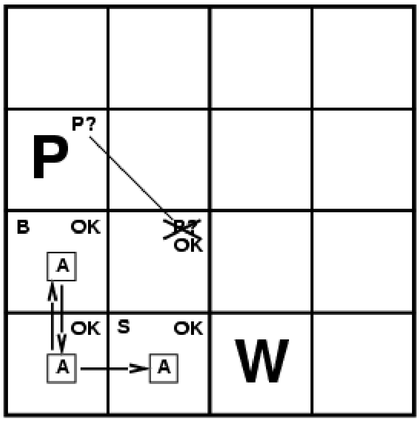
Exploring a wumpus world



Exploring a wumpus world

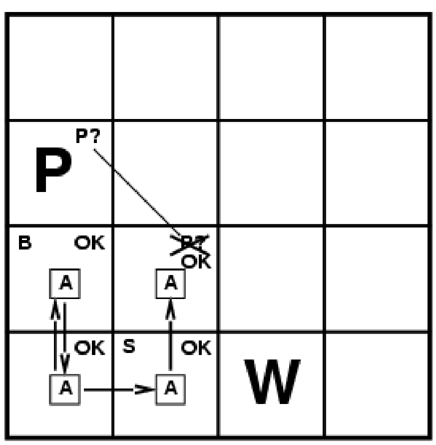


Exploring a wumpus world



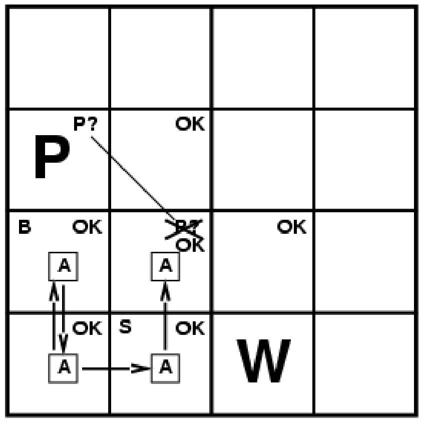
13

Exploring a wumpus world



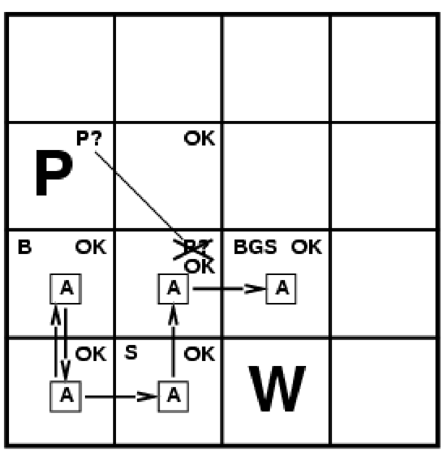
14

Exploring a wumpus world



15

Exploring a wumpus world



16

What is a logic?

- A formal language for representing information such that conclusions can be drawn
  - **Syntax** – what expressions are legal (well-formed *sentences*)
  - **Semantics** – define the “meaning" of sentences
    - Defines the truth of each sentence with respect to each possible world.

- E.g the language of arithmetic
  - x+2 >= y is a sentence, x2+y> is not a sentence
  - x+2 >= y is true in a world where x=7 and y =1
  - x+2 >= y is false in a world where x=0 and y =6

17

What is a logic?

- Semantics in logic
  - Defines the truth of each sentence with respect to each possible world.

- A **model (possible world)** fixes the truth or falsehood of every sentence
  - we know every relevant aspects of the world
  - mathematical abstractions of real worlds with respect to which truth can be evaluated

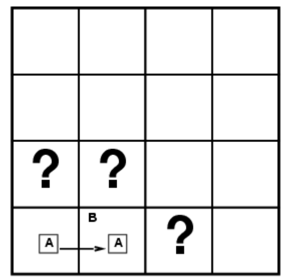
- E.g the language of arithmetic
  - Possible models are all possible assignments of numbers to x and y
  - x+2 >= y is true in a world where x=7 and y =1
  - x+2 >= y is false in a world where x=0 and y =6

18

The wumpus world

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

Consider possible models for *KB* assuming interested in whether 3 adjacent squares contain pits

