Izmir Institute of Technology

CENG 461 – Artificial Intelligence

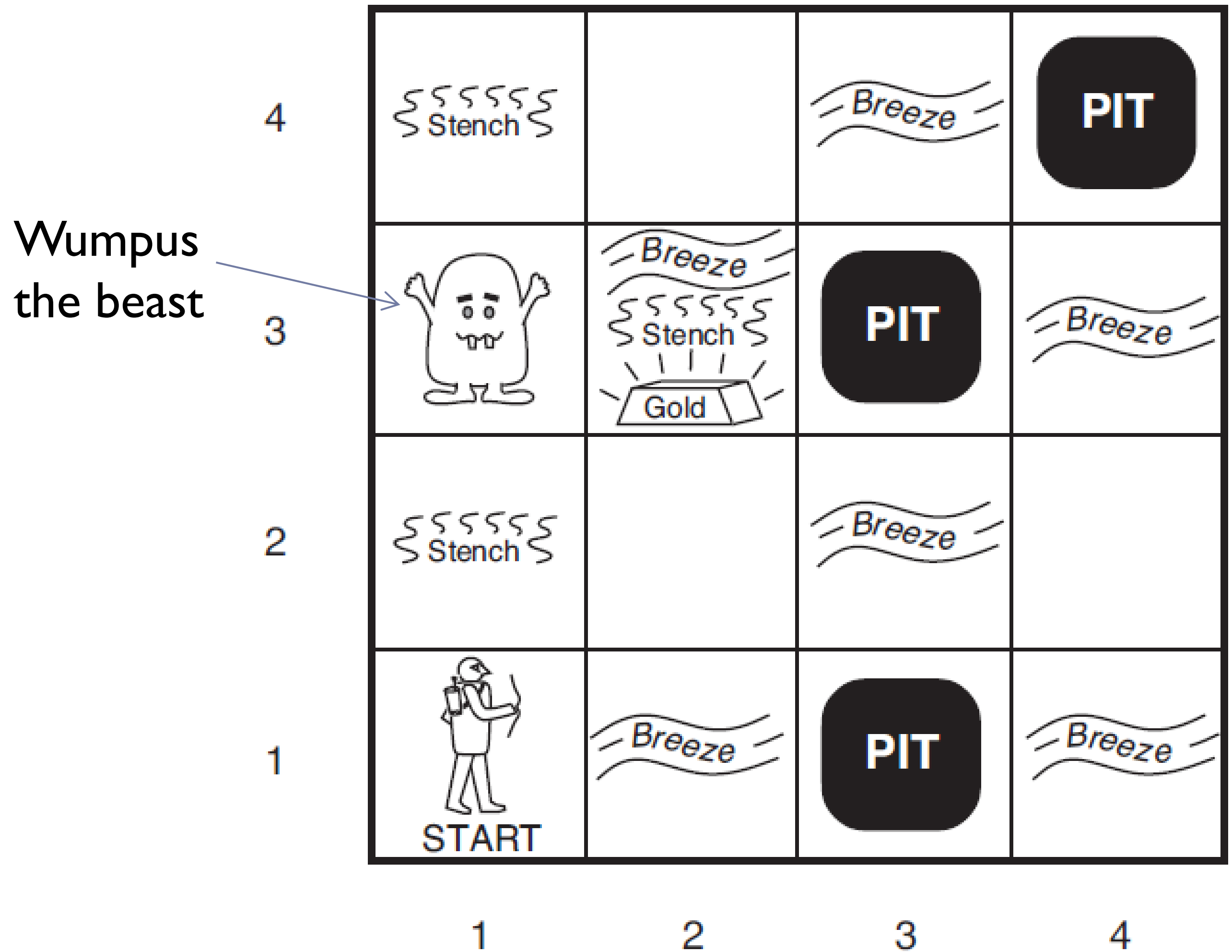
Reasoning with Logic

2015-16 Fall Term

Knowledge-Based Agents

- The search algorithms we have learned previously are quite limited in how they represent the knowledge about the problem.
- For example, it is not possible to reason about the reachability of the solution just from the rules of the 8-puzzle game.
- Knowledge based agent represent their knowledge about the problem by a set of sentences.
- Collectively these sentences are called the knowledge base.

