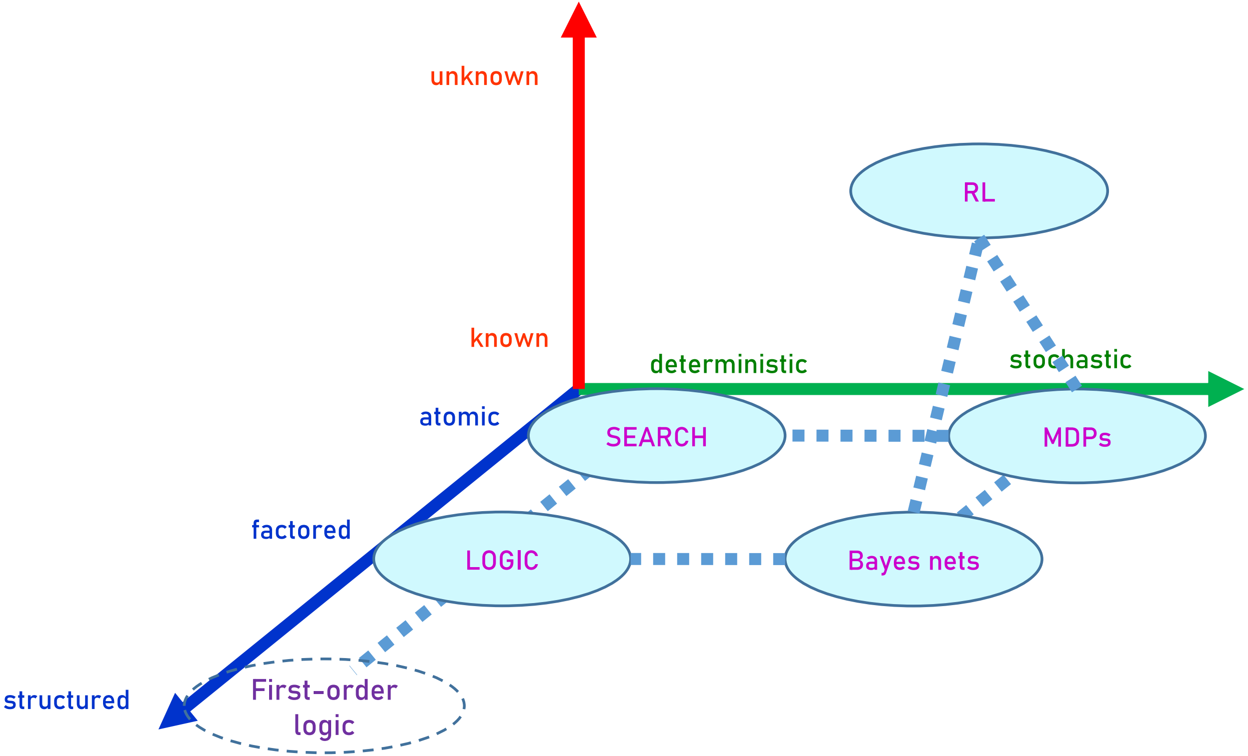
Artificial Intelligence

7. Logical Agent, Propositional Logic

Shashi Prabh

School of Engineering and Applied Science
Ahmedabad University

Contents



unknown
known
deterministic
stochastic
atomic
factored
structured
RL
SEARCH
MDPs
LOGIC
Bayes nets
First-order logic

Contents

Goal: Design knowledge-based agents that use reasoning over an internal representation of knowledge to decide the actions.

Topics

- Basic concepts of knowledge, logic, reasoning
- Knowledge-Based Agents
- Propositional Logic : syntax and semantics
- Inference in Propositional Logic
  - Inference by model checking
  - Inference by theorem proving
- Wumpus world agent using propositional logic

Knowledge-Based (KB) Agents

- Knowledge can be used to make good decisions, i.e., intelligent behavior
  - In early 1960s, McCarthy introduced the idea of using logic to determine actions in his paper "Programs with Common Sense"
  - Rational agents can be defined by the knowledge they possess rather than the programs they run [Newell, "The knowledge level," 1982]
- Problem solving agents are inflexible. A good path finding agent is useful for finding a path but not generalizable to any other task
  - They don't know general facts: the sun sets in the west (strong glaze if driving west), mileage depends on speed, …
  - The only choice for representing what it knows in a partially observable environment is to list all possible concrete states

Knowledge

- A KB agent uses logic to represent knowledge
  - These agents can combine and recombine information
- Knowledge is contained in agents in the form of sentences in a knowledge representation language that are stored in a knowledge base
  - Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent
  - Tell : add new sentence (what it needs to know) to the KB
    - Can also learn the knowledge
  - Ask : the agent queries the KB what to do
    - Answers must be consistent with the KB

Knowledge

- A KB agent is composed of a knowledge base and an inference mechanism
- Agents can be viewed at the knowledge level i.e., what they know, regardless of how implemented (Newell)
- A single inference algorithm can answer any answerable question

Knowledge base	Domain-specific facts
Inference engine	Generic code

KB Agents

- Agents acquire knowledge through perception, learning, language
  - Knowledge of the effects of actions (“transition model”)
  - Knowledge of how the world affects sensors (“sensor model”)
  - Knowledge of the current state of the world
- Can keep track of a partially observable world
- Can formulate plans to achieve goals

KB Agents

```
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
```

Logic

- Logic has its origins in ancient Greek philosophy and mathematics.
  - Plato discussed the syntactic structure of sentences, their truth and falsity, their meaning, and the validity of logical arguments.
  - The first known systematic study of logic was Aristotle's Organon
- KB consists of sentences formed according to the syntax
  - Syntax: What sentences are valid?
    - Example: x + y = 3

Logic

- Semantics determines the truth of a sentence (only true or false) in a possible world or model
  - What do the sentences mean?
  - What are the possible worlds?
  - Which sentences are true in which worlds? (i.e., definition of truth)
- x + y = 3 and y + x = 3 have different syntax but the same semantics

Different kinds of logic

- Propositional logic
  - Syntax: $P \vee (\neg Q \wedge R)$; $X_1 \Leftrightarrow (\text{Raining} \Rightarrow \neg\text{Sunny})$
  - Possible world: {P=true, Q=true, R=false, S=true} or 1101
  - Semantics: $\alpha \wedge \beta$ is true in a world iff is α true and β is true (etc.)

- First-order logic
  - Syntax: $\forall x\ \exists y\ P(x, y) \wedge \neg Q(Joe, f(x)) \Rightarrow f(x)=f(y)$
  - Possible world: Objects o_1, o_2, o_3; P holds for <o_1,o_2>; Q holds for <o_3>; $f(o_1)=o_1$; $Joe=o_3$; etc.
  - Semantics: $\phi(\sigma)$ is true in a world if $\sigma = o_j$ and ϕ holds for o_j; etc.