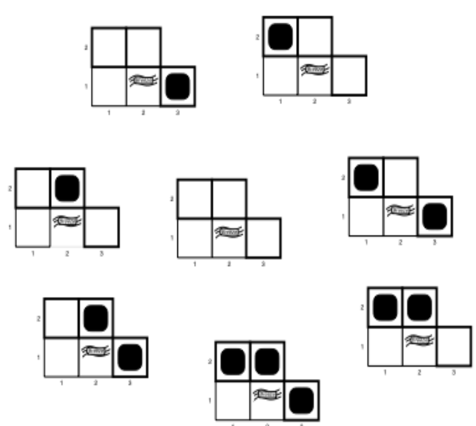


P_{12}, P_{22}, P_{31}

3 Boolean choices $\Rightarrow$ 8 possible models

19

Wumpus world model



20

Knowledge and Model

- We say *m* is a model of a sentence α if α is true in *m (*aka m *satisfies* α; α is satisfied by m; α holds at m)
- ***M(α)*** is the set of all models of α
- A *knowledge base (KB)* is a set of sentences -- real world is one of the model in M(KB)
- The key concept is to view one's knowledge as a set of possible worlds.
- When someone communicates sentence α to us, they are telling us that only worlds in *M*(α) are possible, and every world outside *M*(α) is impossible. Therefore, our state of knowledge in this case is *M*(α)
- In a complete state of ignorance, every world is a possibility and, hence, our knowledge consists of the set of all worlds. As we know more, some of these worlds are deemed as impossible and the set of possible worlds starts to shrink

Logical Reasoning

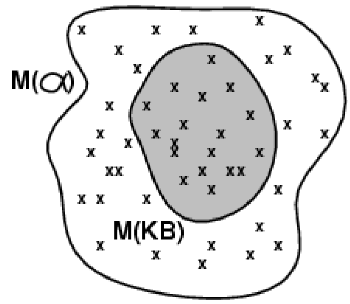
- Entailment means that one thing follows logically from another:

$$\beta \models \alpha$$

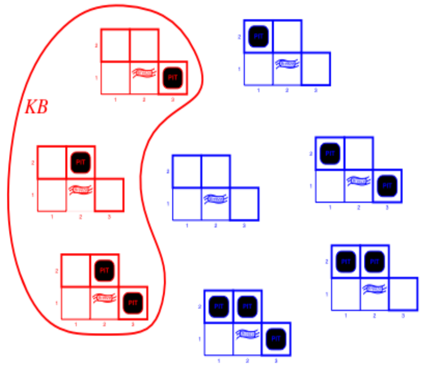
- The sentence β entails the sentence α if and only if α is true in all worlds where β is true
  - Entailment is a relationship between sentences (i.e., syntax) that is based on semantics
- Conclusions we draw from KB are those that follow from KB or entailed by KB

Entailment

- KB $\models$ α iff $M(KB) \subseteq M(\alpha)$

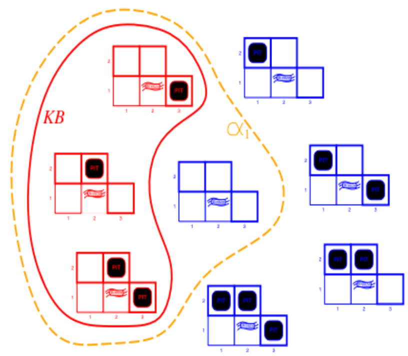


Wumpus world model



KB = wumpus-world rules + observations

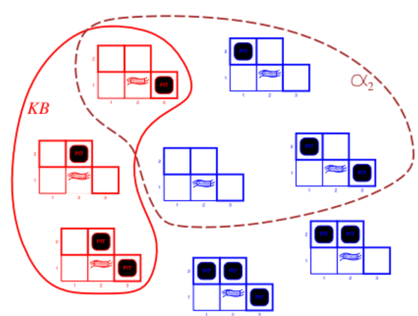
Wumpus world model



KB = wumpus-world rules + observations

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

Wumpus world model



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

Logical inference

- Logic inference: how to decide whether KB $\models \alpha$?
- Logic inference: KB $\vdash_i \alpha$ = sentence α can be derived from KB by procedure i
  - Model checking (see wumpus example): enumerate all possible models and check whether α is true.
- If an algorithm only derives entailed sentences it is called *sound* or *truth preserving*.
  - Otherwise it just makes things up.
  - *i is **sound** if whenever KB* $|-_i \alpha$ *it is also true that KB* $|= \alpha$
- Completeness : the algorithm can derive any sentence that is entailed.
  - *i is **complete** if whenever KB* $|= \alpha$ *it is also true that KB*$|-_i \alpha$

