Artificial Intelligence

Logical Agents

Christopher Simpkins

Logic and AI

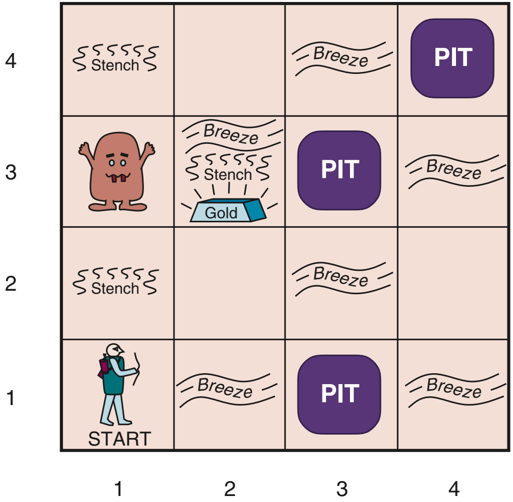
In AI, **knowledge-based** agents use a process of **reasoning** over an internal **representation** of knowledge to decide what actions to take.

Knowledge-Based 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))
  $t \leftarrow t + 1$
  return *action*

The Wumpus World



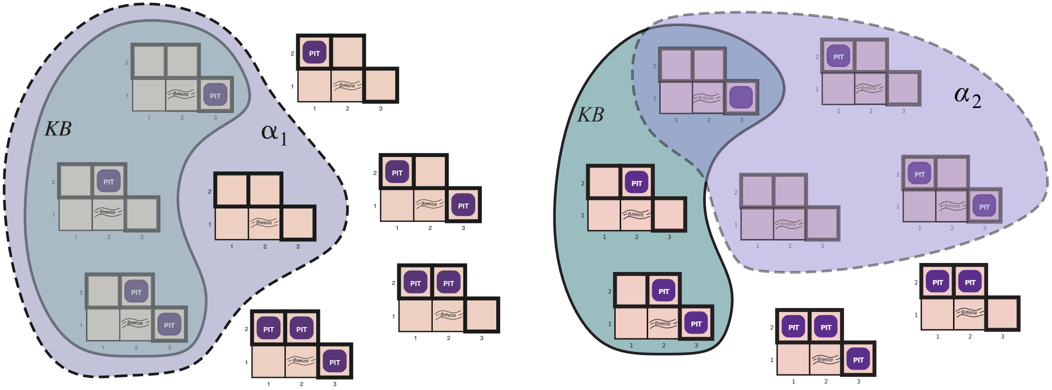
First Steps Wumpus WOrld

(a) (b)

Later Steps Wumpus World

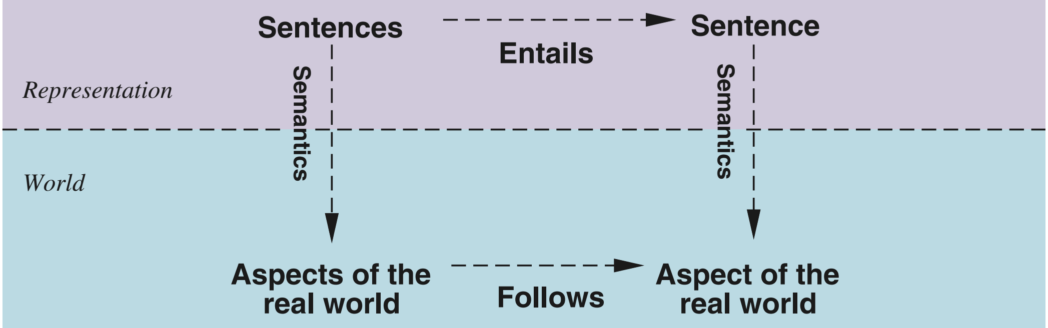
(a) (b)

Logic



(a) (b)

Logic


Sentences
Entails
Sentence
Representation
Semantics
Semantics
World
Aspects of the real world
Follows
Aspect of the real world