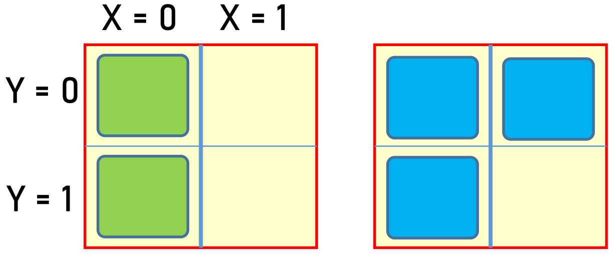
Different kinds of logic

- Relational databases
  - Syntax: ground relational sentences, e.g., Sibling(Ali, Bo)
  - Possible worlds: (typed) objects and (typed) relations
  - Semantics: sentences in the DB are true, everything else is false
    - Cannot express disjunction, implication, universals, etc.
    - Query language (SQL etc.) typically some variant of first-order logic
    - Often augmented by first-order rule languages, e.g., Datalog
  - Knowledge graphs (roughly: relational DB + ontology of types and relations)
    - Google Knowledge Graph: 5 billion entities, 500 billion facts, >30% of queries
    - Facebook network: 2.8 billion people, trillions of posts, maybe quadrillions of facts

Inference

- **Satisfaction:** If a sentence α is true in model m, we say that m satisfies α or sometimes m is a model of α.
  - We use the notation $M(\alpha)$ to mean the set of all models of α.
- **Entailment:** Refers to a sentence following logically from another
  - $\alpha \models \beta$ (or, $\alpha \vdash \beta$) denotes "α entails β" or "β follows from α"
- $\alpha \models \beta$ iff in every model in which α is true, β is also true
  - $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$.
  - α is a stronger assertion than β
- Example: $X = 0 \models XY = 0$

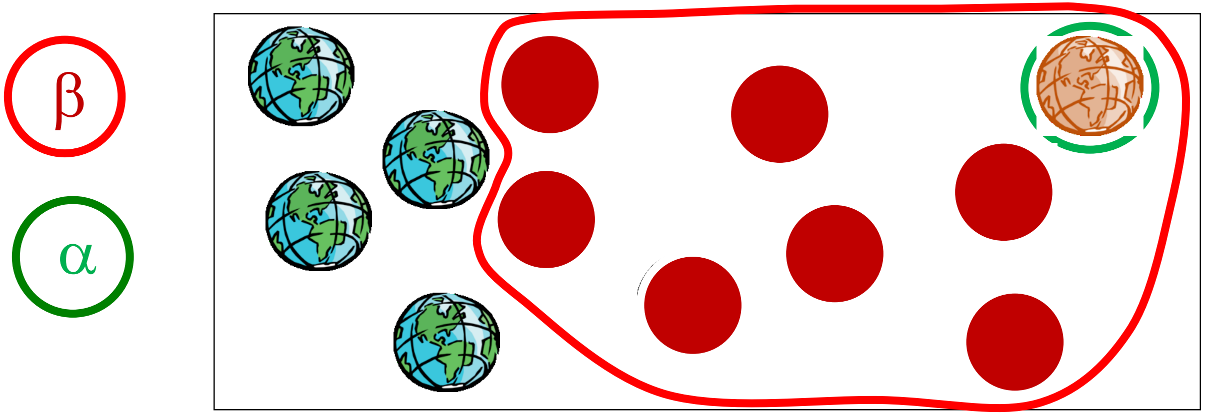
  $XY=0 \not\models X=0$



X = 0   X Y = 0

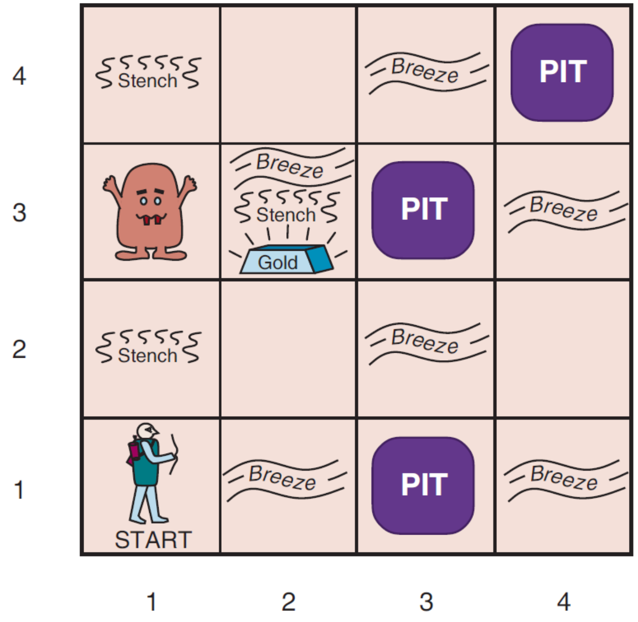
Inference: entailment

- Recall: $\alpha \models \beta$ means that the α-worlds are a subset of the β-worlds
  - Models(α) $\subseteq$ Models(β)
- Example: Given $\alpha = \neg Q \wedge R \wedge S \wedge W$, $\beta = \neg Q$
  - Then $\alpha \models \beta$