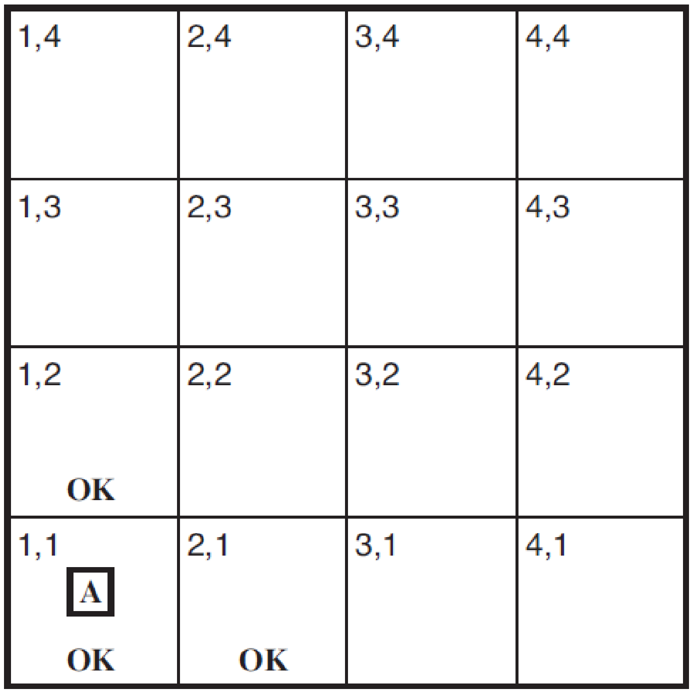
Wumpus World



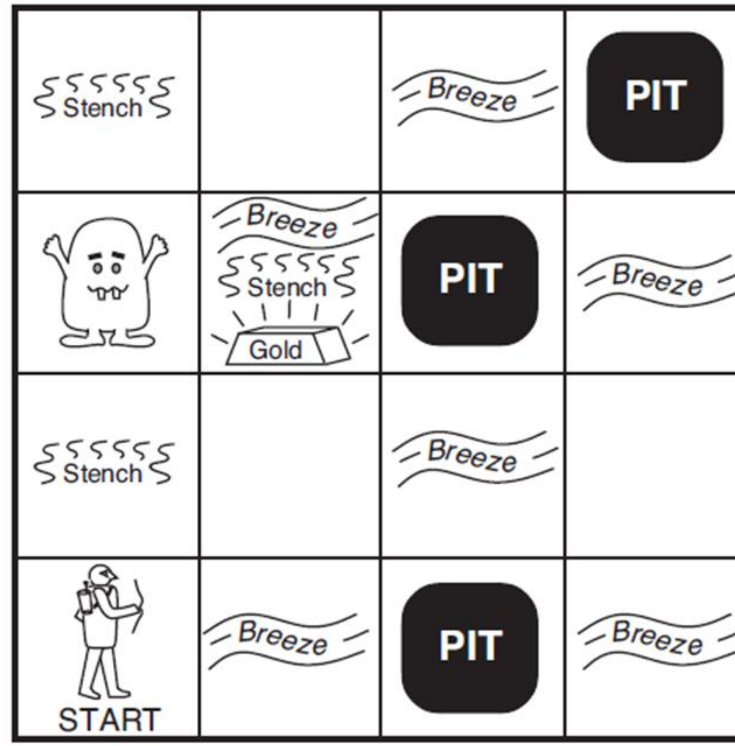
Wumpus World: PEAS description

- Performance Measure
  - +1000 for climbing out with gold, -1000 for death in a pit or by the Wumpus, -1 for each action taken
- Environment
  - 4x4 square grid, agent starts at [1, 1] facing right. Gold and Wumpus location picked at random, must be different than the start location. Each square other than the start has a 0.2 chance to contain a pit.
- Actions
  - The agent can turn left/right 90 degrees, move forward. It can pick up gold, stay at the same square if it bumps to a wall. Dies at a square with a live Wumpus or a pit. Can climb out from the square [1, 1].
- Sensors
  - In the squares directly next to the Wumpus, the agent perceives a **stench**.
    There is a **breeze** in squares next to a pit.
    In the square with the gold there is a **glitter**.
    If the agent hits a wall, it perceives a **bump**.