27

Propositional logic: Syntax

- *Propositional logic* aka *Boolean logic* is the simplest logic – illustrates basic ideas
- The *proposition symbols* P_1, P_2 etc, True, False are sentences - ***atomic sentences***
- Complex sentences can be constructed from simpler sentences using ***logical connectives***
  - 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)

  A ***literal***: an atomic sentence or a negated atomic sentence
  Parentheses are used to avoid ambiguity

28

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

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

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

29

Truth tables for connectives

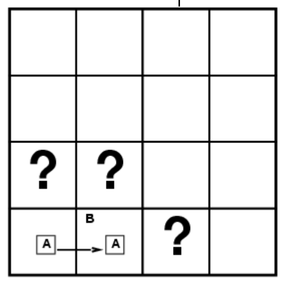
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

30

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

  "A square is breezy **if and only if** there is an adjacent pit"
  How about $P_{21} \Rightarrow B_{11} \wedge B_{22} \wedge B_{31}$?

How to decide whether KB entails $\alpha = \neg P_{1,2}$?

Truth tables for inference

Enumerate the models and check that α is true in every model in which *KB* is true.

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*false*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*false*	*true*	*true*	<u>*true*</u>	<u>*true*</u>
false	*true*	*false*	*false*	*true*	*false*	*false*	*false*	*true*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*false*

Inference by model checking

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

```
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))
```

- For n symbols, time complexity is $O(2^n)$, space complexity is $O(n)$.
- Not efficient

Inference rules in PL

Theorem proving: decide entailment using inference rules

- Modus Ponens $$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$
- And-elimination: from a conjunction any conjuncts can be inferred: $$\frac{\alpha \wedge \beta}{\alpha}$$
- All logical equivalences of next slide can be used as inference rules. $$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

Logical equivalence

- Two sentences are *logically equivalent* iff true in same set of models, or α ≡ ß iff α |= ß and ß |= α.

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

35

Example

- Assume KB:

$$\neg P_{1,1},\ B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}),\quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1}),\quad \neg B_{1,1},\quad B_{2,1}$$

- How can we prove $\neg P_{1,2}$?

$R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	**Biconditional elim.**
$R_7 : (P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$	**And elim.**
$R_8 : \neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$	**Contraposition**
$R_9 : \neg(P_{1,2} \vee P_{2,1})$	**Modus Ponens**
$R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$	**De Morgan's rule**

36

Validity and satisfiability

- A sentence is *valid* if it is true in all models,
  - 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*:
  - $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- A sentence is *satisfiable* if it is true in *some* model
  - e.g., $A \vee B$, C
- A sentence is *unsatisfiable* if it is true in *no* models
  - e.g., $A \wedge \neg A$
- Satisfiability is connected to inference via the following:
  - $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable
  - proof by contradiction/refutation
- The problem of determining entailment is reduced to that of determining the satisfiability of a sentence

Proof methods

- 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 application
    - Can be cast as a search problem: use of (complete) inference rules as actions
    - Typically require transformation of sentences into a normal form
  - Model checking
    - truth table enumeration
    - Checking satisfiability → a CSP
      - improved backtracking (DPLL)
      - heuristic local search in model space (sound but incomplete)

Resolution

A sound and complete inference rule

- A ***clause*** is a disjunction of literals

Unit Resolution Inference Rule, each l is a literal, l_i and m are **complementary literals**

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

Full **Resolution** Rule is a generalization of this rule, l_i and m_j are complementary literals

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

For clauses of length two:

$$\frac{\ell_1 \vee \ell_2, \qquad \neg \ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Factoring: the resulting clause keeps only one copy of each literal

