Artificial Intelligence

Logical AI

Christopher Simpkins

KENNESAW STATE UNIVERSITY

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 base**: a set of sentences.
- **Sentence**: an assertion about the world expressed in a **knowledge represeantation language**, like propositional logic.
- **Axioms**: sentences taken as given – not derived from other sentences, assumptions.
- **Inference**: deriving new sentences from old sentences.
- **Background knowledge**: sentences present in an agent's knowledge base before it starts perceiving and acting.

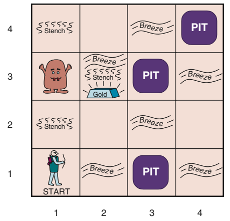
Knowledge-Based Agents

- MAKE-PERCEPT-SENTENCE constructs sentence asserting that agent perceived the given percept at the given time.
- MAKE-ACTION-QUERY constructs sentence that asks what action to take at current time.
- MAKE-ACTION-SENTENCE constructs sentence asserting chosen action was executed.

```
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
```

- Logical agents are described at the **knowledge level** using **declarative** statements of knowledge and goals.
- At the **implemetation level** we use a **procedureal** approach, encoding behaviors directly in program code.

The Wumpus World



A cave that you can drop into or climb out of at square [1, 1].

- **Performance measure**: $+1000$ for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken, –10 for using the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.

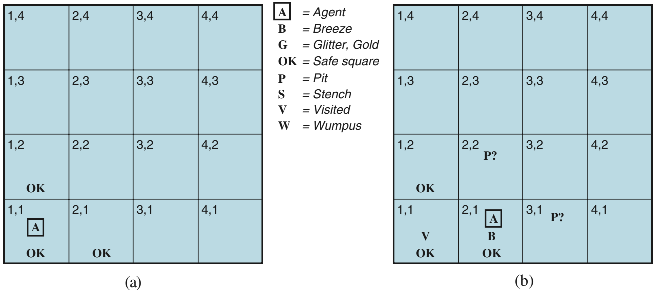
- **Environment**: A 4$\times$4 grid of rooms, agent always starts at [1,1], facing east. Gold and the wumpus placed uniformly randomly from the squares other than the start square. Each non-start square can be a pit with probability 0.2.
- **Actuators**: Forward, TurnLeft by $90°$, or TurnRight by $90°$. The agent dies if it enters pit or a live wumpus square. Moves into walls have no effect. Grab picks up gold if agent in gold square. Shoot can fires arrow in direction agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first Shoot action has any effect. Climb climbs out of the cave if at [1,1].

KENNESAW STATE UNIVERSITY

First Steps Wumpus World

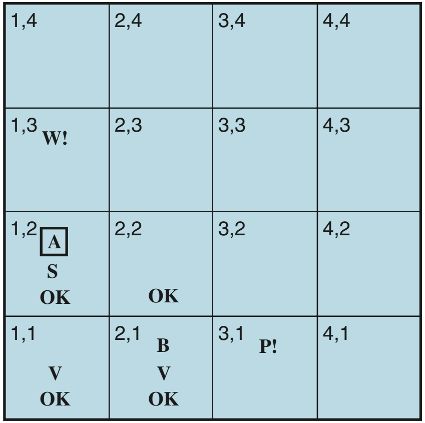
- **Sensors**: The agent has five sensors, each of which gives a single bit of information:
  - In squares directly (not diagonally) adjacent to wumpus, agent perceives a Stench.
  - In squares directly adjacent to a pit, the agent perceives a Breeze.
  - In the square with gold, agent perceives a Glitter.
  - When an agent walks into a wall, it perceives a Bump.
  - When the wumpus is killed, it emits a Scream perceivable anywhere in the cave.

Percepts encoded as list of five symbols indicating presense or absence (by None) of: [Stench,Breeze,Glitter,Bump,Scream] (a bit vector).



