Logical Agents

Professor Marie Roch

Chapter 7, Russell & Norvig

Logical agents

Two key components

- Representation of the world
  - Background information
  - Percepts
- Ability to reason:
  Derive new information based on *inference*

Knowledge base (KB)

- Sentence – A statement about the agent's world
  - Based on sensors

    Danger around corner.

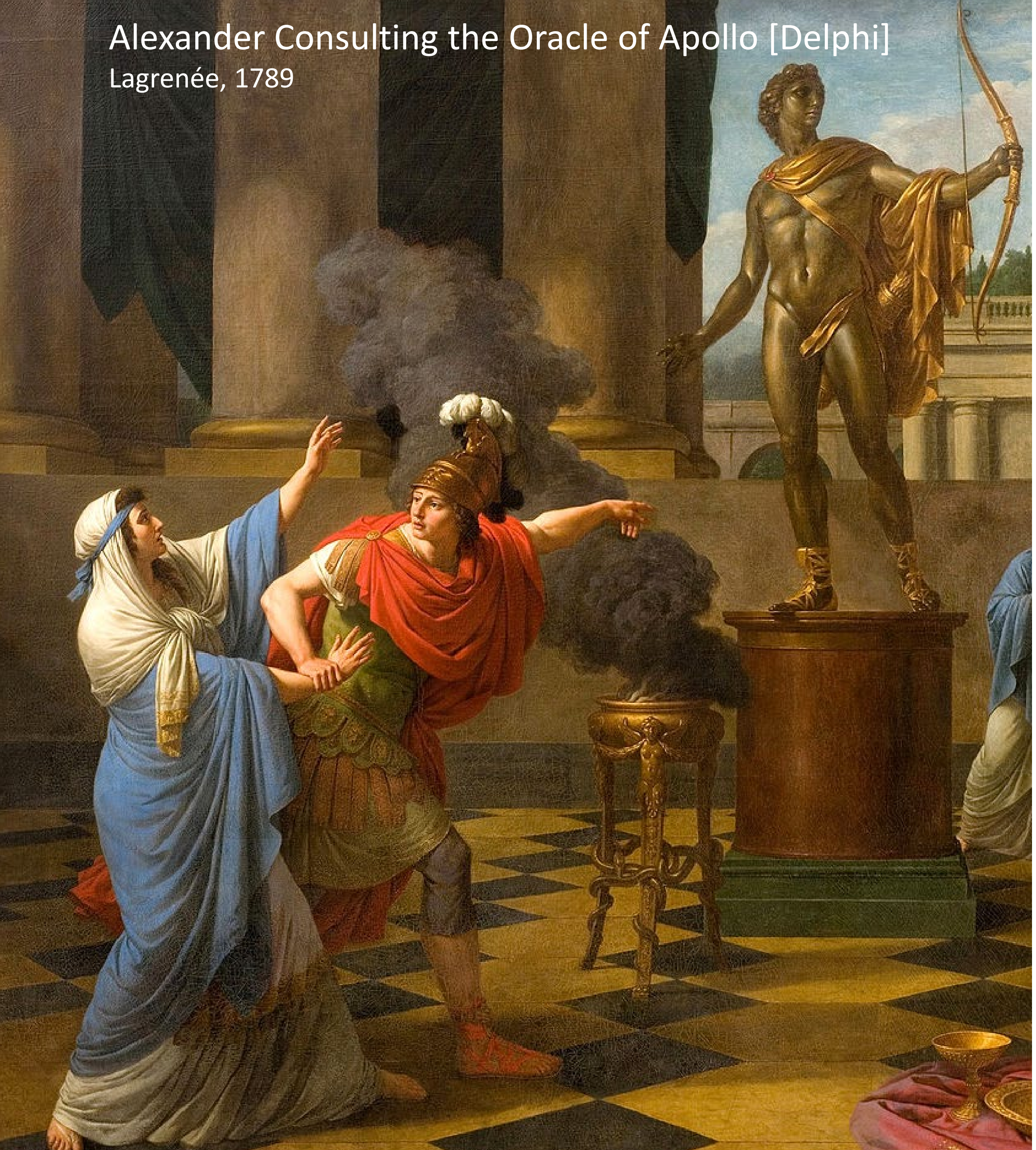
    Range to target is 50 m.

    Day is cloudy.
  - Based on background knowledge (or possibly learning)

    Solar cells lose efficiency on cloudy days.

    Ascending inclines requires more power.

image: invenergy.com


Alexander Consulting the Oracle of Apollo [Delphi]
Lagrenée, 1789

Knowledge base (KB) operations

Agent interactions with KB

- Tell – Agent informs KB of percepts
  Adds new *timestamped* sentences to the KB
- Ask – Agent asks KB which action should be taken. Reasoning occurs here.
- Tell – Agent tells KB that the action performed.

KB operations

We will use the following functions to build sentences:

- Make-Percept-Sentence – Construct a sentence from a percept.
- Make-Action-Query – Construct sentence that asks which action should be done at current time.
- Make-Action-Sentence – Construct sentence indicating that the specified action was executed at time t.

