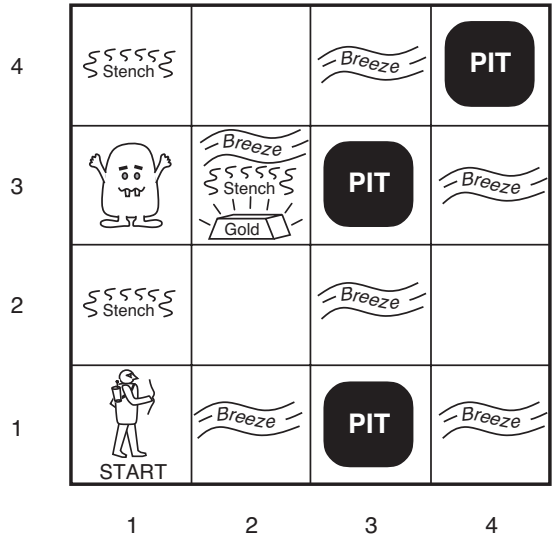
CS444

Nathan Sprague

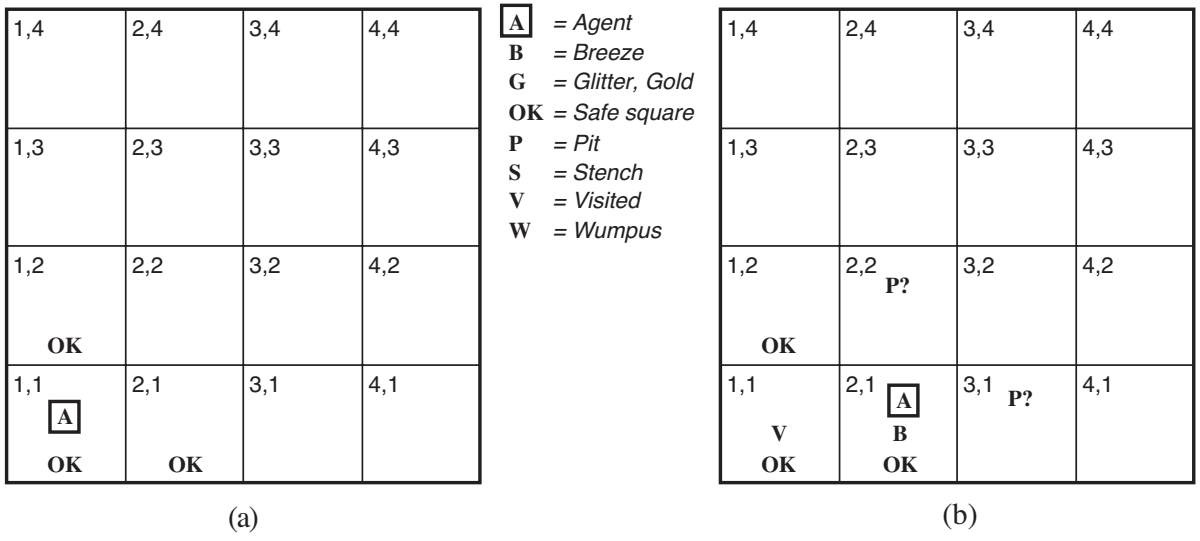
September 14, 2012

Logic!