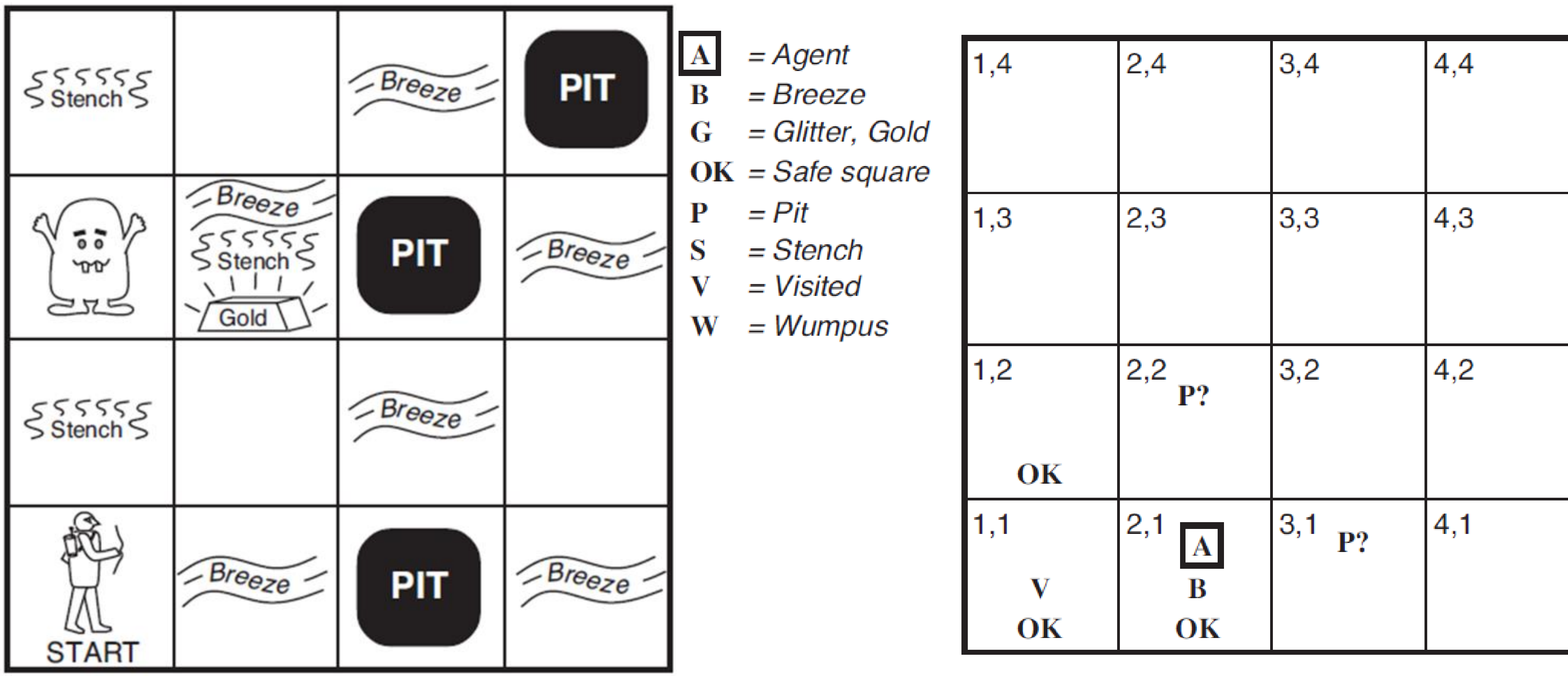
Wumpus World: An example conquest



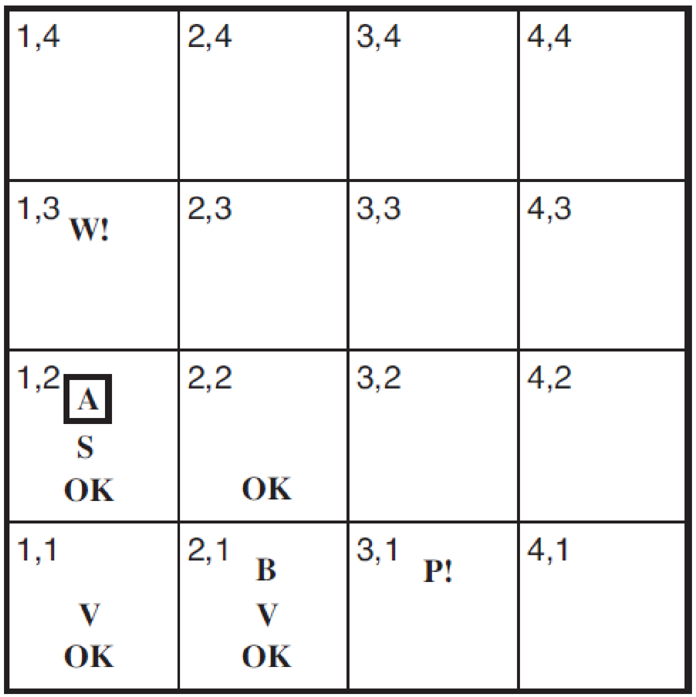
A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*