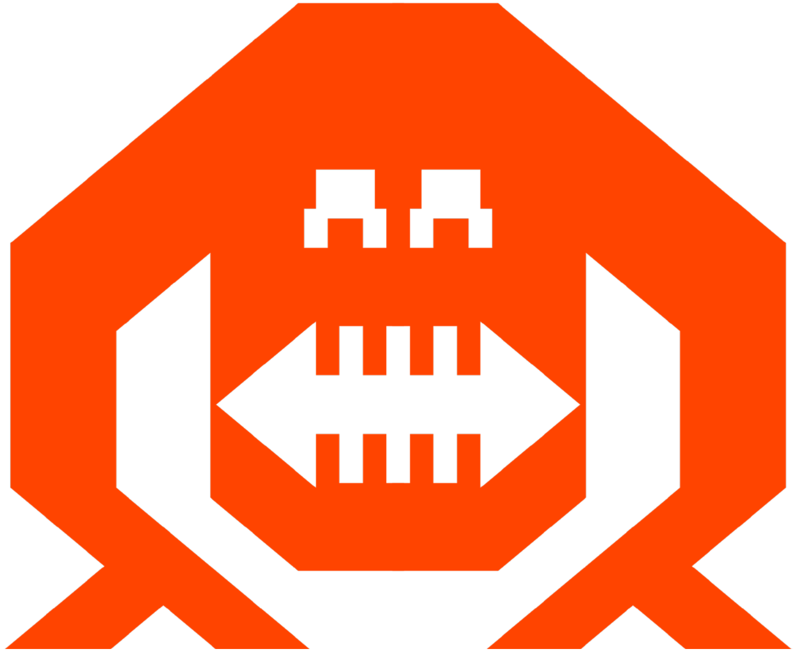
KB-agent

```
class KBagent:
   KB # knowledge base
   t = 0

   def getaction(percept):
      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 # caller performs action
```

t could be a real timestamp instead of a counter

Beware the wumpus



- Early computer game
  - Maze of dark caverns with
    - Pits – too dark, you might fall in
    - Wumpus – Sedentary beast.
      Does not move but it will eat you alive if you stumble into it…
    - Pot of gold
  - Your goal
    - Armed with a bow and single arrow, climb down into the caverns, grab the gold and get out without falling into a pit or being gobbled up by the wumpus.
    - Try not to shoot the wumpus, he's an apex predator and good for the environment.

Beware the wumpus

- Actions
  - Move *forward*
  - Turn left
  - Turn right
  - Shoot (can only do this once) – Arrow flies in the direction you fire until it hits a wall or kills the wumpus.
  - Grab – Grab the gold if it is in your current position.
  - Climb – Climb out of the case, only valid in starting cave.

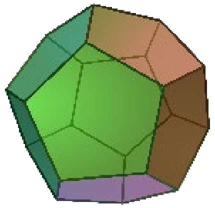
Sensors

- Chemical – Wumpus emits a *stench* that can be perceived in the next cavern
- Tactile
  - *breeze* is perceived when next to a pit
  - *bump* when agent walks into a wall
- Visual – All that glitters is gold, agent perceives *glitter* when in the same room as the gold.
- Auditory – The wumpus is usually pretty quiet, but emits a blood curdling *scream* when it dies.