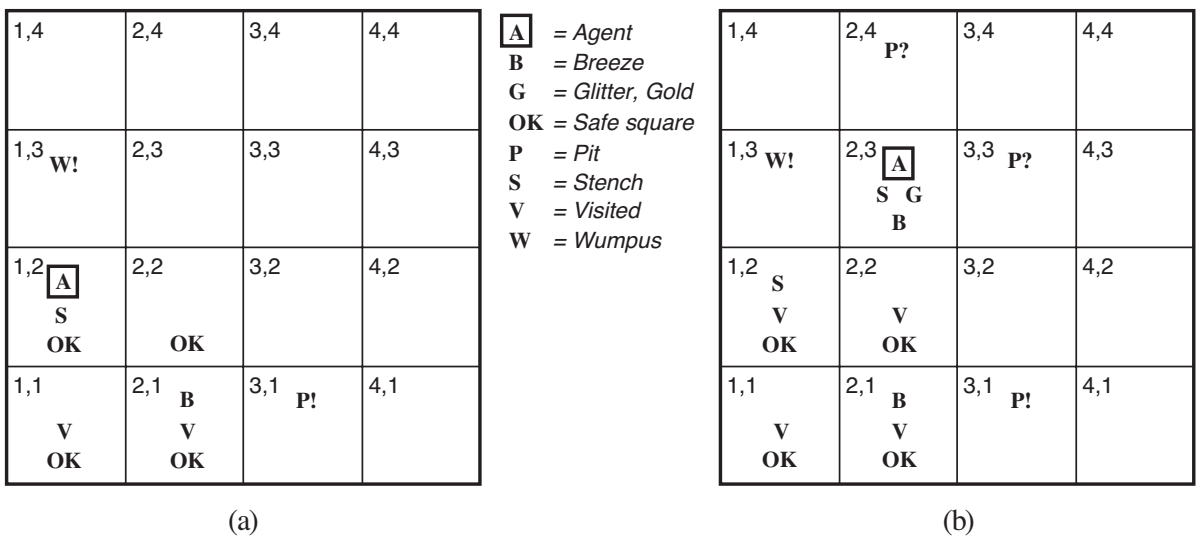
Wumpus World



Wumpus World



(a) (b)



(a) (b)

Propositional Logic

- Symbols represent propositions that can be true or false.
  - (Atomic sentences)
- Complex sentences created from combining atomic sentences with logical connectives:

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

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic.

Semantics of Propositional Logic

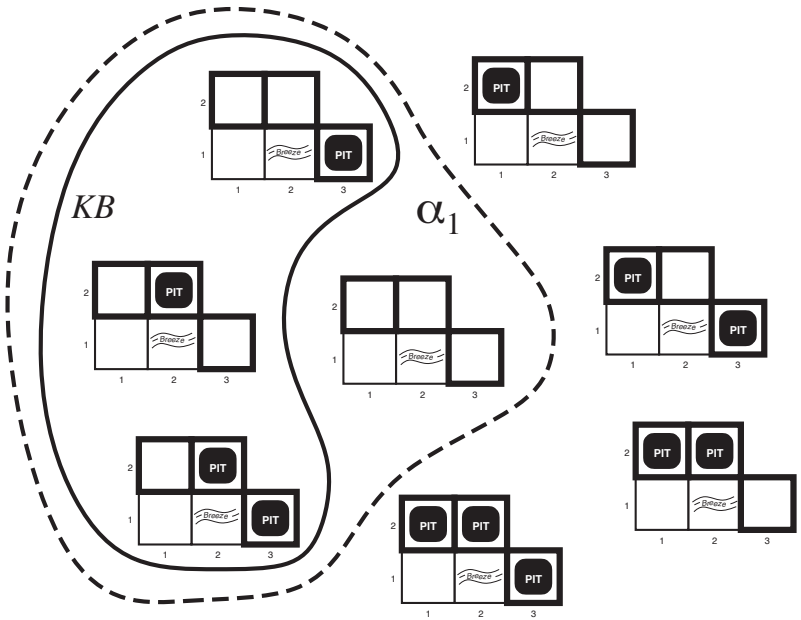
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *true*. Another way to look at this is to think of each row as a model, and the entries under each column for that row as saying whether the corresponding sentence is true in that model.