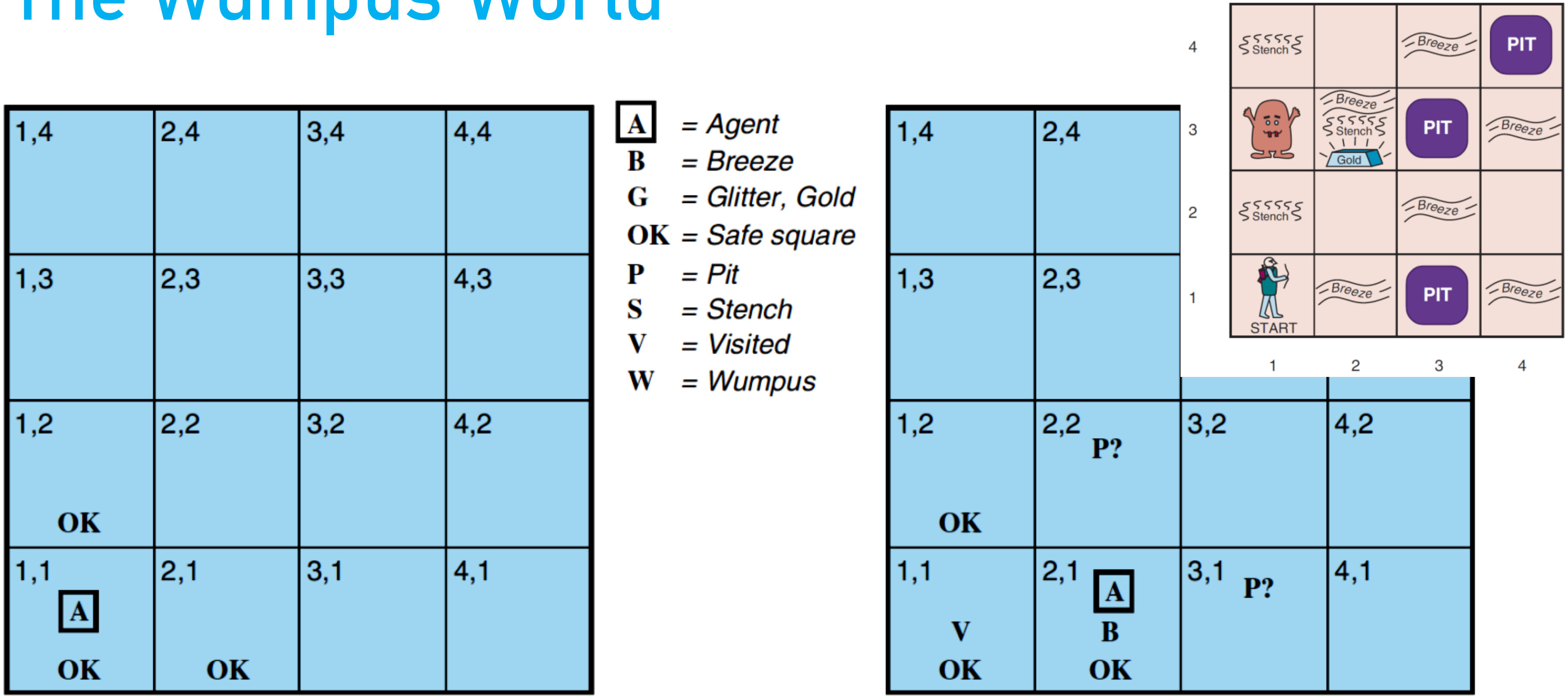


The Wumpus World

- Partially observable environment
- Sensors
    - Breeze, Stench, Glitter, Bump, Scream
- Percepts are 5-tuple: if there is a stench and a breeze, but no glitter, bump, or scream
    - [Stench, Breeze, None, None, None]
- Performace measure
    - +1000 for exiting the cave with the gold
    - –1000 for falling into a pit or being eaten
    - –1 for each action taken
    - –10 for using up the arrow



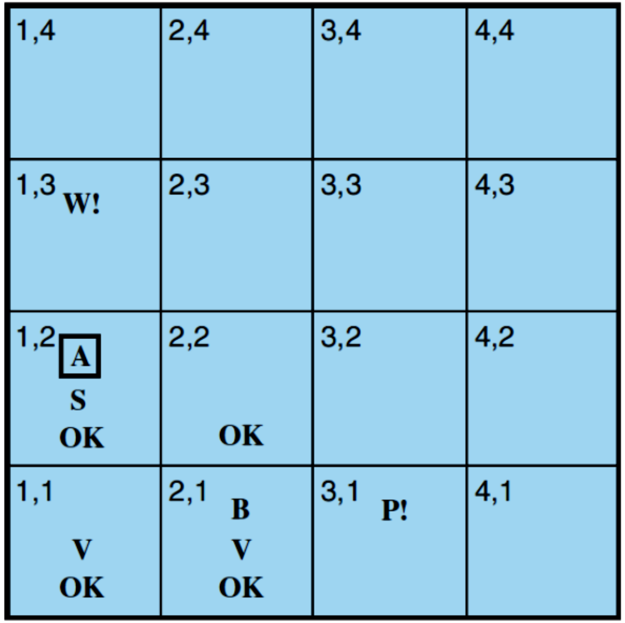
The Wumpus World



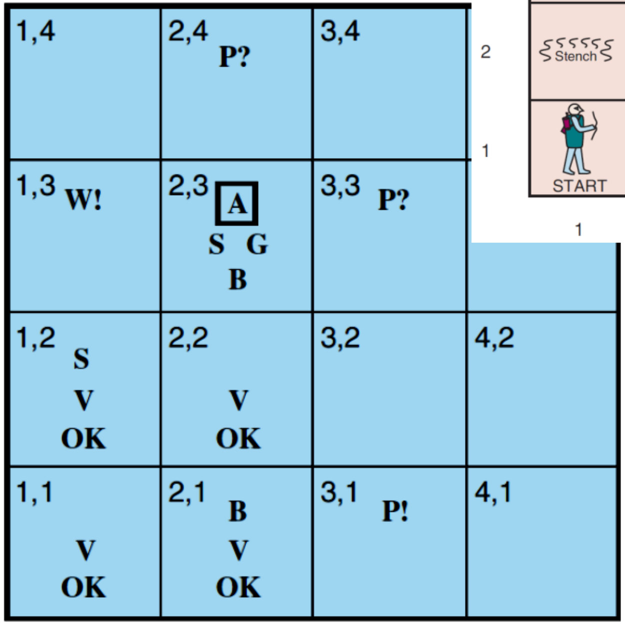
(a)

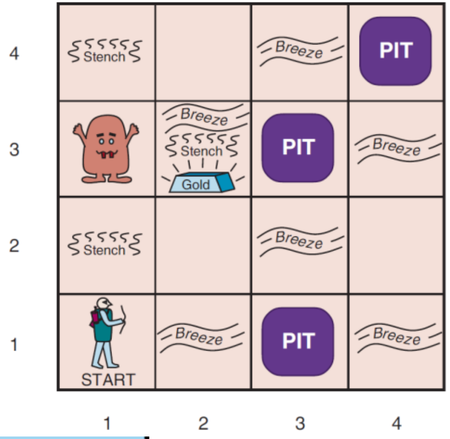
(b)