Beware the wumpus

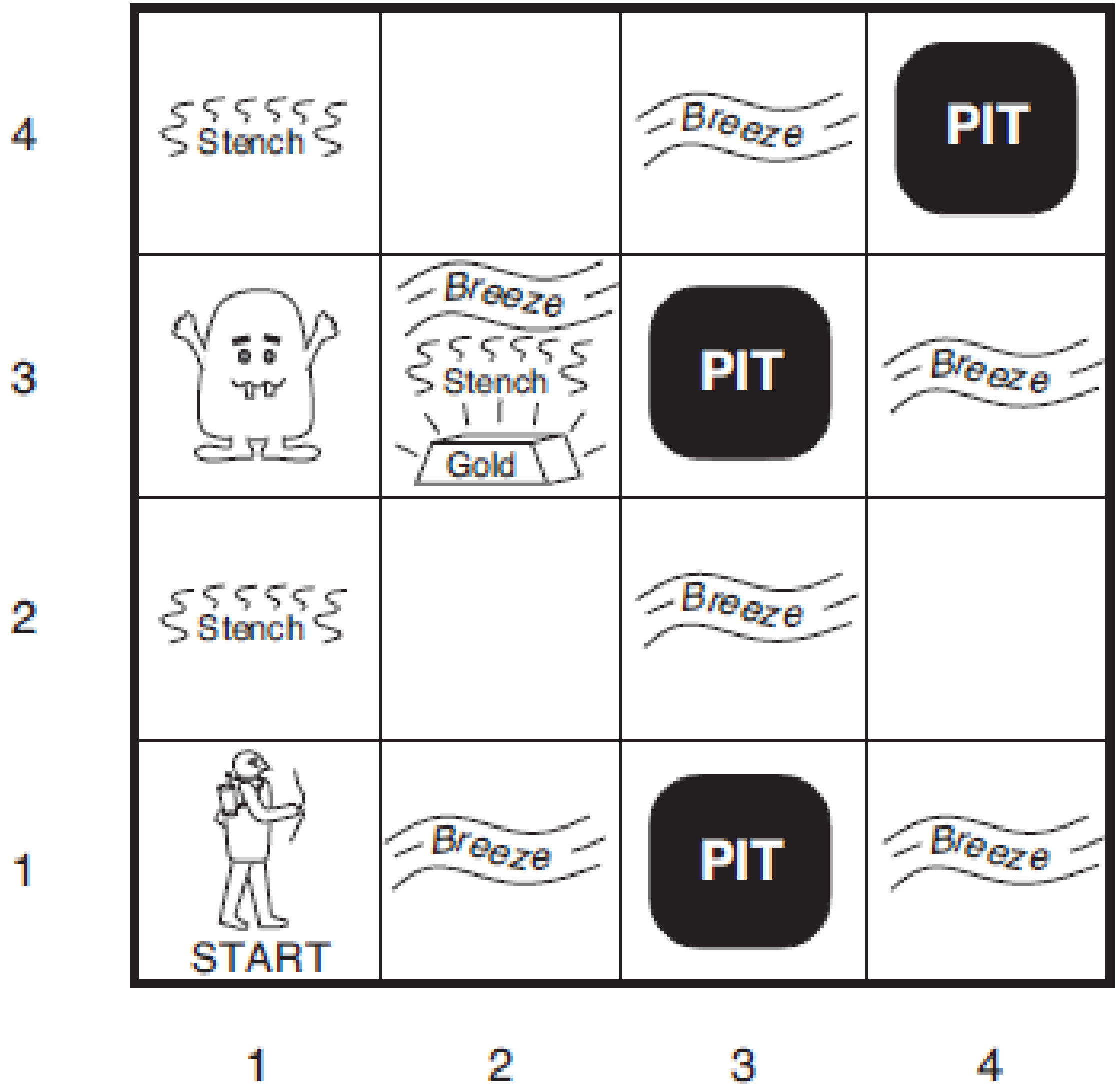
- Environment
  - 4x4 grid of caverns*
  - Grids other than the start square
    - P(pit) = .2
    - Wumpus placed randomly in non-pit, non-start cave
    - Gold randomly placed
  - Agent starts in x=1, y=1 facing such that they move positively along the y axis.

  - Some environments are unfair (e.g. gold in a pit or surrounded by pits ~ 21% of time)



*Written by Gregory Yob in the early 1970s, the cave topology was based on a dodecahedron, you can play a modified version online.