Models and Entailment

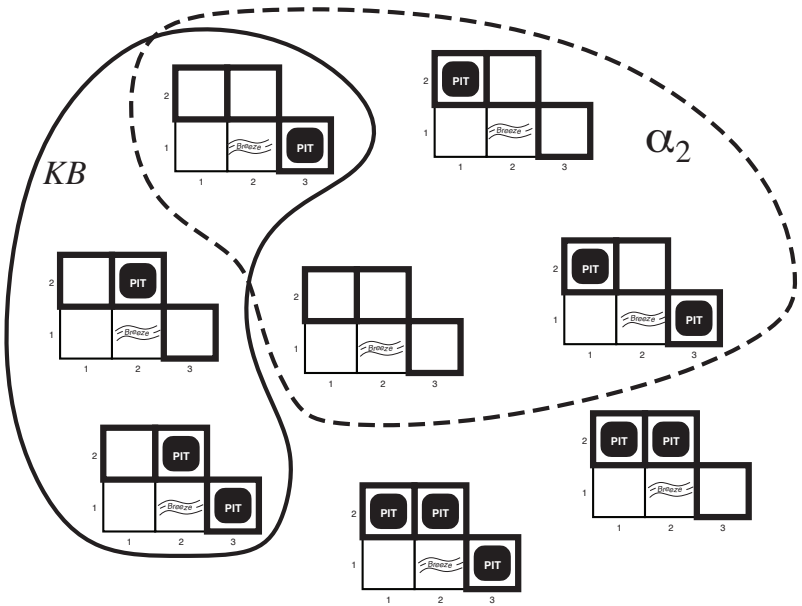
- A model assigns a value to all variables.
  - A possible world.
- Entailment $\alpha \models \beta$
  - β follows logically from α.
  - In every model in which α is true β is true.
  - $\alpha \models \beta$ if and only if $M(\alpha) \subseteq M(\beta)$.

Example...



- KB = Nothing in [1,1] and Breeze in [1,2]
- α_2 = no pit in [1,2]

Example...



- KB = Nothing in [1,1] and Breeze in [1,2]
- α_2 = no pit in [2,2]

Nice Idea, How Do We Implement It?

Nice Idea, How Do We Implement It?

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

  $symbols \leftarrow$ a list of the proposition symbols in KB and α
  return TT-CHECK-ALL($KB, \alpha, symbols, \{\}$)

---

function TT-CHECK-ALL($KB, \alpha, symbols, model$) **returns** *true* or *false*
  if EMPTY?($symbols$) **then**
    if PL-TRUE?($KB, model$) **then return** PL-TRUE?($\alpha, model$)
    else return *true // when KB is false, always return true*
  else do
    $P \leftarrow$ FIRST($symbols$)
    $rest \leftarrow$ REST($symbols$)
    return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
      and
      TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))

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*.

Inference!

- Entailment: $\alpha \models \beta$
- Inference: $\alpha \vdash_i \beta$
- We want inference algorithms that are:
  - Sound
  - Complete
  - What about TT-ENTAILS?

Inference

- Good news:
  - TT-ENTAILS is sound and complete.
- Bad news?
- Worse news: Propositional entailment is co-NP-complete.

Theorem Proving

- Terminology:
  - equivalence $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

Equivalences

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

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

Theorem Proving

- Terminology:
  - equivalence: $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$
  - validity/tautology
  - satisfiability
- Proof by contradiction:
  - $\alpha \models \beta$ if and only if $(\alpha \wedge \neg\beta)$ is unsatisfiable.

Inference and Proofs

Inference Rules: Modus Ponens:

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

And-Elimination:

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

Any equivalence...

Inference and Proofs

- Now we have a second way of automating proofs: Search.
  - Knowledge base is the state.
  - Inference rules are the actions.
  - The goal is the sentence we are trying to prove.

Resolution

- How many inference rules do we need?

Resolution

- One.
- Caveat: can only use it on sentences in conjunctive normal form (CNF)
  - ("or *clauses* "*and*ed together.)
  - Example: $(A \vee B) \wedge \neg C$
    - $(A \vee B)$ is a clause.
    - $\neg C$ is a negative literal. (Also a unit clause.)
- Good news: any sentence in propositional logic can be efficiently converted to CNF.