CNF

- Every sentence is logically equivalent to a conjunction of disjunctions of literals
- CNF (Conjunctive normal form)
  - Conjunction of disjunctions of literals

Conversion to CNF

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

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
  - $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg \alpha \vee \beta$.
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
- Move $\neg$ inwards using de Morgan's rules:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
- Apply distributivity law ($\vee$ over $\wedge$) and flatten:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

41

Resolution algorithm

- Proof by contradiction, i.e., show $KB \wedge \neg \alpha$ unsatisfiable
- First $KB \wedge \neg \alpha$ is converted into CNF
- Then apply resolution rule to resulting clauses.
  - Every pair that contains complementary literals is resolved, the new clause is added to the set
- The process continues until:
  - There are no new clauses that can be added
    - Hence KB **does not** entail α
  - Two clauses resolve to derive the empty clause (resolve P with not P)
    - Hence KB **does** entail α

42

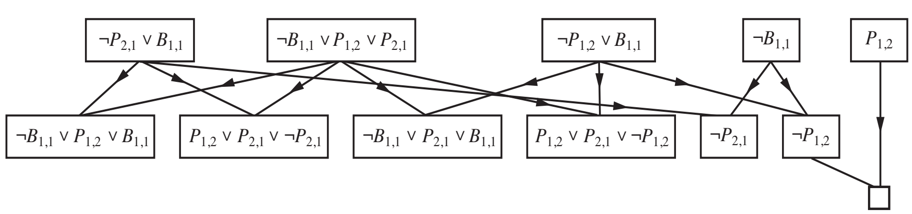
Resolution algorithm

- 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 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
```


Resolution example

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



- Any clause in which two complementary literals appear can be discarded

Completeness of Resolution

- *Ground resolution theorem*: if a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause
- Resolution is complete in the sense that it can always be used to either confirm or refute a sentence (it can not be used to enumerate all possible true sentences)
- In general, exponential complexity

45

Definite clauses

- The completeness of resolution makes it a very important inference method
- Real-world knowledge bases often contains only a restricted form of clauses:
  - Definite clauses = disjunction of literals with exactly one positive literal

$$\left(\neg L_{1,1} \vee \neg Breeze \vee B_{1,1}\right) \rightarrow \left(L_{1,1} \wedge Breeze\right) \Rightarrow B_{1,1}$$

- Definite clause =
  - proposition symbol (called a *fact*); or
  - (conjunction of symbols) $\Rightarrow$ symbol

46

Forward and backward chaining

- KB = conjunction of definite clauses
  - E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

- Modus Ponens: complete for KBs of definite clauses

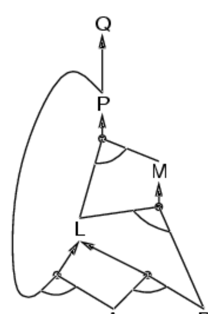
$$\frac{\alpha_1, \ldots ,\alpha_n, \qquad \alpha_1 \wedge \ldots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

- Deciding entailment (of a proposition symbol) can be done in a time linear in size of the knowledge base:
  - forward chaining
  - backward chaining

AND-OR graph

- KB can be represented as an AND-OR graph, multiple links joined by an arc indicate a conjunction

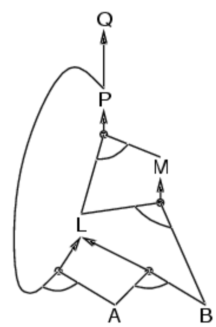
$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ A & \\ B & \end{aligned}$$



Forward chaining

- Idea: fire any rule whose premises are satisfied in the *KB*,
  - add its conclusion to the *KB*, until query is found or no inferences can be made

$$
\begin{aligned}
P &\Rightarrow Q \\
L \wedge M &\Rightarrow P \\
B \wedge L &\Rightarrow M \\
A \wedge P &\Rightarrow L \\
A \wedge B &\Rightarrow L \\
A & \\
B &
\end{aligned}
$$



49

Forward chaining algorithm

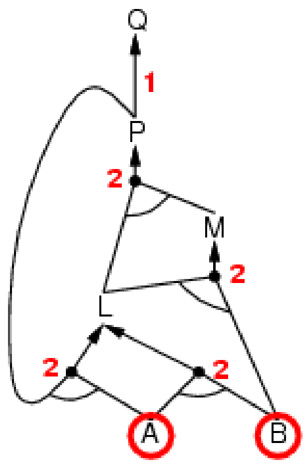
- Forward chaining is sound and complete