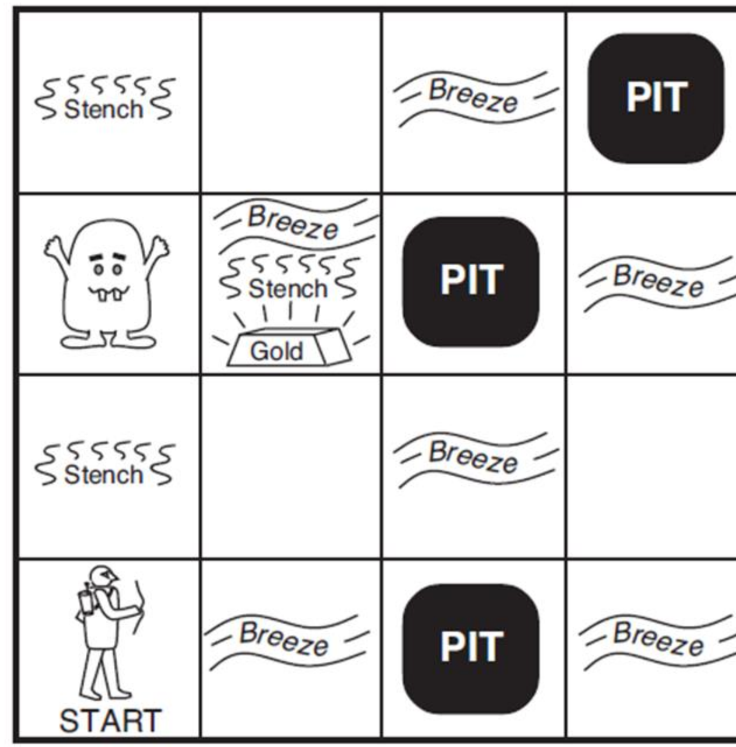
Wumpus World: An example conquest



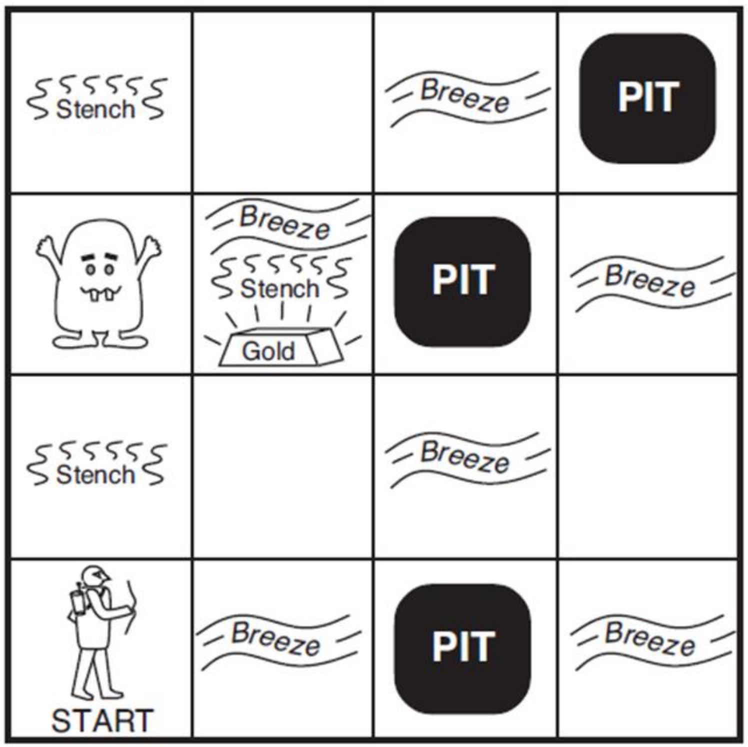
Wumpus World: An example conquest



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



Wumpus World: An example conquest



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

1,4	2,4 **P?**	3,4	4,4
1,3 **W!**	2,3 $\boxed{A}$ **S G B**	3,3 **P?**	4,3
1,2 **S V OK**	2,2 **V OK**	3,2	4,2
1,1 **V OK**	2,1 **B V OK**	3,1 **P!**	4,1