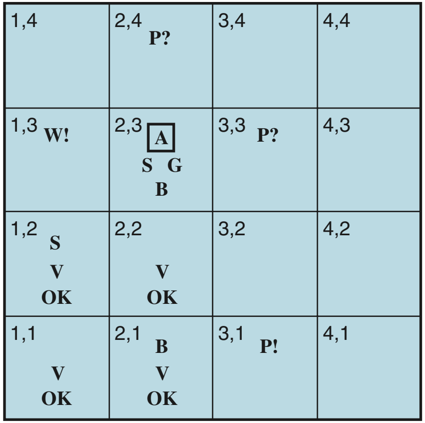
- (a) after percept [None,None,None,None,None]
- (b) after moving to [2,1] and perceiving [None,Breeze,None,None,None]

Later Steps Wumpus World



(a)

$\boxed{A}$	= *Agent*
B	= *Breeze*
G	= *Glitter, Gold*
OK	= *Safe square*
P	= *Pit*
S	= *Stench*
V	= *Visited*
W	= *Wumpus*



(b)

Logic

Basics:

- **Syntax** specifies the form of sentences.
  - $x + y = 4$ is *well-formed*, but $x4y+ =$ is not.
- **Semantics** specifies the meaning of sentences.
  - $x + y = 4$ is **true** in a world where $x = 1$ and $y = 3$.
- **Model**: a formal specification of a *possible world*, that is, a set of assignments of values to the variables in the sentences of a knowledge base.
  - Given a model $\{x = 3, y = 2\}$, the sentence $x + y = 4$ is false.

Satisfaction:

- "m satisfies α": sentence α is true in model m, also "m is a model of α."
- $M(\alpha)$ the set of all models of α, i.e., the set of all models in which α is true.

Entailment: $\alpha \models \beta$: β *follows logically* from α

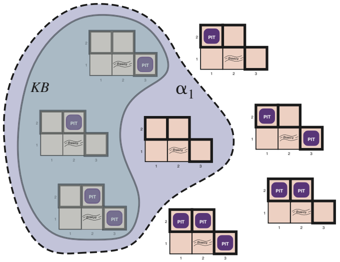
Formal definition of entailment:

$$\alpha \models \beta \text{ if and only if } M(\alpha) \subseteq M(\beta)$$

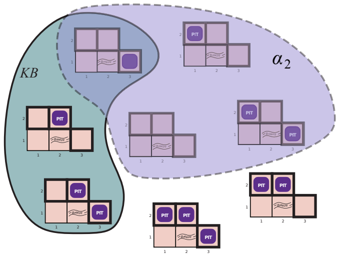
KENNESAW STATE UNIVERSITY

Possible Models of Pits in Wumpus World

The presence of pits in squares [1, 2], [2, 2] and [3, 1] gives rise to $2^3 = 8$ possible models.



(a)



(b)

Solid line delineates KB based on percept [None, None, None, None, None] in [1,1].

- (a). α_1 = "There is no pit in [1, 2]." Here, $KB \models \alpha_1$
- (b). α_2 = "There is no pit in [2, 2]." Here, $KB \not\models \alpha_2$

Logical inference via model checking: because of (b), cannot conclude α_2 (or $\neg\alpha_2$).

- $M(KB) \models \alpha_1$ but $M(KB) \not\models \alpha_2$

Inference Algorithms

If an inference algorithm i can derive α from KB, we write

$$KB \vdash_i \alpha$$

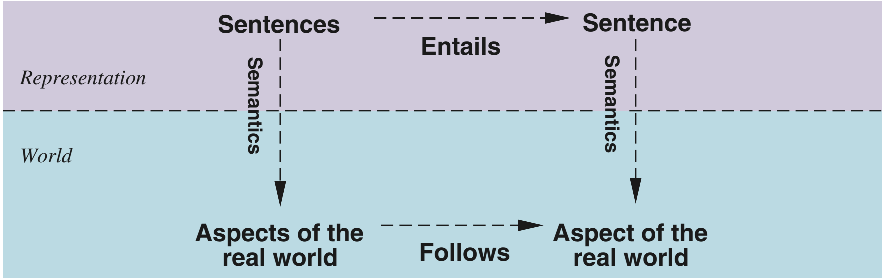
which is pronounced "α is derived from KB by i" or "i derives α from KB."

Important properties of inference algorithms:

- An inference algorithm that derives only entailed sentences is called **sound** or **truth-preserving**.
- An inference algorithm is **complete** if it can derive any sentence that is entailed.

Representation vs World

If KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world?



Grounding: connection between logical reasoning and the real environment. How do we know that KB is true in the real world.

- Subject of volumes of philosophical investigation.
- For us: if agent perceives it, it is true.