```
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
```

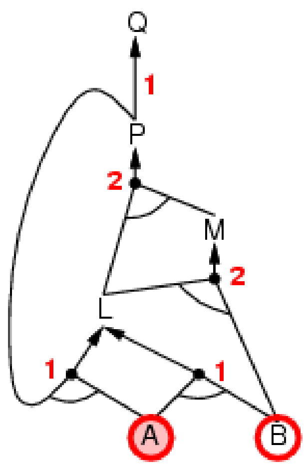
50

Forward chaining example



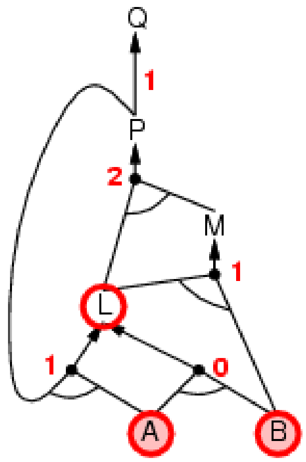
51

Forward chaining example

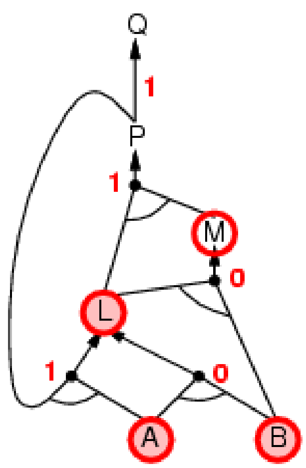


52

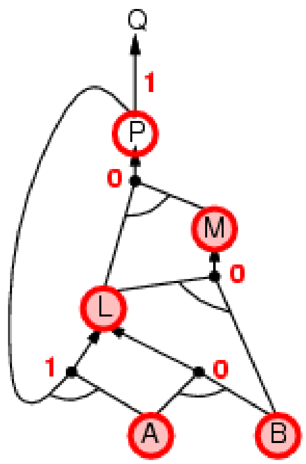
Forward chaining example



Forward chaining example

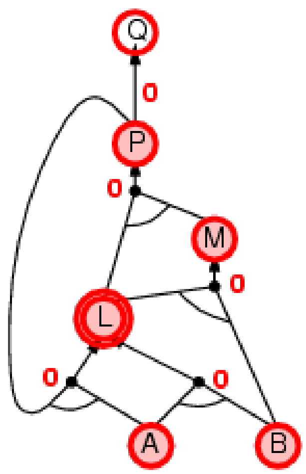


Forward chaining example



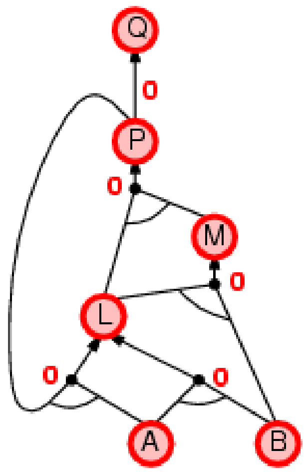
55

Forward chaining example



56

Forward chaining example



57

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*

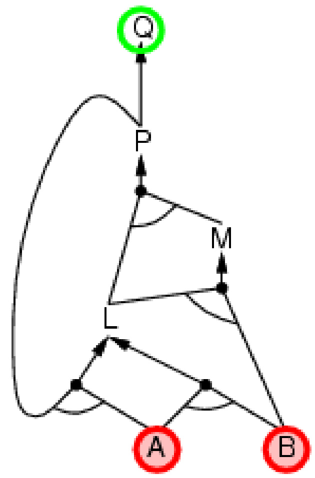
Can be implemented as a recursive depth-first search

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

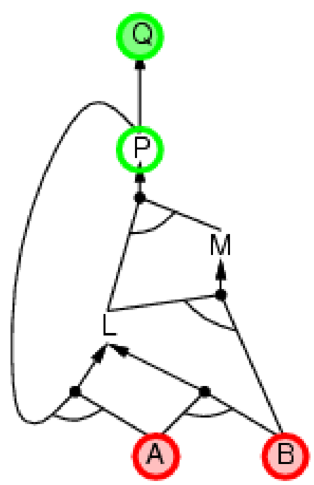
58

Backward chaining example



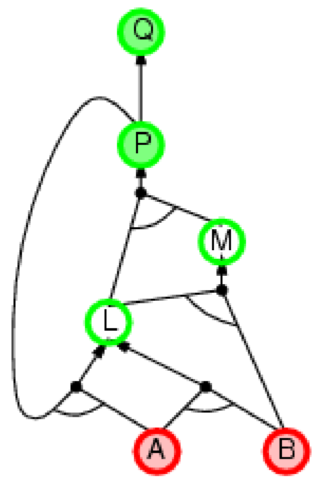
59

Backward chaining example



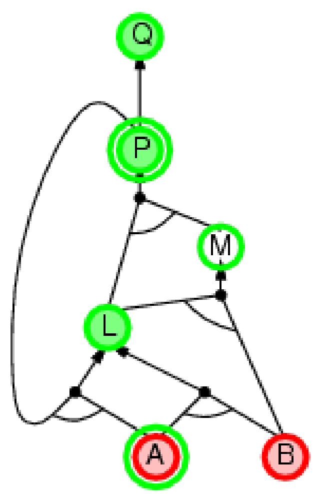
60

Backward chaining example



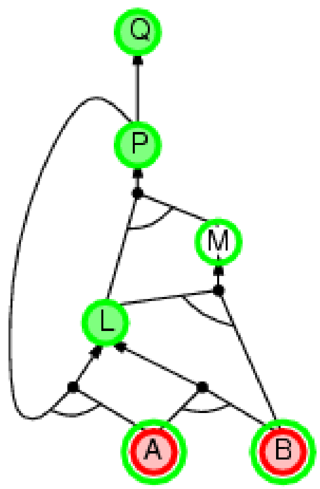
61

Backward chaining example



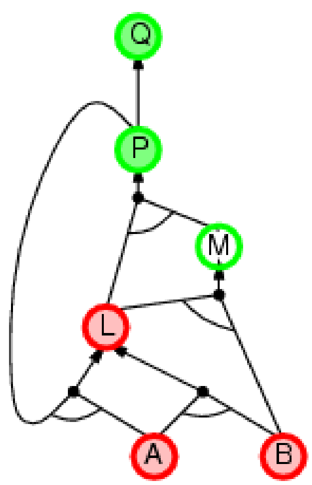
62

Backward chaining example



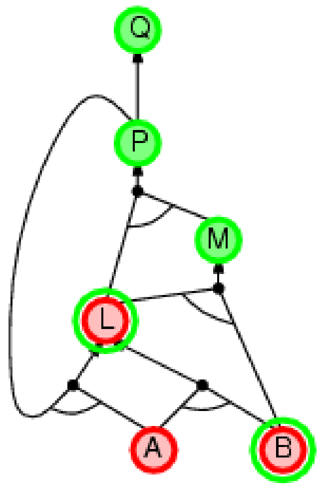
63

Backward chaining example



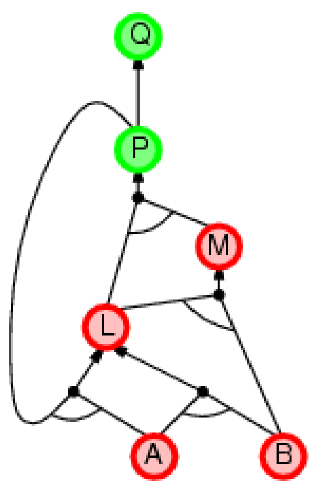
64

Backward chaining example



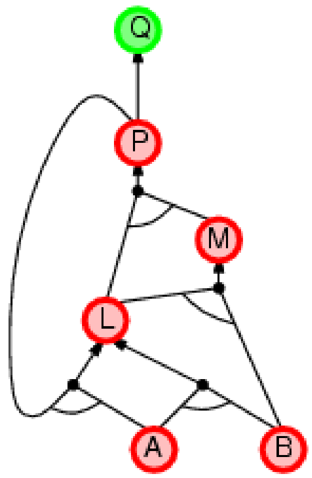
65

Backward chaining example



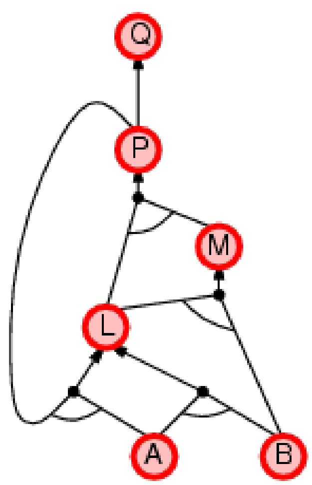
66

Backward chaining example



67

Backward chaining example



68

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?
- Complexity of BC can be much less than linear in size of KB -- it touches only relevant facts.

69

Effective Propositional Model Checking