The Wumpus World



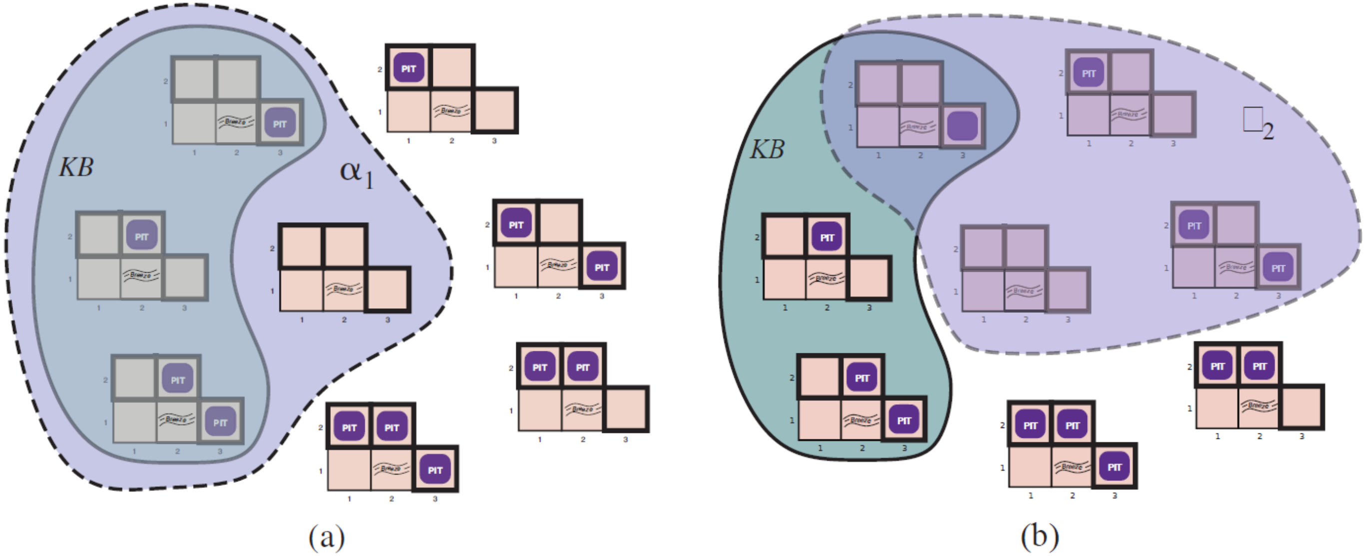
A = Agent
B = Breeze
G = Glitter, Gold
OK = Safe square
P = Pit
S = Stench
V = Visited
W = Wumpus





Logic

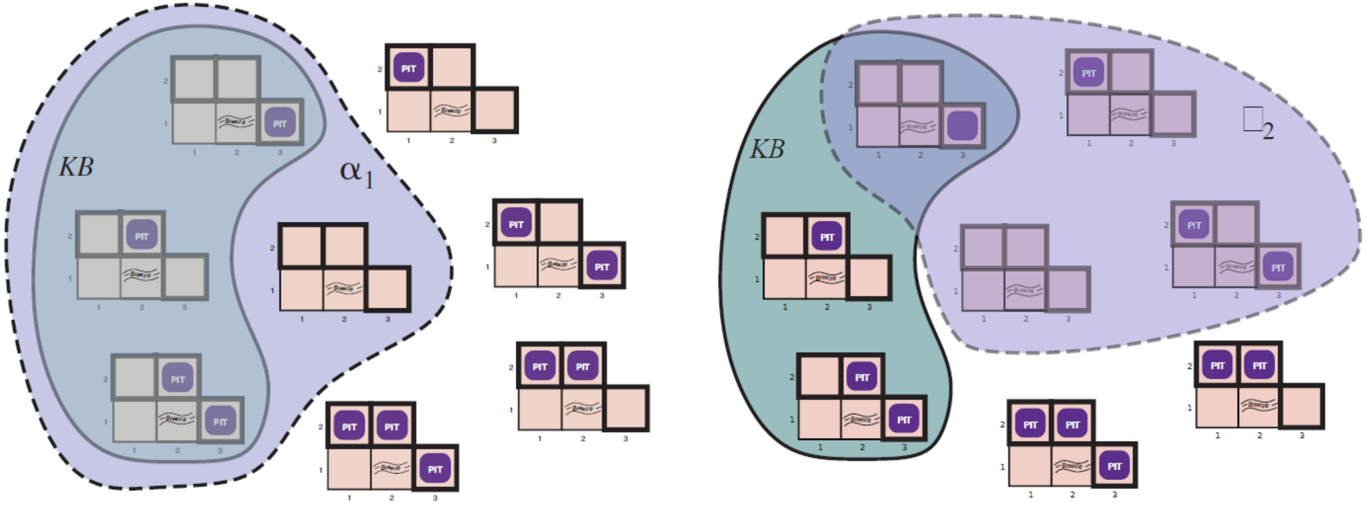
- Wumpus world example: the agent has visited [1, 1] and [2, 1]
  - 8 possible models for pits in the neighboring squares
  - α_1 = "No pit in [1, 2]"
  - α_2 = "No pit in [2,2]"



Logic

- Wumpus world example: the agent has visited [1, 1] and [2, 1]
  - 8 possible models for pits in the neighboring squares
  - α_1 = "No pit in [1, 2]"
  - α_2 = "No pit in [2,2]"

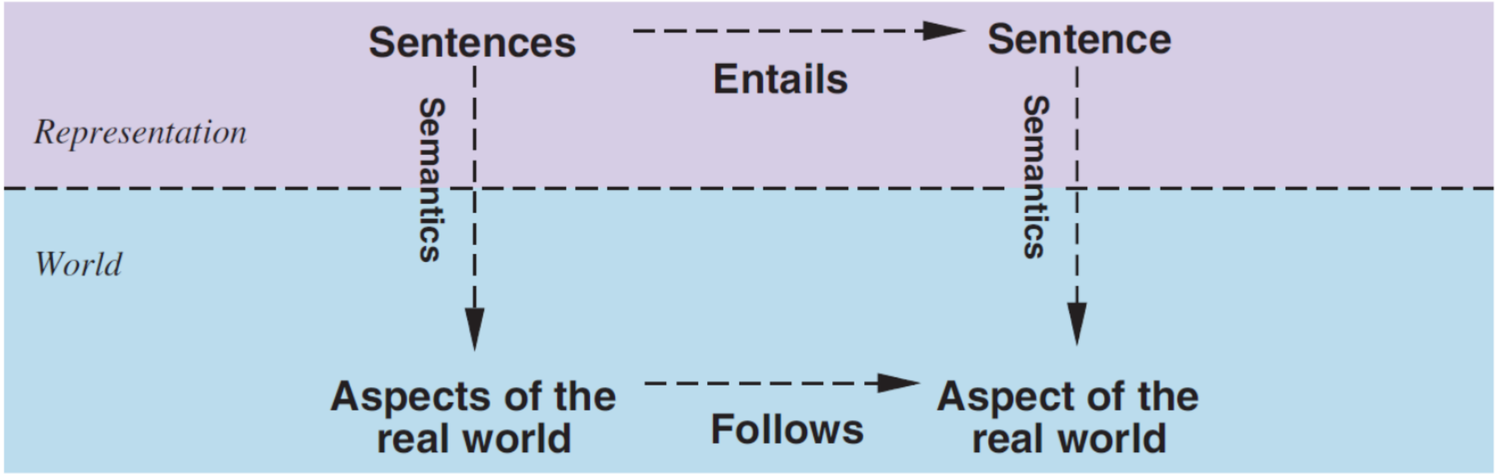
$KB \models \alpha_1, \quad KB \not\models \alpha_2$