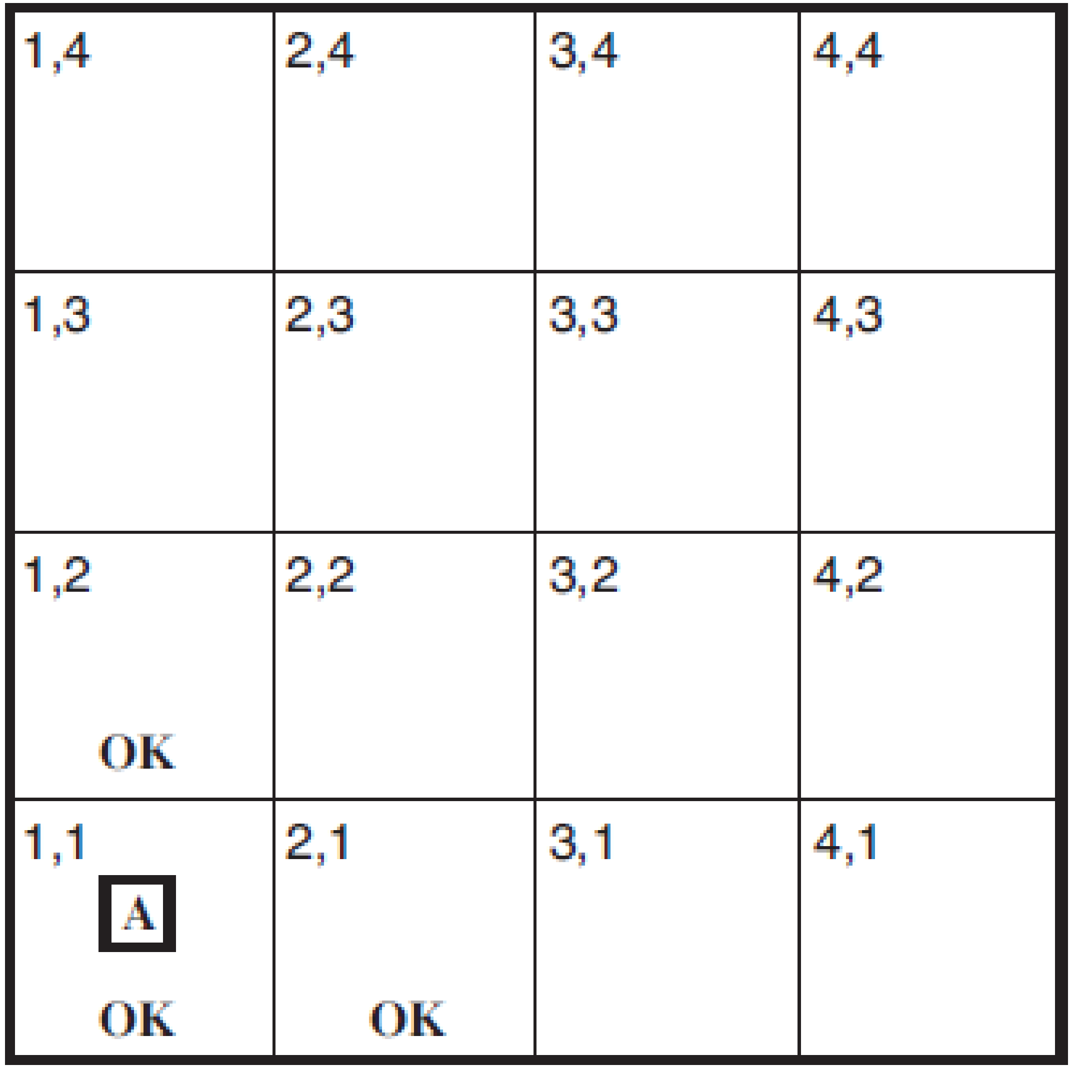
Welcome to wumpus world



Initial knowledge base

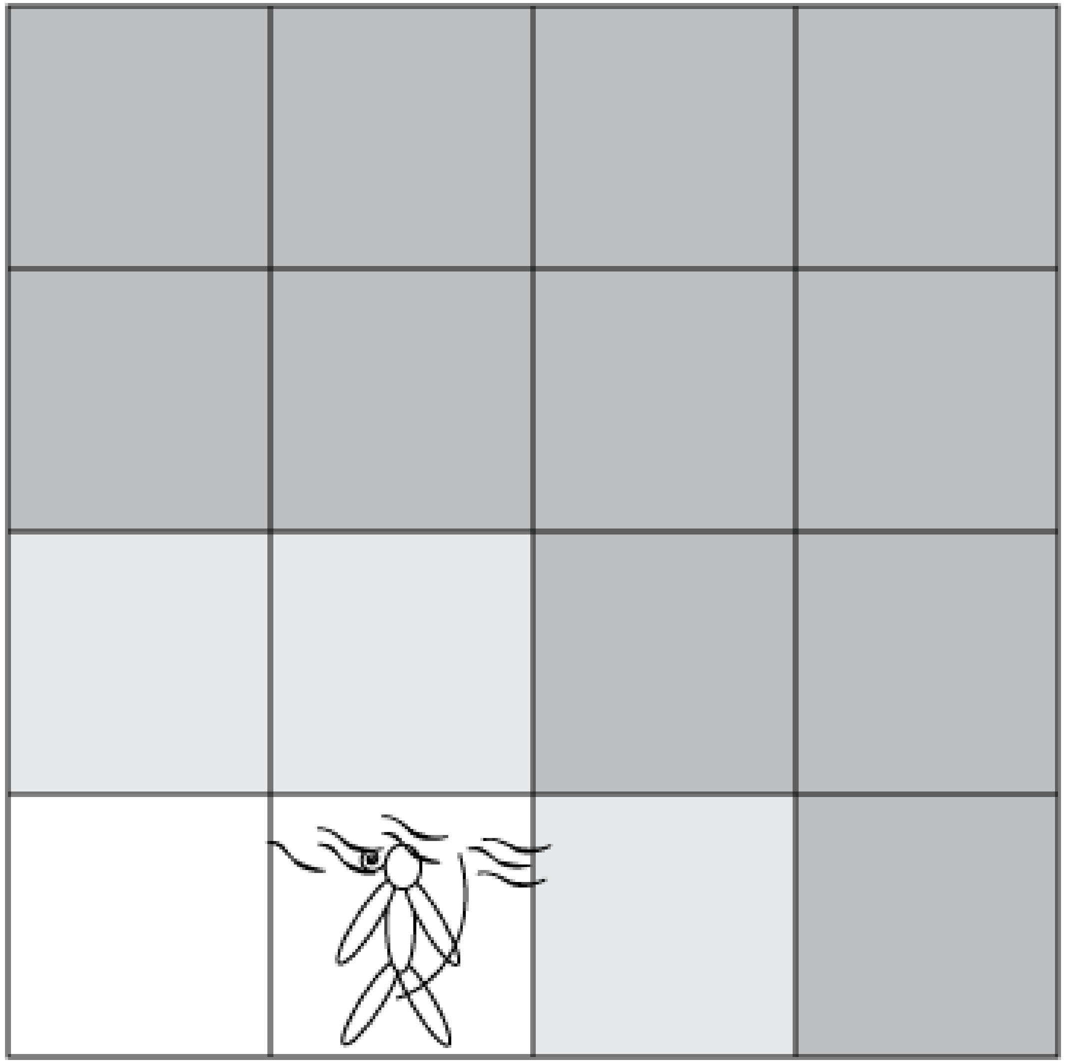
- Basic knowledge of environment (e.g. perceived a breeze next to a pit.
- Current location [1,1] is safe
- First percept:
  [No Stench, No Breeze, No Glitter, No Bump, No Scream]
- What can we learn?