- Two families of efficient algorithms for propositional inference
- Use algorithms for checking satisfiability -- can be cast as a CSP
  - $KB \models \alpha$ if and only if ($KB \wedge \neg\alpha$) is unsatisfiable
- Important algorithms in their own right as many computer science problems can be reduced to satisfiability problem
- Complete backtracking search algorithms
  - DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - `WalkSAT` algorithm

70

The DPLL algorithm

- Determine if an input propositional logic sentence (*in CNF*) is satisfiable.
- A recursive depth-first enumeration of possible models
- 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. E.g. $(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$
  2. Pure symbol heuristic
     Pure symbol: always appears with the same "sign" in all clauses.
     e.g., In the three clauses $(A \lor \neg B)$, $(\neg B \lor \neg C)$, $(C \lor A)$, A and B are pure, C is impure.
     Assign a pure symbol so that their literals are true.
  3. Unit clause heuristic
     Unit clause: only one literal in the clause or only one literal which has not yet received a value.
     The only literal in a unit clause must be true. First do this assignments before continuing with the rest (unit propagation!).

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, [P = value|model])
   P, value ← FIND-UNIT-CLAUSE(clauses, model)
   if P is non-null then return DPLL(clauses, symbols–P, [P = value|model])
   P ← FIRST(symbols); rest ← REST(symbols)
   return DPLL(clauses, rest, [P = true|model]) or
          DPLL(clauses, rest, [P = false|model])
```