(a) (b)

Logic

- If KB is true in the real-world, then any sentence derived from KB is also true in the real world



Propositional Logic Syntax

- Atomic sentences (Literal): single proposition symbol, e.g., True, False, $W_{1,3}$
- Complex sentences are formed form simpler sentences using logical connectives

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

Propositional logic syntax

- Given: a set of proposition symbols $\{X_1, X_2, \ldots, X_n\}$
  - we often add True and False for convenience
- X_i is a sentence
- NOT: If α is a sentence then $\neg\alpha$ is a sentence
- AND: If α and β are sentences then $\alpha \wedge \beta$ is a sentence
- OR: If α and β are sentences then $\alpha \vee \beta$ is a sentence
- Implies: If α and β are sentences then $\alpha \Rightarrow \beta$ is a sentence
- IFF: If α and β are sentences then $\alpha \Leftrightarrow \beta$ is a sentence
- And p.s. there are no other sentences!

Propositional logic semantics

- Let m be a model assigning true or false to $\{X_1, X_2, \ldots, X_n\}$
- If α is a symbol then its truth value is given in m

- $\neg\alpha$ is true in m iff α is false in m
- $\alpha \wedge \beta$ is true in m iff α is true in m and β is true in m
- $\alpha \vee \beta$ is true in m iff α is true in m or β is true in m
- $\alpha \Rightarrow \beta$ is true in m iff α is false in m or β is true in m (i.e., $\beta \vee \neg\alpha$)
- $\alpha \Leftrightarrow \beta$ is true in m iff $\alpha \Rightarrow \beta$ is true in m and $\beta \Rightarrow \alpha$ is true in m

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*

Inference: proofs

- Method 1: model-checking
  - Enumerates all possible models and checks for every possible world: if α (KB) is true, make sure that β is true too
    - $M(\alpha) \subseteq M(\beta)$; $M(KB) \subseteq M(\beta)$
  - OK for propositional logic (finitely many worlds); not easy for first-order logic

- Method 2: theorem-proving
  - Search for a sequence of proof steps (applications of inference rules) leading from α to β
  - E.g., from $P \wedge (P \Rightarrow Q)$, infer Q by Modus Ponens

Inference: proofs

- A proof is a demonstration of entailment between α and β
  - Have a set of formulas and want to check the truth of some conclusion based on the given formulas
- Sound algorithm: everything it claims to prove is in fact entailed
- Complete algorithm: everything that is entailed can be proved

Wumpus World KB

- Partially observable environment
- Symbols

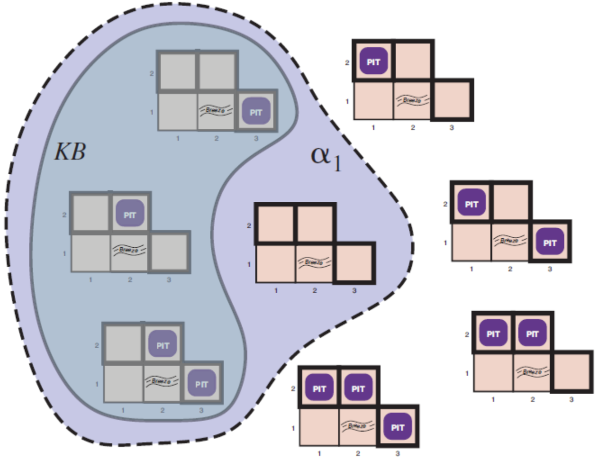
$P_{x,y}$ is true if there is a pit in $[x,y]$.
$W_{x,y}$ is true if there is a wumpus in $[x,y]$, dead or alive.
$B_{x,y}$ is true if there is a breeze in $[x,y]$.
$S_{x,y}$ is true if there is a stench in $[x,y]$.
$L_{x,y}$ is true if the agent is in location $[x,y]$.

Wumpus World KB

- Initial state R1 : $\neg P_{1,1}$
- Sensor Model - state facts about how percepts arise
  - <Percept variable (at t)> $\Leftrightarrow$ <some condition on world (at t)>
  - R2 : $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - R3 : $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
    - Note: True in all wumpus worlds
- The breeze percepts for the first two squares visited
  - R4 : $\neg B_{1,1}$
    - $L_{1,1} \wedge \text{Breeze} \Rightarrow B_{1,1}$
  - R5 : $B_{2,1}$
  - $M(KB) = \bigcap_{R_i \in KB} M(R_i)$

How many possible worlds?

- The symbols are $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$
- 7 symbols => 2^7 = 128 possible worlds (models)
  - With just 80 symbols there are $2^{80} \simeq 10^{21}$ possible models!
- KB is True in 3 of these
- And $\neg$P1,2 is True in all three
- Propositional entailment is co-NP-complete
  - Inference algorithms for propositional logic have exponential worst case complexity in the size of the input



Inference: Truth table enumeration

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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*

Propositional logic semantics in code

```
function PL-TRUE?(α,model) returns true or false
    if α is a symbol then return Lookup(α, model)
    if Op(α) = ¬ then return not(PL-TRUE?(Arg1(α),model))
    if Op(α) = ∧ then return  and(PL-TRUE?(Arg1(α),model),
                                  PL-TRUE?(Arg2(α),model))
    etc.
```

Example: $P_{1,1} \wedge (P_{2,2} \vee P_{3,1}) \rightarrow T \wedge (F \vee T) = T \wedge T = T$

Propositional theorem proving

- Recall: Theorem proving refers to searching for a sequence of proof steps (i.e., applications of inference rules) leading from α to β
- Sound algorithm: everything it derives is in fact entailed
- Complete algorithm: every that is entailed can be derived

Some reasoning tasks

- Localization with a map and local sensing:
  - Given an initial KB, plus a sequence of percepts and actions, where am I?
- Mapping with a location sensor:
  - Given an initial KB, plus a sequence of percepts and actions, what is the map?
- Simultaneous localization and mapping:
  - Given …, where am I and what is the map?
- Planning:
  - Given …, what action sequence is guaranteed to reach the goal?
- ALL OF THESE USE THE SAME KB AND THE SAME ALGORITHM!