[No Stench, No Breeze, No Glitter, No Bump, No Scream]



A - agent position
OK - safe

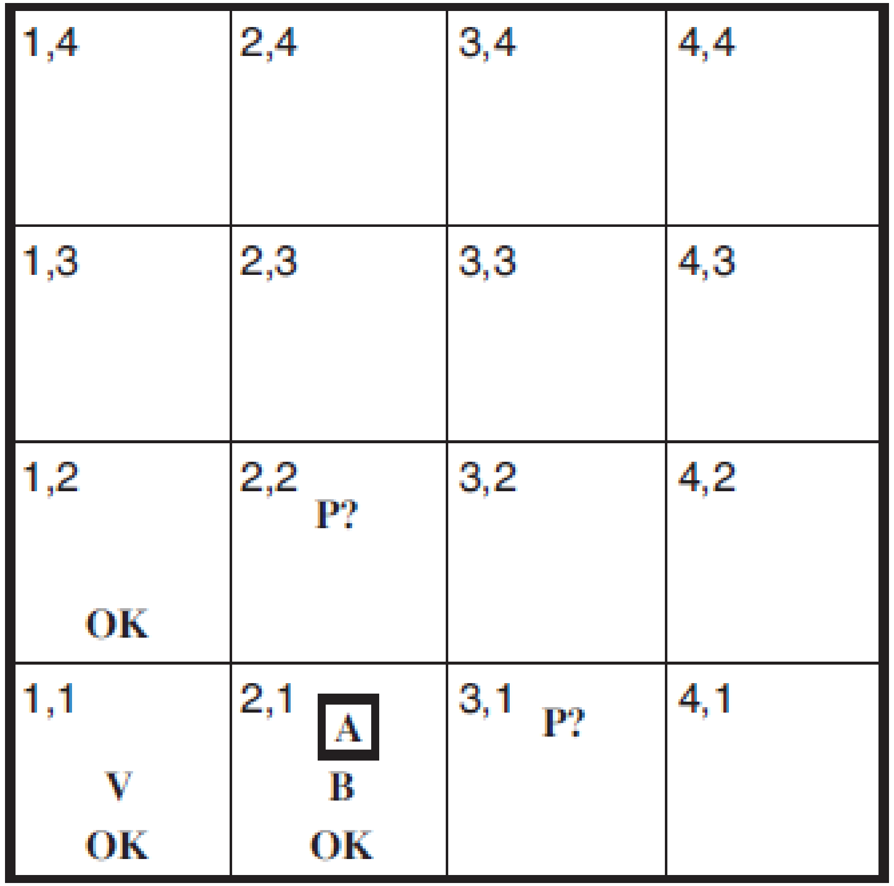
Agent moves to 2,1



2,1 Percepts
No Stench
Breeze
No Glitter
No Bump
No Scream

White caves – visited
Light gray caves – surrounding visited ones
Dark gray caves – The great unknown

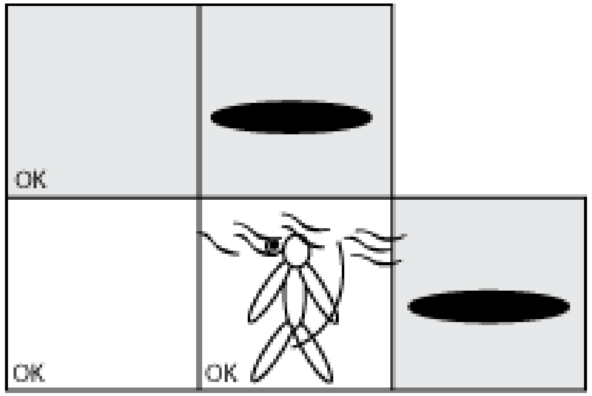
Danger approaches…



A - agent position
OK – safe
B – breeze
P? – possible pit

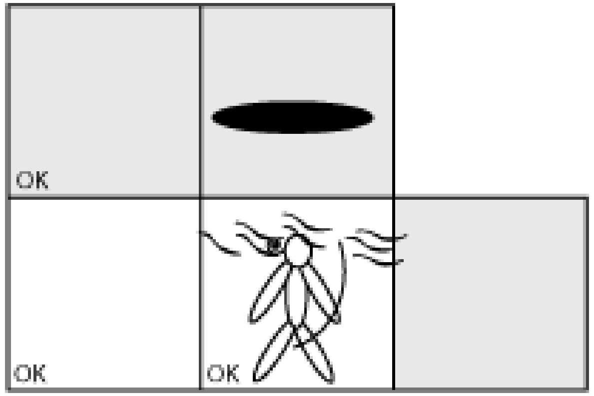
2,1 Percepts
No Stench
Breeze
No Glitter
No Bump
No Scream