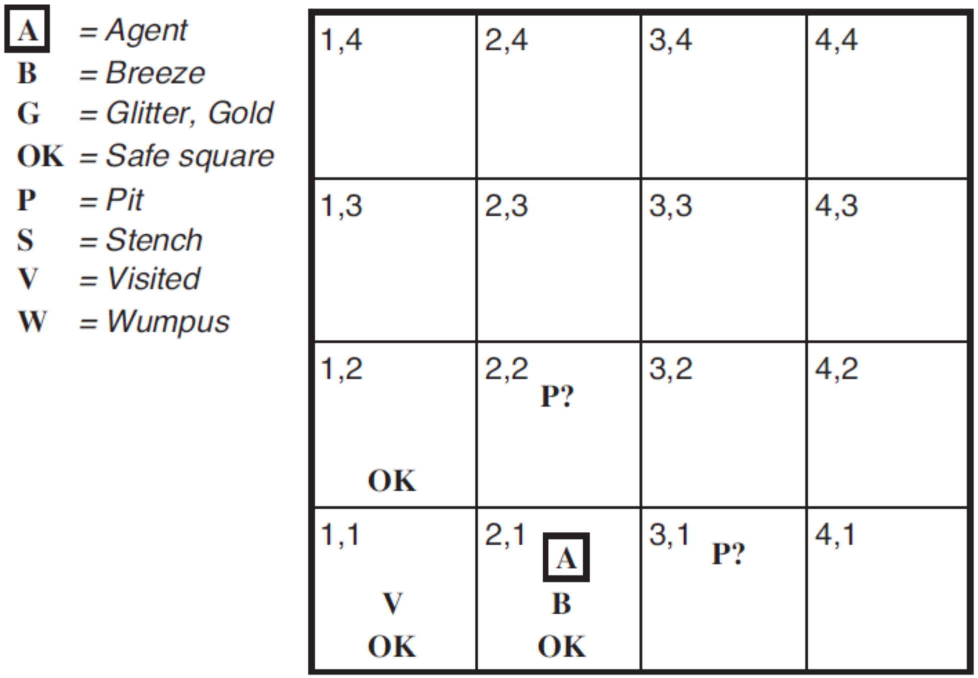
Knowledge Representation with Logic

- One way to represent our knowledge about the world is to construct logical sentences.
- Each particular form of logic constrains what we can say in these sentences and how we must say it.
  - A logic form has a particular syntax
  - It also defines the meaning of sentences
- In standard logic, sentences are either true or false.
- A model represents the state of a possible world.
  A model m is said to satisfy a logical sentence if the sentence is true in m.

Logical Reasoning

- Once we have a representation of the real world with logical sentences, we can infer new facts about the world using rules of logic.
- If a sentence follows logically from another sentence, we write: $\alpha \vDash \beta$, this is called logical entailment.
- $\alpha \vDash \beta$ if and only if, in every model in which α is true, β is also true.
- We can think of the knowledge base KB as a single sentence that asserts the truth of all sentences in itself.
- We can then write KB $\vDash \beta$ if β can be inferred from the sentences in the knowledge base.

Knowledge Base: Example



- Let KB = 'Nothing is sensed in [1,1]' and 'there is a breeze in [2,1]'.
- Let two possible conclusions be:

α_1=There's no pit in [1,2]

α_2=There's no pit in [2,2]

- Can you write the followings?

$KB \models \alpha_1$ Yes

$KB \models \alpha_2$ No

Propositional Logic: Syntax

- Atomic sentences are formed by a single propositional symbol.
- Example: $W_{1,3}$ denotes that the Wumpus is in [1, 3]
- The symbol *True* is always true, *False* is always false.
- Complex sentences are made by combining simple sentences by:
  - $\neg$: Not, negation of truth
  - $\land$: And, logical conjunction
  - $\lor$: Or, logical disjunction
  - $\Rightarrow$: Implication
  - $\Leftrightarrow$: Equivalence

Propositional Logic: Semantics

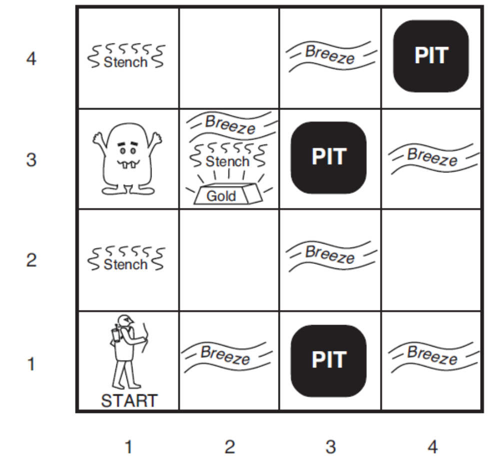
- A model m = {P=true, Q=false, R=true} specifies the truth value of every symbol related to the world
- We can decide the truth of any sentence by using the rules below together with a given model:
  - Truth of each atomic sentence is given by the model m
  - $\neg P$ is true iff P is false in m
  - $P \wedge Q$ is true iff P and Q are both true in m
  - $P \vee Q$ is true iff either of P or Q is true in m
  - $P \Rightarrow Q$ is true unless (P is true and Q is false) in m
  - $P \Leftrightarrow Q$ is true if P and Q are both true or both false in m

Propositional Logic

- Truth Tables

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔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

Knowledge Base for the Wumpus World



- 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]
  - $B_{x,y}$ is true if the agent perceives a breeze in [x, y]
  - $S_{x,y}$ is true if the agent perceives a stench in [x, y]
- There is no pit in [1,1]:
  - $\mathbf{R_1} : \neg P_{1,1}$
- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square.
  - $\mathbf{R_2} : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
  - $\mathbf{R_3} : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- The breeze percepts for the first two squares visited:
  - $\mathbf{R_4} : \neg B_{1,1}$
  - $\mathbf{R_5} : B_{2,1}$

Inference Example for the Wumpus World

- Let's check if $\neg P_{1,2}$ is entailed by our KB (**R_1** to **R_5**).
- Algorithm 1: Model-Checking:
  Enumerate all possible models and check if $\neg P_{1,2}$ is true in all the models in which KB is true.
- The relevant symbols are
  - $B_{1,1}$, $B_{2,1}$, $P_{1,1}$, $P_{1,2}$, $P_{2,1}$, $P_{2,2}$, and $P_{3,1}$.
- There are $2^7 = 128$ possible models.
- In three of these, KB is true.
- In those three, $\neg P_{1,2}$ is true, hence there is no pit in [1,2].
- $P_{2,2}$ is true in two of the three models and false in one, so we cannot yet tell whether there is a pit in [2,2].