Propositional Logic

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{True} \mid \mathit{False} \mid P \mid Q \mid R \mid \ldots \\
\mathit{ComplexSentence} & \rightarrow & (\, \mathit{Sentence}\,) \\
& \mid & \neg\, \mathit{Sentence} \\
& \mid & \mathit{Sentence} \land \mathit{Sentence} \\
& \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
& \mid & \mathit{Sentence} \;\Rightarrow\; \mathit{Sentence} \\
& \mid & \mathit{Sentence} \;\Leftrightarrow\; \mathit{Sentence} \\
\\
\textsc{Operator Precedence} & : & \neg, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

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*

Propositional Theorem Proving

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	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 Theorem Proving

```
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
      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 }))
```

Propositional Theorem Proving

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

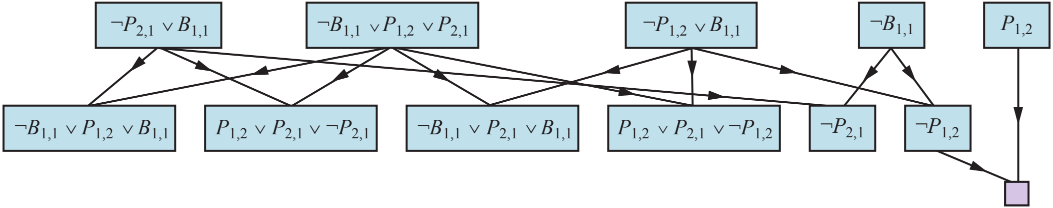
Propositional Theorem Proving

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

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 ← the set of clauses in the CNF representation of KB ∧ ¬α
  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
```

Propositional Theorem Proving



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 initially the number of symbols in clause *c*'s premise
  inferred ← a table, where *inferred*[*s*] is initially *false* for all symbols
  queue ← a queue of symbols, initially symbols known to be true in *KB*

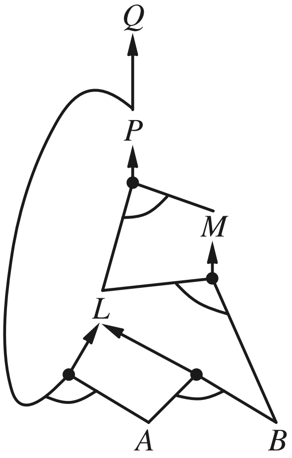
  while *queue* is not empty **do**
    p ← POP(*queue*)
    if *p* = *q* **then return** *true*
    if *inferred*[*p*] = *false* **then**
      inferred[*p*] ← *true*
      for each clause *c* in *KB* where *p* is in *c*.PREMISE **do**
        decrement *count*[*c*]
        if *count*[*c*] = 0 **then**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

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving

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

(a)



(b)

Propositional Model Checking

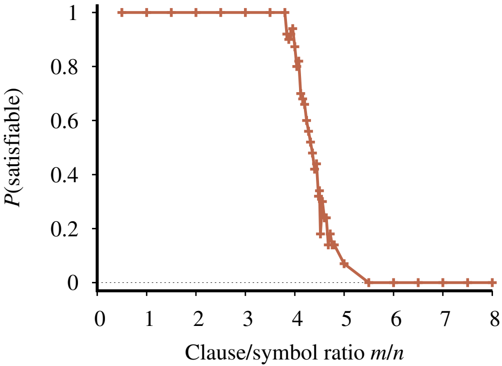
```
function DPLL-SATISFIABLE?(s) returns true or false
   inputs: s, a sentence in propositional logic