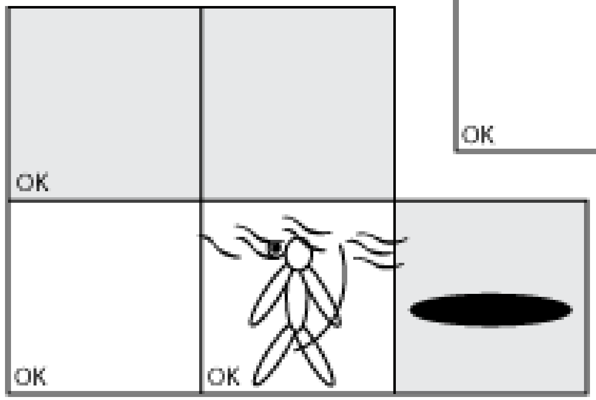
Knowledge base is extended based on percept breeze



Enumeration of instances

Three possible worlds



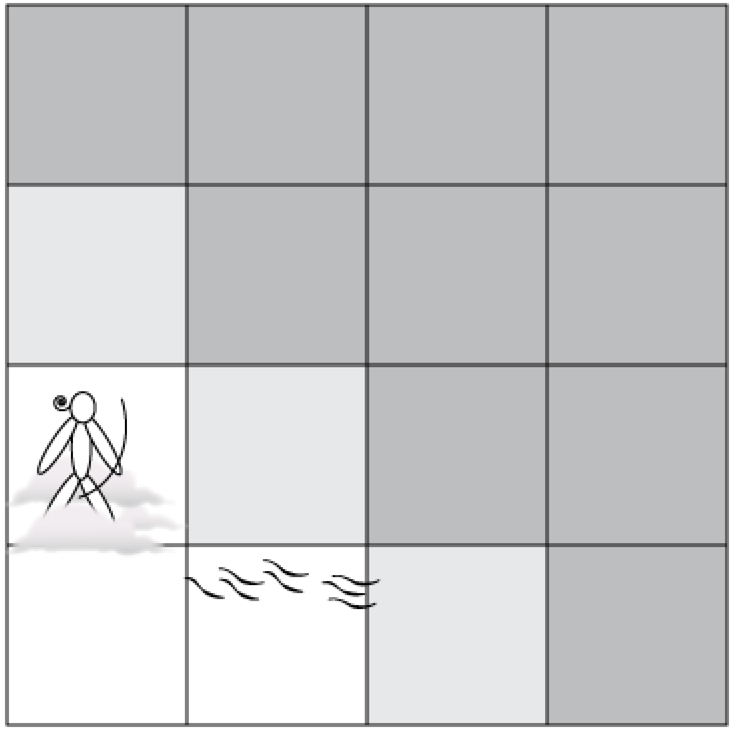


Playing it safe

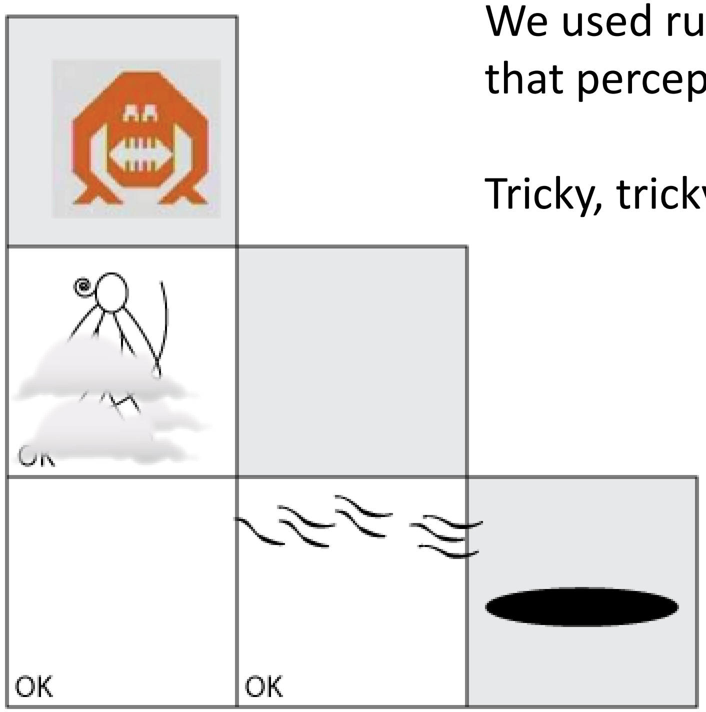
- We don’t know if it is safe to move into 3,1 or 2,2
- Move back to 1,1 then 1,2

I SMELL A WUMPUS!

- What can we learn based on our rules?



Logical deduction



We used rules to infer something that percepts did not tell us.