Propositional Logic

- **Atomic** sentences consist of a single proposition symbol.
- Proposition symbol stands for a proposition that can be true or false.
  - We use symbols that start with uppercase letter and may contain other letters or subscripts, e.g., : P, Q, R, $W_{1,3}$ and FacingEast
  - True and False have fixed meanings
- **Complex sentence**: one or more atomic sentences constructed from **logical connectives**.
- $\neg$ (not) Unary connective. $\neg W_{1,3}$ is the **negation** of $W_{1,3}$
  - A **positive literal** is an atomic sentence.
  - a ** negative literal** is a negated atomic sentence.
- $\wedge$ (and). Binary connective. **Conjunction**, e.g., $W_{1,3} \wedge P_{3,1}$
- $\vee$ (or). Binary connective. **Disjunction**, e.g., $(W_{1,3} \wedge P_{3,1}) \vee W_{2,2}$
- $\implies$ (implies). Binary connective. **Implication**, e.g., $(W_{1,3} \wedge P_{3,1}) \implies \neg W_{2,2}$
  - $(W_{1,3} \wedge P_{3,1})$ is the **premise** or **antecedent**.
  - $\neg W_{2,2}$ is the **conclusion** or **consequent**.
  - Also known as rules or if-then statements.
  - Some authors use $\supset$ or $\rightarrow$
- $\iff$ (if and only if). Binary connective. $W_{1,3} \iff \neg W_{2,2}$ is a **biconditional**

Grammar of Propositional Logic

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

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*

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving

So far we've done **model checking**: enumerating models and showing that the sentence must hold in all models.

Now we turn to **theorem proving**: applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.

Some basic concepts:

- **Logical equivalence**: two sentences α and β are logically equivalent if they are true in the same set of models. $\alpha \equiv \beta$
  - $\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$
- **Validity**: A sentence is valid if it is true in all models. For example, the sentence $P \wedge \neg P$ is valid. Valid sentences are also known as **tautologies**.

Deduction theorem:

For any sentences α and $beta$, $\alpha \models \beta$ if and only iff the sentence $(\alpha \implies \beta)$ is valid.

Inference Rules

General form:

$$\frac{Givens}{Conclusions}$$

Modus ponens

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*	$\underline{\mathit{true}}$
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	$\underline{\mathit{true}}$
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	$\underline{\mathit{true}}$
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?($KB$, $\alpha$) returns $true$ or $false$
    inputs: $KB$, the knowledge base, a sentence in propositional logic
            $\alpha$, the query, a sentence in propositional logic

    $symbols \leftarrow$ a list of the proposition symbols in $KB$ and $\alpha$
    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
        $P \leftarrow$ FIRST($symbols$)
        $rest \leftarrow$ REST($symbols$)
        return (TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = true\}$)
                and
                TT-CHECK-ALL($KB, \alpha, rest, model \cup \{P = false\}$))
```

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

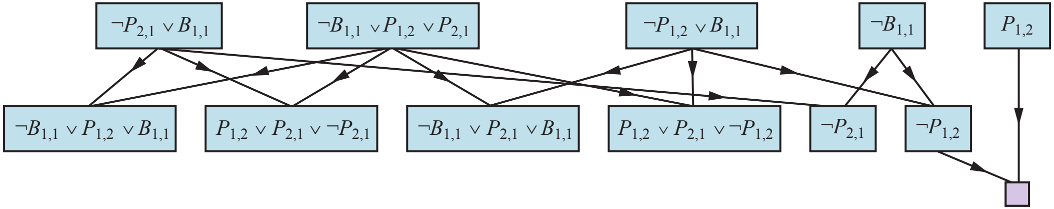
$$\begin{array}{rcl}
\mathit{CNFSentence} & \rightarrow & \mathit{Clause}_1 \wedge \cdots \wedge \mathit{Clause}_n \\
\mathit{Clause} & \rightarrow & \mathit{Literal}_1 \vee \cdots \vee \mathit{Literal}_m \\
\mathit{Fact} & \rightarrow & \mathit{Symbol} \\
\mathit{Literal} & \rightarrow & \mathit{Symbol}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

Propositional Theorem Proving