   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})
```

Propositional Model Checking

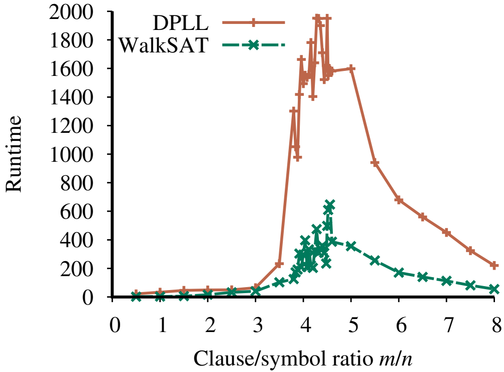
function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
    inputs: *clauses*, a set of clauses in propositional logic
            p, the probability of choosing to do a "random walk" move, typically around 0.5
            max_flips, number of value flips allowed before giving up

    model ← a random assignment of *true*/*false* to the symbols in *clauses*
    for each $i = 1$ **to** *max_flips* **do**
        if *model* satisfies *clauses* **then return** *model*
        clause ← a randomly selected clause from *clauses* that is false in *model*
        if RANDOM$(0, 1) \leq p$ **then**
            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*

Propositional Model Checking



(a)



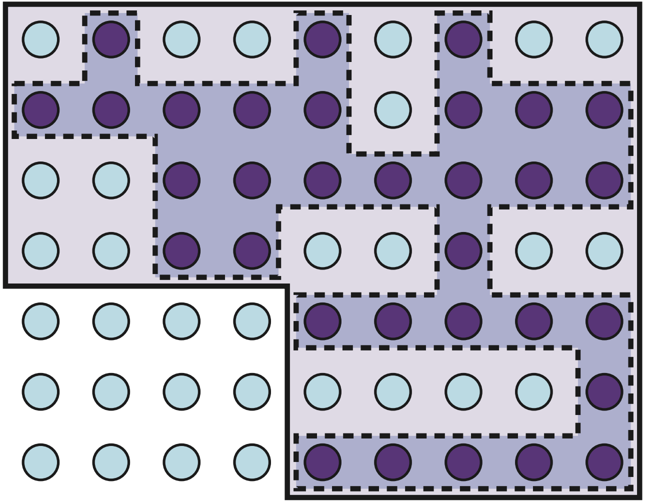
(b)

KENNESAW STATE UNIVERSITY

Agents Based on Propositional Logic

```
function HYBRID-WUMPUS-AGENT(percept) returns an action
  inputs: percept, a list, [stench,breeze,glitter,bump,scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
              t, a counter, initially 0, indicating time
              plan, an action sequence, initially empty

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x,y] : ASK(KB, OK^t_{x,y}) = true}
  if ASK(KB, Glitter^t) = true then
    plan ← [Grab] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  if plan is empty then
    unvisited ← {[x,y] : ASK(KB, L^{t'}_{x,y}) = false for all t' ≤ t}
    plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB, HaveArrow^t) = true then
    possible_wumpus ← {[x,y] : ASK(KB, ¬ W_{x,y}) = false}
    plan ← PLAN-SHOT(current, possible_wumpus, safe)
  if plan is empty then          // no choice but to take a risk
    not_unsafe ← {[x,y] : ASK(KB, ¬ OK^t_{x,y}) = false}
    plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
  if plan is empty then
    plan ← PLAN-ROUTE(current, {[1, 1]}, safe) + [Climb]
  action ← POP(plan)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action

function PLAN-ROUTE(current,goals,allowed) returns an action sequence
  inputs: current, the agent's current position
          goals, a set of squares; try to plan a route to one of them
          allowed, a set of squares that can form part of the route

  problem ← ROUTE-PROBLEM(current, goals,allowed)
  return SEARCH(problem)        // Any search algorithm from Chapter 3
```

Agents Based on Propositional Logic



Agents Based on Propositional Logic

```
function SATPLAN(init, transition, goal, T_max) returns solution or failure
   inputs: init, transition, goal, constitute a description of the problem
           T_max, an upper limit for plan length

   for t = 0 to T_max do
      cnf ← TRANSLATE-TO-SAT(init, transition, goal, t)
      model ← SAT-SOLVER(cnf)
      if model is not null then
         return EXTRACT-SOLUTION(model)
   return failure
```