Tricky, tricky…

A little formality

- Models
  - Representation of world
  - Assignment of values to variables
  - All possible worlds consists of all possible assignments to model variables.
  - The knowledge base is some subset of the possible worlds.
- Sentences
  - Require syntax
  - Have semantics with respect to models.
    In most logical models: True/False

e.g. There is a pit at 3,1

Satisfaction and entailment

- Suppose α is true in model M, then we state:
  M satisfies α
  or equivalently: M is a model of α
- M(α) means all models that satisfy α.

- Reasoning – entailment

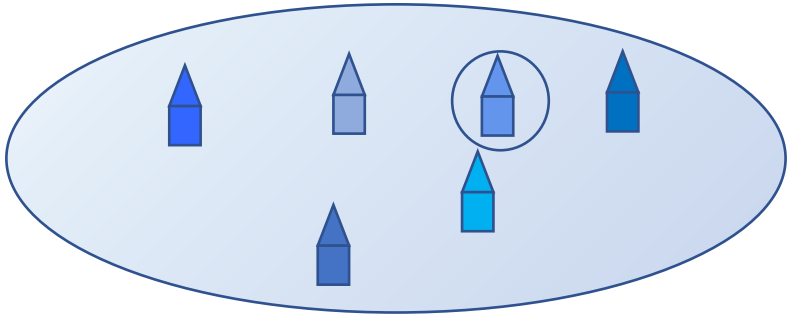
$$\alpha \vDash \beta \text{ if and only if } M(\alpha) \subseteq M(\beta)$$

α is a *stronger* assertion* than β; there may be more worlds associated with β
That is: β logically follows from α if $\alpha \vDash \beta$

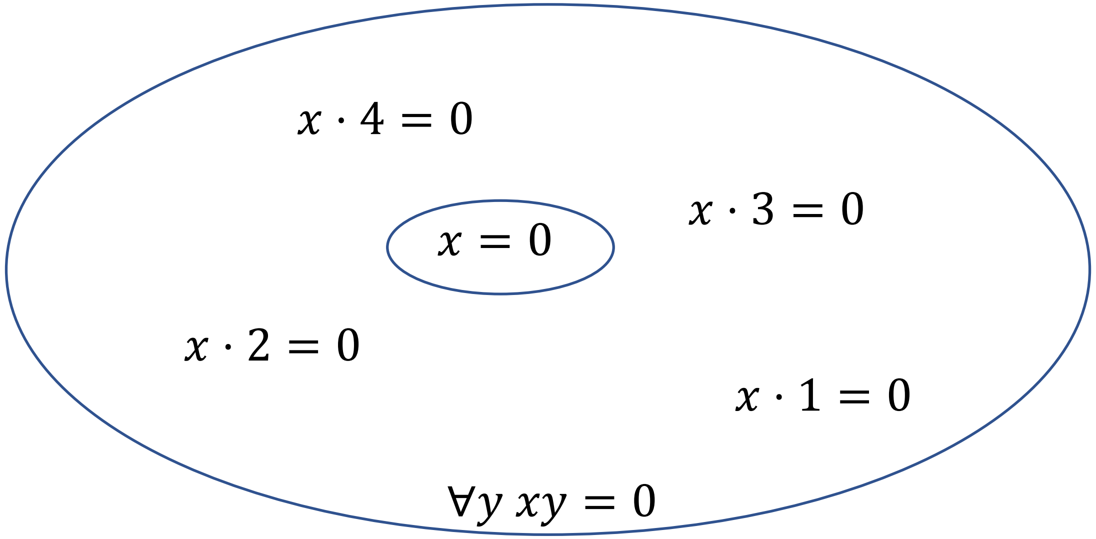
*stronger can be interpreted as tighter here