Model-Checking for $\neg P_{1,2}$

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

- Algorithm 2: We can try to prove that a statement is true by using the KB as a set of axioms and applying inference rules.
- If the number of models is large but the proof is short, Algorithm 2 is more efficient.

Inference Rules

- $((\alpha \Rightarrow \beta) \wedge \alpha) \Rightarrow \beta$ ... Modus ponens
- $(\alpha \wedge \beta) \Rightarrow \alpha$ ... And elimination
- $\neg(\neg\alpha) \Leftrightarrow \alpha$ ... double negation
- $(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\beta \Rightarrow \neg\alpha)$ ... contraposition
- $(\alpha \Rightarrow \beta) \Leftrightarrow (\neg\alpha \vee \beta)$ ... implication elimination
- $(\alpha \Leftrightarrow \beta) \Leftrightarrow (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$ ... biconditional elimination
- $\neg(\alpha \wedge \beta) \Leftrightarrow (\neg\alpha \vee \neg\beta)$ ... De Morgan
- $\neg(\alpha \vee \beta) \Leftrightarrow (\neg\alpha \wedge \neg\beta)$ ... De Morgan
- $\alpha \wedge (\beta \vee \gamma) \Leftrightarrow (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ ... distribution
- $\alpha \vee (\beta \wedge \gamma) \Leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$ ... distribution

Propositional Theorem Proving

- Check if $\neg P_{1,2}$ is entailed by our KB:
  - R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$ ...biconditional elimination
  - R_6: $(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$ ... $\land$ elim.
  - R_7: $((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1})$ ...contraposition
  - R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \lor P_{2,1}))$
  - R_4: $\neg B_{1,1}$ ...Modus ponens R_8 and R_4
  - R_9 : $\neg(P_{1,2} \lor P_{2,1})$ ...De Morgan
  - R_{10} : $\neg P_{1,2} \land \neg P_{2,1}$ ... $\land$ elimination
  - R_{11} : $\neg P_{1,2}$

Limitations of Propositional Logic

- No uncertainty
  - We can only have two values: true or false
- No objects/relations/functions
  - We can not define objects, relations between them or functions of them.
  - E.g.: "Squares neighboring a pit are breezy."
- No shorthand notation
  - We do not have a notation to express that a group of sentences are true.
  - E.g. If we know that no location in Wumpus world contains a pit, we can not say it shortly. We have to express it as a conjunction of all locations.

First-order Logic

- We can extend propositional logic to include
  - Objects
  - Relations
  - Functions
  - Quantifiers such as "for all", "there exists"

Formal Languages

Language	Ontological Commitment (What exist in the world)	Epistemological Commitment (What an agent believes)
Propositional logic	facts	true / false / unknown
First-order logic	facts, objects, relations	true / false / unknown
Probability theory	facts	degree of belief $\in [0, 1]$
~~Fuzzy logic~~	~~Facts with degree of truth~~ $\in [0, 1]$	~~known interval value~~