Conjunctive Normal Form

- Eliminate $\Leftrightarrow$: replace $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
- Eliminate $\Rightarrow$: replace $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
- Move $\neg$ inward using De Morgan's law.
- Distribute $\vee$ over $\wedge$ wherever possible.

Converting to CNF

- $(\neg A \wedge B) \Leftrightarrow C$

Converting to CNF

- $(\neg A \wedge B) \Leftrightarrow C$
- $((\neg A \wedge B) \Rightarrow C) \wedge (C \Rightarrow (\neg A \wedge B))$

Converting to CNF

- $(\neg A \wedge B) \Leftrightarrow C$
- $((\neg A \wedge B) \Rightarrow C) \wedge (C \Rightarrow (\neg A \wedge B))$
- $(\neg(\neg A \wedge B) \vee C) \wedge (\neg C \vee (\neg A \wedge B))$

Converting to CNF

- $(\neg A \wedge B) \Leftrightarrow C$
- $((\neg A \wedge B) \Rightarrow C) \wedge (C \Rightarrow (\neg A \wedge B))$
- $(\neg(\neg A \wedge B) \vee C) \wedge (\neg C \vee (\neg A \wedge B))$
- $((A \vee \neg B) \vee C) \wedge (\neg C \vee (\neg A \wedge B))$

Converting to CNF

- $(\neg A \wedge B) \Leftrightarrow C$
- $((\neg A \wedge B) \Rightarrow C) \wedge (C \Rightarrow (\neg A \wedge B))$
- $(\neg(\neg A \wedge B) \vee C) \wedge (\neg C \vee (\neg A \wedge B))$
- $((A \vee \neg B) \vee C) \wedge (\neg C \vee (\neg A \wedge B))$
- $(A \vee \neg B \vee C) \wedge (\neg C \vee \neg A) \wedge (\neg C \vee B)$

Resolution Inference Rule

$$\frac{l_1 \vee ... \vee l_k, \quad m_1 \vee ... \vee m_n}{l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n}$$

- Where l_i and m_j are complementary literals. Duplicate literals are removed.
- Example: $(A \vee B \vee \neg C)$, $(C \vee \neg D \vee A)$
- Resolves to: $A \vee B \vee \neg D$

Resolution Theorem Proving

- In order to prove $KB \models \alpha$,
  - Convert $KB \wedge \neg\alpha$ to CNF.
  - Apply resolution rule until:
    - No new clauses can be added ($KB \not\models \alpha$)
    - You derive the empty clause ($KB \models \alpha$)

Exercise

Complete the following proof using resolution:
KB:
$A \Rightarrow B$
$\neg(\neg A \wedge C)$
$(C \wedge A)$

Query:
B

Psuedocode

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 $KB \wedge \neg\alpha$
  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.

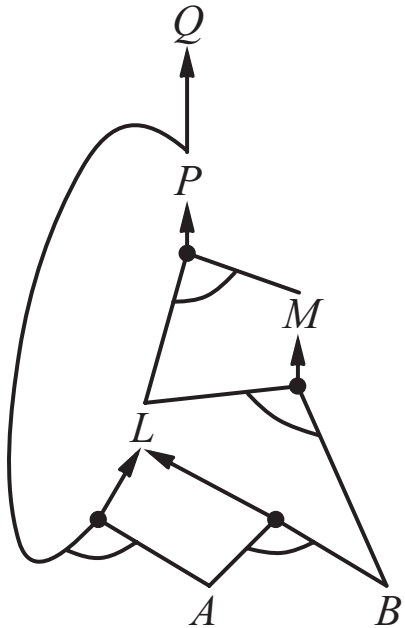
Efficiency?

Definite Clauses and Horn Clauses

- Definite Clause: Exactly one positive literal:
  - $(A \vee \neg B)$, $(A \vee \neg B \neg C)$
- Horn clause: *at most* one positive literal:
  - $(A \vee \neg B)$, A, $(\neg B \neg C)$

Forward Chaining

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



Forward Chaining

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*