Logical equivalences

- α and β are logically equivalent, $\alpha \equiv \beta$, if $\alpha \models \beta$ and $\beta \models \alpha$

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

Inference rules

- Chain of conclusions
- Modus ponens (mode that affirms)
  - Given:
    - (WumpusAhead ∧ WumpusAlive) ⇒ Shoot
    - (WumpusAhead ∧ WumpusAlive)
  - Then infer:
    - Shoot

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

- And-elimination

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

Both are sound but not complete. Consider { } ⇒ P ∨ ¬P

- Logical equivalence rules

Inference example

- Starting with KB containing R1 to R5, prove $\neg P_{1,2}$
- Biconditional elimination to R2:
  - R6 : (B1,1 ⇒ (P1,2 ∨ P2,1)) ∧ ((P1,2 ∨ P2,1) ⇒ B1,1) .
- And-Elimination to R6:
  - R7 : ((P1,2 ∨ P2,1) ⇒ B1,1) .

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

- Logical equivalence for contrapositives gives
  - R8 : (¬B1,1 ⇒ ¬(P1,2 ∨ P2,1)) .
- Modus Ponens with R8 and the percept R4 (i.e., ¬B1,1) gives
  - R9 : ¬(P1,2 ∨ P2,1) .

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

- Finally, De Morgan's rule gives the conclusion
  - R10 : ¬P1,2 ∧ ¬P2,1 .
- Neither [1,2] nor [2,1] contains a pit!

Conjunctive normal form (CNF)

- Every sentence can be expressed as a conjunction of clauses
- Each clause is a disjunction of literals
- Each literal is a symbol or a negated symbol
- Example: $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Conjunctive normal form (CNF)

Conversion to CNF of $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

- 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 $\neg\alpha \vee \beta$:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$.
- CNF requires $\neg$ to appear in literals. Moving $\neg$ inwards:
  - $\neg(\neg\alpha) \equiv \alpha$ (double-negation 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)

Conjunctive normal form (CNF)

- In the example, we require one application of the last rule:
  - $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$.

- Now we have a sentence containing nested ∧ and ∨ operators applied to literals.
- Apply the distributive law distributing ∨ over ∧ wherever possible:
  - $(\neg B1{,}1 \vee P1{,}2 \vee P2{,}1) \wedge (\neg P1{,}2 \vee B1{,}1) \wedge (\neg P2{,}1 \vee B1{,}1)$

Conjunctive normal form (CNF)

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

Satisfiability and entailment

- A sentence is satisfiable if it is true in at least one model
  - Called the SAT problem
  - First NP-Complete problem (Cook-Levin Theorem)
  - $(P \vee \neg Q) \wedge (\neg P \vee Q)$ is satisfiable ($P = T$, $Q = T$)
  - $P \wedge \neg P$ is unsatisfiable

- A sentence is a tautology if it is true in all models
  - E.g., $P \vee \neg P$
- α is a tautology if $\neg\alpha$ is unsatisfiable
- $\alpha \models \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable

Satisfiability and entailment

- Suppose we have a hyper-efficient SAT solver (WARNING: NP-COMPLETE). How can we use it to test entailment?
  - $\alpha \models \beta$

    iff $\alpha \Rightarrow \beta$ is true in all worlds

    iff $\neg(\alpha \Rightarrow \beta)$ is false in all worlds

    iff $\alpha \wedge \neg\beta$ is false in all worlds, i.e., unsatisfiable
- So, add the negated conclusion to what you know, test for (un)satisfiability; also known as reductio ad absurdum
- Is KB $\cup$ $\{ \neg\beta \}$ satisfiable? If no, KB $\models \beta$
- Efficient SAT solvers operate on conjunctive normal form (CNF)

Proof by Resolution (Robinson, 1965)

- Unit resolution $\dfrac{A \lor B \lor \neg C,\ \neg A \lor D}{B \lor \neg C \lor D}$
- The resolution inference rule takes sentences in CNF and negation of query: ($KB \land \neg\alpha$) is converted into CNF
- The resolution rule is applied to the resulting clauses
  - Each pair containing complementary literals is resolved
    - The new clause is added to the set if it is not already present
- Until:
  - There are no new clauses that can be added: KB does not entail α
  - Or, two clauses resolve to yield the empty clause: $KB \models \alpha$

Proof by Resolution (Robinson, 1965)

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

- Additions to KB:
  - R11 : $\neg B_{1,2}$
  - R12 : $B_{1,2} \Leftrightarrow (P_{1,1} \lor P_{2,2} \lor P_{1,3})$
- Inferences:
  - R13 : $\neg P_{2,2}$
  - R14 : $\neg P_{1,3}$

Proof by Resolution (Robinson, 1965)

- Biconditional elimination to R3 and Modus Ponens with R5:

  R15 : P1,1 ∨ P2,2 ∨ P3,1 .

- Unit resolutions:
- The literal ¬P2,2 resolves with P2,2 to give the resolvent

  R16 : P1,1 ∨ P3,1 .

- Similarly, the literal ¬P1,1 in R1 resolves with P1,1 in R16 to give

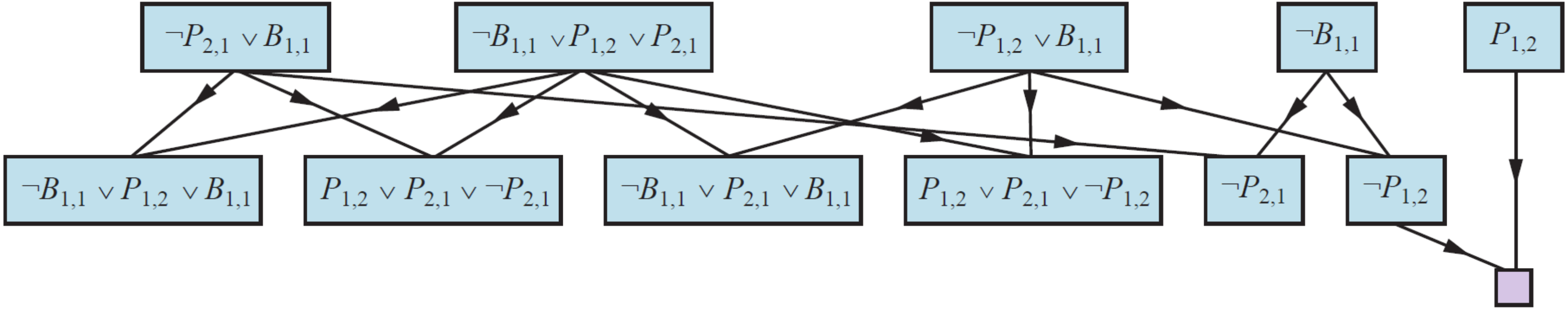
  R17 : P3,1

Proof by Resolution (Robinson, 1965)