Many other heuristics for improving performance.
There exists efficient implementation of DPLL.

The Local search algorithms

- Incomplete, local search algorithms can be applied
- Steps are taken in the space of complete assignments, flipping the truth value of one variable at a time.
- Evaluation function: The min-conflict heuristic of minimizing the number of unsatisfied clauses.
- Balance between greediness and randomness.
  - To avoid local minima

73

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
           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
```

74

The WalkSAT algorithm

- One of the simplest and most effective local search algorithms
- If it returns failure, we cannot tell whether the sentence is unsatisfiable
- Most useful when we expect a solution to exist
- Local search cannot prove entailment

75

KB in the wumpus world

A knowledge base about the physics of the Wumpus-world:

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

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$ for every square

$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$ for every square

$W_{1,1} \vee W_{1,2} \vee \ldots \vee W_{4,4}$ (at least one wumpus)

$\neg W_{1,1} \vee \neg W_{1,2}$ (at most one wumpus)

$\neg W_{1,1} \vee \neg W_{1,3}$

…

Update initial KB with percepts

Logical model of the effects of actions

Construct a wumpus world agent (Chapter 7.7)

76

Expressiveness limitation of propositional logic

- KB contains "physics" sentences for every single square

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$ for every square

- Better to have just two sentences for breezes and stenches for all squares.
  - Impossible for propositional logic.

77

Expressiveness limitation of propositional logic

- We should be able to reason about location and time automatically using logical inference
  - For every time *t* and every location [*x*,*y*],

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

  - PROBLEM: Rapid proliferation of clauses.
- Propositional logic lacks the expressive power to deal concisely with time and space

78

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
- Resolution is complete for propositional logic
  Forward, backward chaining are linear-time, complete for definite clauses
- Efficient model-checking inference algorithms
- Propositional logic lacks expressive power