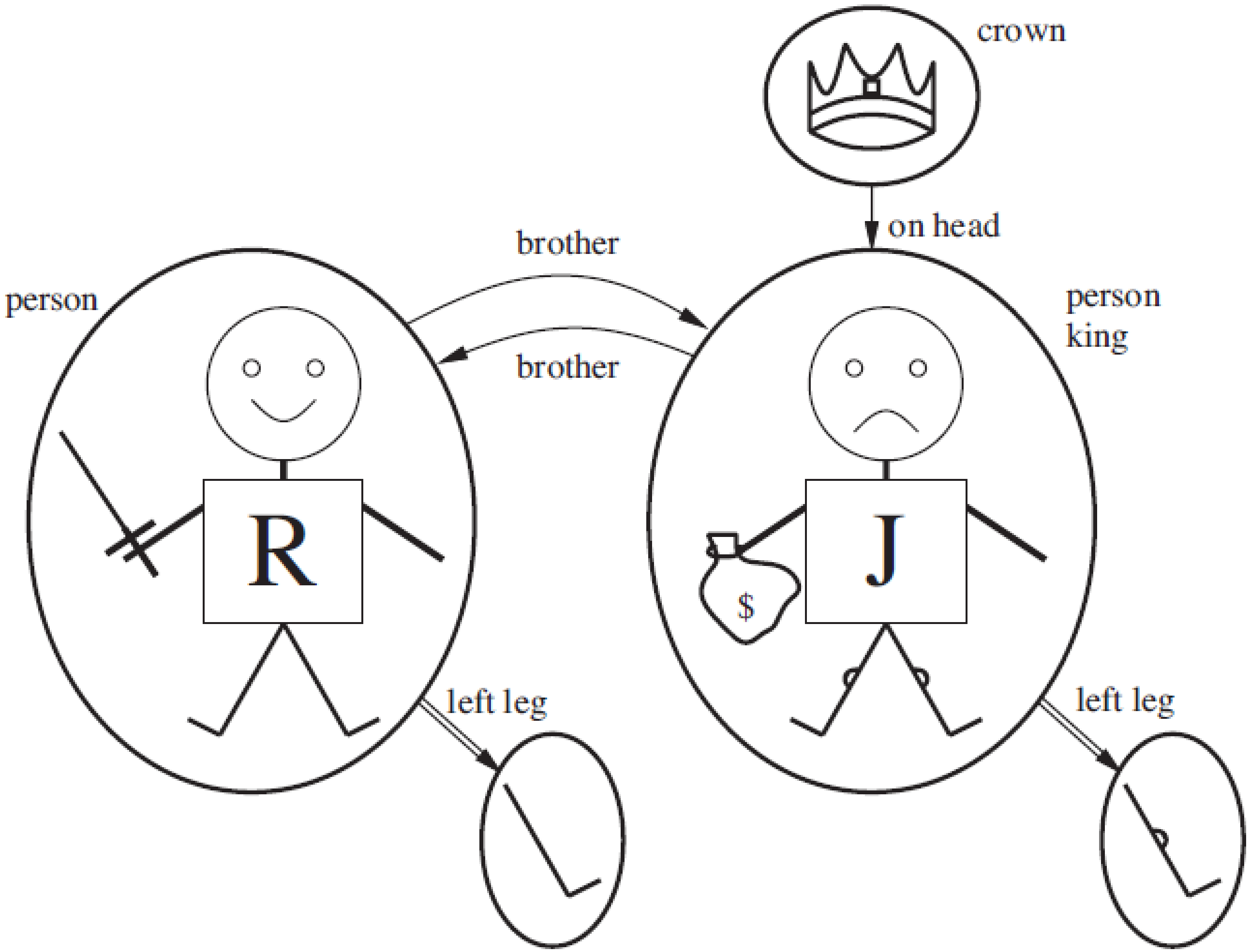
First-order Logic

- Objects
  - A model in first-order logic contains objects
- Relations
  - A relation is just a set of tuples of objects.

    R = {<A, B>, <C, D>}: A binary relation R for which A&B is related, C&D is also related.
  - A unary relation (that takes a single object) is a property.

    P = {<A>, <F>}: Object A has the property P, so does the object F.

Example: First-order Logic



Sentences in First-order Logic

- Brother (Richard, John)
- Brother (Richard, John) $\wedge$ Brother (John, Richard)
- King(Richard) $\vee$ King(John)
- $\neg$King(Richard) $\Rightarrow$ King(John)

Quantifiers in First-order Logic

- Instead of enumerating each object, we can express properties of a collection of objects with quantifiers.
- Universal quantification (∀)
  - $\forall x\ King(x) \Rightarrow Person(x)$
  - ∀ is usually pronounced "for all"
- Sometimes, we want to speak about some object (as opposed to all objects) without naming a particular one.
- Existential quantification (∃)
  - $\exists x\ Crown(x) \land OnHead(x, John)$
  - $\exists x$ is usually pronounced as "there exists"

Negation of ∀ and ∃

- $\forall x$, Likes(x, IceCream) is equivalent to
  $\neg \exists x$, $\neg$Likes(x, IceCream)
- $\forall x \, \neg P \equiv \neg \exists x \, P$
- $\neg \forall x \, P \equiv \exists x \, \neg P$
- $\forall x \, P \equiv \neg \exists x \, \neg P$
- $\exists x \, P \equiv \neg \forall x \, \neg P$

Back to the Wumpus World

- Actions
  - Turn(Right), Turn(Left), Forward, Grab, Climb
- Relations
  - Adjacency of squares:

    $\forall x, y, a, b \; Adjacent([x, y], [a, b]) \Leftrightarrow$

    $(x = a \wedge (y = b - 1 \vee y = b + 1)) \vee$

    $(y = b \wedge (x = a - 1 \vee x = a + 1))$
  - At(Agent, s): Agent is at square s
  - Breezy(s): There is a breeze at square s
  - BestAction(s,x): x is the best thing to do at square s