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 $KB \wedge \neg\alpha$
  new ← { }
  while *true* **do**
    for each pair of clauses C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Proof by Resolution (Robinson, 1965)



- Resolution is complete for propositional logic
- More powerful than modus ponens
- Exponential time in the worst case

Simple theorem proving: Forward chaining

- Forward chaining applies Modus Ponens to generate new facts:
  - Given $X_1 \wedge X_2 \wedge \ldots X_n \Rightarrow Y$ and $X_1, X_2, \ldots, X_n$, infer Y
  - Given $L_{1,1} \wedge Breeze \Rightarrow B_{1,1}$, $L_{1,1}$ and Breeze, infer $B_{1,1}$
- Forward chaining keeps applying this rule, adding new facts, until nothing more can be added
- Requires KB to contain only Horn clauses
  - At-most one positive literal
- Runs in linear time using two simple tricks:
  - Each symbol X_i knows which rules it appears in
  - Each rule keeps count of how many of its premises are not yet satisfied

Simple theorem proving: Forward chaining

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

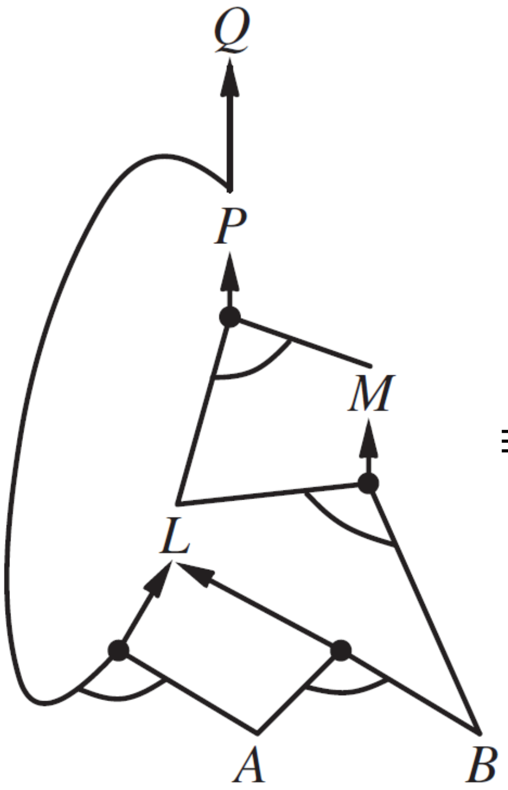
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