```
function PL-RESOLUTION($KB$, $\alpha$) returns true or false
  inputs: $KB$, the knowledge base, a sentence in propositional logic
          $\alpha$, the query, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of $KB \land \neg\alpha$
  $new \leftarrow \{\}$
  while true 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
     $clauses \leftarrow clauses \cup new$
```

KENNESAW STATE UNIVERSITY

Propositional Theorem Proving


$\neg P_{2,1} \vee B_{1,1}$
$\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
$\neg P_{1,2} \vee B_{1,1}$
$\neg B_{1,1}$
$P_{1,2}$
$\neg B_{1,1} \vee P_{1,2} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{2,1}$
$\neg B_{1,1} \vee P_{2,1} \vee B_{1,1}$
$P_{1,2} \vee P_{2,1} \vee \neg P_{1,2}$
$\neg P_{2,1}$
$\neg P_{1,2}$

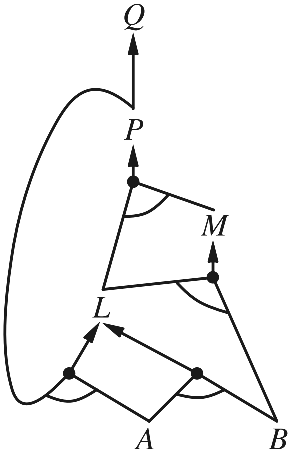
Propositional Theorem Proving

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

   while $queue$ is not empty do
      $p \leftarrow$ POP($queue$)
      if $p = q$ then return $true$
      if $inferred[p] = false$ then
         $inferred[p] \leftarrow true$
         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 $false$
```

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)

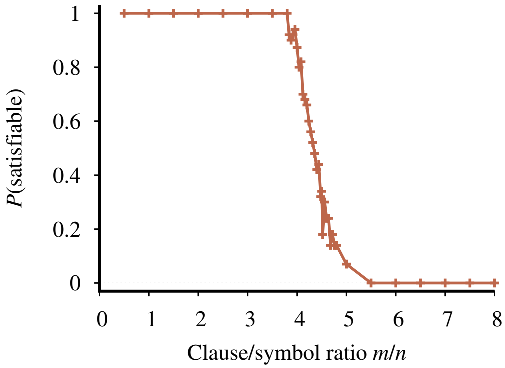
Propositional Model Checking

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

  $clauses \leftarrow$ the set of clauses in the CNF representation of $s$
  $symbols \leftarrow$ 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 $false$
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if $P$ is non-null then return DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if $P$ is non-null then return DPLL($clauses, symbols - P, model \cup \{P{=}value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P{=}true\}$) or
         DPLL($clauses, rest, model \cup \{P{=}false\}$)
```

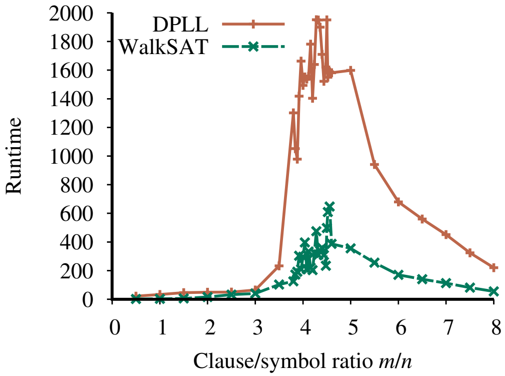
Propositional Model Checking

```
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)

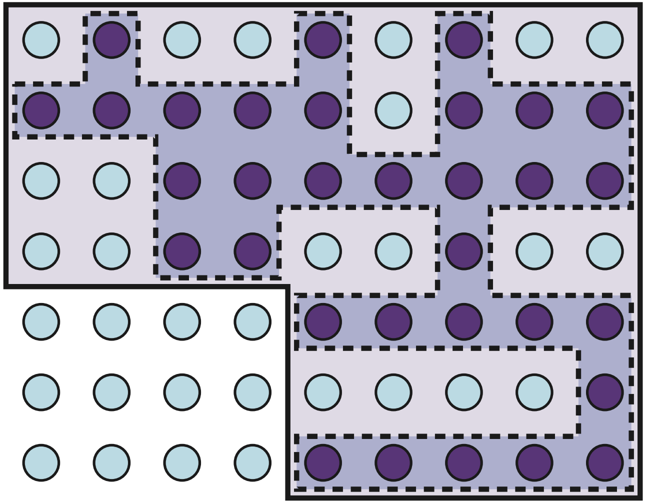


(b)

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 $t' \leq t$}
    plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB, $HaveArrow^t$) = true then
    possible_wumpus ← {[x,y] : ASK(KB, $\neg W_{x,y}$) = 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, $\neg OK^t_{x,y}$) = 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
    $cnf \leftarrow$ TRANSLATE-TO-SAT($init$, $transition$, $goal$, $t$)
    $model \leftarrow$ SAT-SOLVER($cnf$)
    if $model$ is not null then
      return EXTRACT-SOLUTION($model$)
  return $failure$
```

KENNESAW STATE UNIVERSITY