Back to the Wumpus World

- Diagnostic rules lead from observed effects to hidden causes
  - $\forall s\ Breezy(s) \Rightarrow \exists r\ \ Adjacent\ (r, s) \wedge Pit\ (r)$
  - $\forall s\ \neg Breezy(s) \Rightarrow \neg\exists r\ \ Adjacent\ (r, s) \wedge Pit\ (r)$
  - $\forall s\ Glitter(s) \Rightarrow BestAction(s, Grab)$
- Causal rules reflect the assumed direction of causality
  - $\forall r\ Pit(r) \Rightarrow \forall s\ \ Adjacent\ (r, s) \Rightarrow Breezy\ (s)$
  - $\forall s\ (\forall r\ Adjacent\ (r, s) \Rightarrow \neg Pit\ (r)) \Rightarrow \neg\ Breezy\ (s)$

- Similar to propositional logic, with KB as a set of axioms and applying inference rules, we can reach conclusions

Planning

- The problem solving agent needs domain specific heuristics to perform well, otherwise search space is too big.
- The propositional logic based agent may not be adequate when there are many actions and states.
- Let's define a way to plan a set of actions from an initial state to a goal state, named <u>classical planning</u>.

Classical Planning

- States
  - Each state is a conjunction of facts about the world.
  - Anything not specified true is accepted as false.
  - Every object corresponds to a single Constant Symbol

  E.g.: $At(Plane_1, Melbourne), At(Plane_2, Sydney), At(Fred, Sydney)$
- Goals
  - Goal may be a state. E.g.: $At(Plane_2, Tahiti)$
  - Or another condition. E.g.: $Rich(Fred) \land Famous(Fred)$

Classical Planning

- Actions
  - Defined in terms of preconditions before it is executed and effects occur after it is executed.
  - E.g. An action for flying a plane from a location to another:

Action(Fly(p, from, to),
 PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
 EFFECT: $\neg At(p, from) \wedge At(p, to)$)

  - For concrete actions we need to substitute real objects in place of variables in the action

Action(Fly(P_1, SFO, JFK),
 PRECOND: $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
 EFFECT: $\neg At(P_1, SFO) \wedge At(P_1, JFK)$)

Example: Air cargo transport

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

The following plan is a solution to the problem:

$$
\begin{aligned}
&[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK), \\
&\ Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]
\end{aligned}
$$

Algorithms for Planning

- Forward (progression) state-space search
  - We can use search algorithms to start from the initial state, pick and apply actions to the current frontier states to reach and explore other states till we reach the goal state.
  - This is inefficient since we will need to explore a lot of useless states and actions.

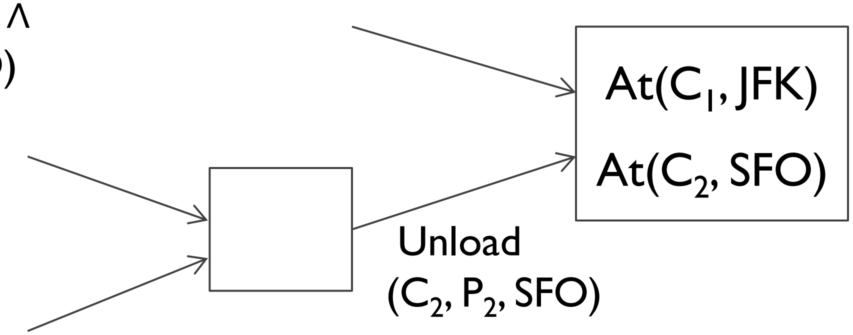
Algorithms for Planning

- Backwards (regression) relevant states search
  - We start at the goal state: $At(C_1, JFK) \land At(C_2, SFO)$
  - We can then pick actions that lead to such a state and trace backward until we reach the initial state. $At(C_1, SFO) \land At(C_2, JFK)$
  - We aim actions that are relevant—those that could be the last step in a plan leading up to the current goal state.
- We pick the first action backwards as:

$Action(Unload(C_2, P_2, SFO),$

$PRECOND: In(C_2, P_2) \land At(P_2, SFO) \land Cargo(C_2) \land Plane(P_2) \land Airport(SFO)$

$EFFECT: At(C_2, SFO) \land \neg In(C_2, P_2))$



Coding Logical Algorithms

- The imperative languages such as C/C++/Java/Python are not well suited to implement the algorithms with a first-order logic representation.
- Instead a declarative language such as Prolog is better suited to the task.

Coding Logical Algorithms

- In Prolog you specify your program as a set of declarations:

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

- Then, you can then make queries like:

```
?- sibling(sally, erica).
Yes
```