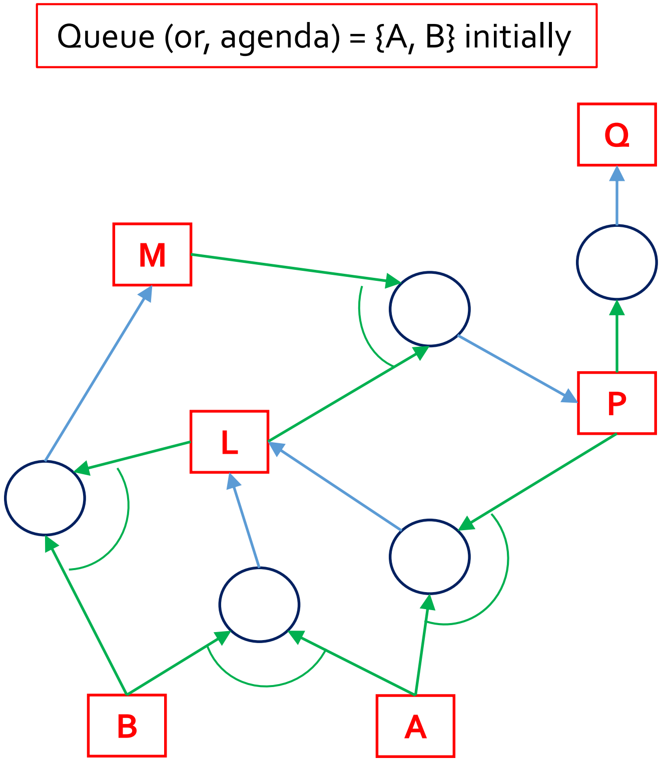
$A \wedge B \Rightarrow L$

A

B



$\equiv$



Forward chaining algorithm: Details

```
function PL-FC-ENTAILS?(KB, q) returns true or false
    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 s
    queue ← a queue of symbols, initially symbols known to be true in KB
    // Some editions use "agenda" to refer to the queue

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

Properties of forward chaining

- Data-driven reasoning: reasoning starts with known data
  - Can be used to arrive at conclusions without specific query
- Theorem: FC is sound and complete for definite-clause KBs
- Soundness of FC follows from soundness of Modus Ponens

- Backward chaining
  - Starts with the query and works backwards
  - Useful for goal-directed reasoning
    - Where is the Wumpus?
  - Faster than linear in KB size since only the relevant sentences are checked

Properties of forward chaining

- Completeness proof:
  - FC reaches a fixed point where no new atomic sentences are derived
  - Consider the final set of known-to-be-true symbols as a model m, other ones false
  - Every clause in the original KB is true in m

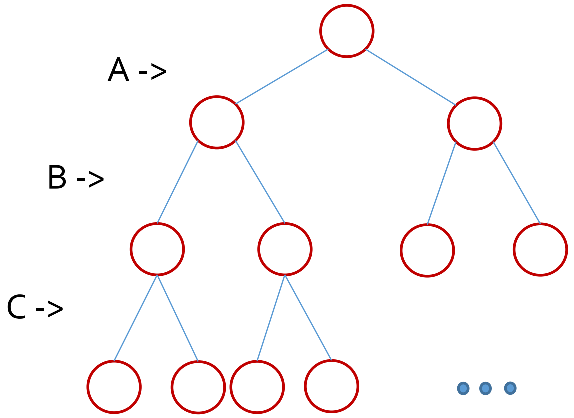
    Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m
    Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m
    Therefore the algorithm has not reached a fixed point!
  - Hence m is a model of KB
  - If KB $\models$ q, q is true in every model of KB, including m

Solving SAT Problems

- Problem:
  - Is a given propositional formula satisfiable?
  - If yes, produce a model.
- Examples:
  - $(P \lor \neg Q) \land (\neg P \lor Q)$? Yes, $(P = T, Q = T)$
  - $P \land \neg P$? No
- SAT is a kind of CSP where the domain is restricted to T and F
- How to solve SAT problems?
  - Truth table enumeration - though complete but not efficient
  - How can we make it faster?

Solving SAT Problems

- Better: partial assignment tree search with backtracking
  - $A \vee \neg B, \neg A \vee B, \neg A \vee \neg B, A \vee B \vee C$



- How can we make it even faster?
  - CNF + Search + Backtracking + Inference (+ Heuristics)

Efficient SAT solvers

- DPLL (Davis-Putnam-Logemann-Loveland, 1962) algorithm is the core of modern solvers
- Recursive depth-first search over models with some extras
  - Early termination: stop if
    - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}
    - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is falsified by {A=false, B=false}
  - Pure literals: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true

Efficient SAT solvers

- Unit clauses: if a clause is left with a single literal, set symbol to satisfy clause
  - E.g., if A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
  - Satisfying the unit clauses often leads to further propagation, new unit clauses

DPLL algorithm

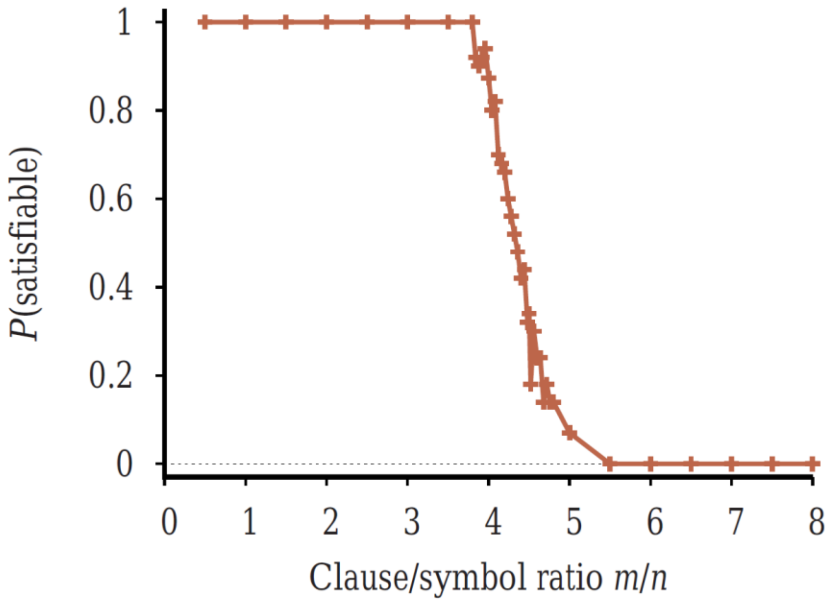
```
function DPLL(clauses, symbols, model) returns true or false
    if every clause in clauses is true in model then return true
    if some clause in clauses is false in model then return false
    P, value ←FIND-PURE-SYMBOL(symbols, clauses, model)
    if P is non-null then return DPLL(clauses, symbols – 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 or(DPLL(clauses, rest, model ∪ {P=true}),
              DPLL(clauses, rest, model ∪ {P=false}))
```

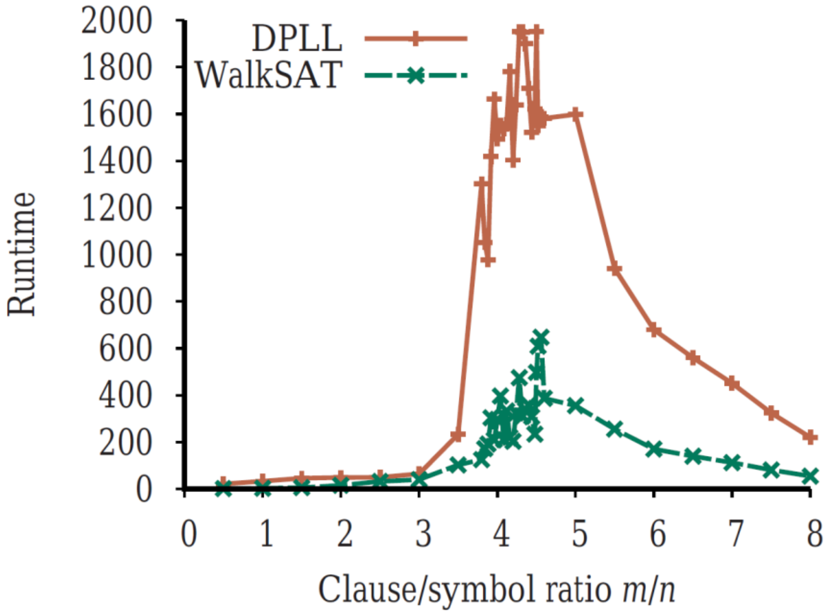
Efficiency

- Naïve implementation of DPLL: solves ~100 variables
- Extras:
  - Smart variable and value ordering
  - Divide and conquer
  - Caching unsolvable subcases as extra clauses to avoid redoing them
  - Cool indexing and incremental recomputation tricks so that every step of the DPLL algorithm is efficient (typically O(1))
    - Index of clauses in which each variable appears +ve/-ve
    - Keep track number of satisfied clauses, update when variables assigned
    - Keep track of number of remaining literals in each clause
- Real implementation of DPLL: solves ~100 Million variables

Probability of satisfiability

- The probability of satisfiability of overconstrained instances (i.e., large clause to symbol ratio) tends to 0
  - The solutions are densely distributed in underconstrained ones
  - The transition is sharp





SAT solvers in practice

- Circuit verification: does this VLSI circuit compute the right answer?
- Software verification: does this program compute the right answer?
- Software synthesis: what program computes the right answer?
- Protocol verification: can this security protocol be broken?
- Protocol synthesis: what protocol is secure for this task?
- Lots of combinatorial problems: what is the solution?
- Planning: how can I kill the wumpus and get the gold?

Summary

- One possible agent architecture: knowledge + inference
- Logics provide a formal way to encode knowledge
  - A logic is defined by: syntax, set of possible worlds, truth condition
- A simple KB for an agent covers the initial state, sensor model, and transition model
- Logical inference computes entailment relations among sentences, enabling a wide range of tasks to be solved

Summary

- Theorem provers apply inference rules to sentences
  - Forward chaining applies modus ponens with definite clauses; linear time
  - Resolution is complete for PL but exponential time in the worst case
- SAT solvers based on DPLL provide incredibly efficient inference
- Logical agents can do localization, mapping, SLAM, planning (and many other things) just using one generic inference algorithm on one knowledge base

- Reading: Chapter 7
- Assignments: WalkSAT, SATPlan, PS 5, logic.ipynb

- Next: First order logic, Chapter 8

- Mid-Term Project Evaluation on Oct 23