Examples of entailment

- House is cornflower blue ⊨ house is a shade of blue



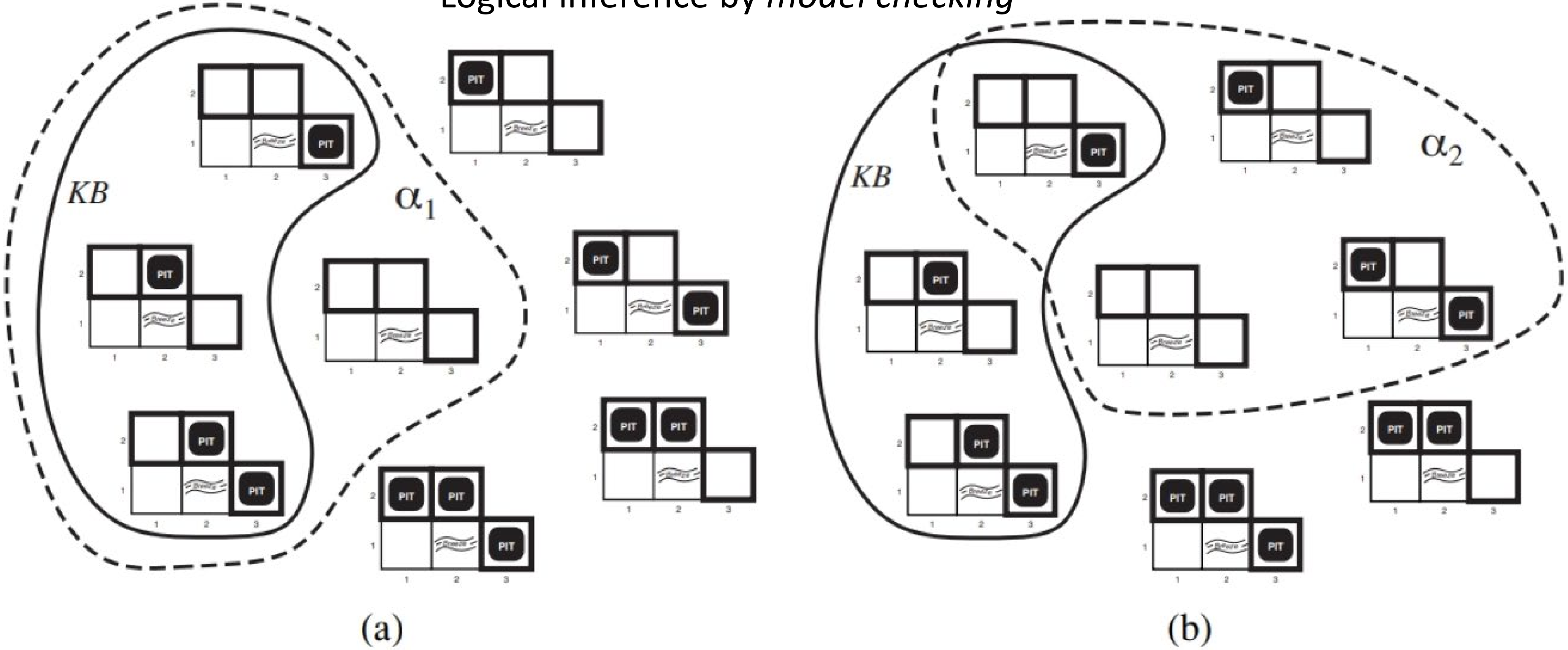
- $x = 0 \vDash xy = 0$



all possible pits in black squares, 2^3 possibilities…
KB shows what we know based on rules & percepts

RETURN TO WUMPUS WORLD

Logical inference by *model checking*



$$\alpha_1 = \text{No pit at } [1,2] \qquad \alpha_2 = \text{No pit at } [2,2]$$

Does KB |= α_1? KB |= α_2?

Inference algorithms…

- Are sound if inference only derives entailed sentences

  If our algorithm entailed KB $|= \alpha_2$, we might fall into a pit!

  She's a witch – not very sound…

- Are complete if they can derive any sentence that is entailed.

  Becomes an issue when the left-hand side of the entailment is infinite, e.g., $\alpha_1 |= \alpha_2$, α_1 is infinite

Inference algorithms

If our algorithm is sound, then our entailments are correct, but…

what connection does this have with reality?

Grounding is the correspondence between model and reality.

If the KB is well grounded, our entailments *should* follow in the real world.

Sentence construction: Propositional Logic

Propositional logic is for the most part the logic you learned to program with:

Sentence $\rightarrow$ AtomicSentence | ComplexSentence

AtomicSentence $\rightarrow$ true|false|Literal

ComplexSentence $\rightarrow$ (Sentence) | [Sentence]

| $\neg$ Sentence *negation*

| Sentence ^ Sentence *conjunction*

| Sentence v Sentence *disjunction*

| Sentence $\Rightarrow$ Sentence *implication*

| Sentence $\Leftrightarrow$ Sentence *biconditional*

operator priority of logical connectives is shown by order,
e.g. (A v B ^ C) implies (A v (B ^ C)), literal is a propositional symbol e.g. A

Sentence semantics

- Sentences are reduced to true or false with respect to a specific model.
- In the Wumpus cave
  - We might denote the presence or absence of a pit by literals indexed by location: $P_{x,y}$
  - Example: $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$ that have true/false values in any given model.
- Models must specify values for each proposition.
- To resolve a sentence: apply logical connectives to truth values (see Fig. 7.8 for truth tables)

Knowledge base rules

In the wumpus cave,

- denote pit & wumpus presence/absence by $P_{i,j}$ and $W_{i,j}$
- there is a breeze if and only if there is a neighboring pit:

$$B_{2,2} \Leftrightarrow P_{2,1} \vee P_{2,3} \vee P_{1,2} \vee P_{3,2}$$
$$B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1}$$

- there is a stench iff the Wumpus is in the next cavern

$$S_{2,2} \Leftrightarrow W_{2,1} \vee W_{2,3} \vee W_{1,2} \vee W_{3,2}$$
$$S_{1,1} \Leftrightarrow W_{1,2} \vee W_{2,1}$$

Stench and breeze sentences are true in *all* wumpus worlds.

Percepts

- There is no pit at 1,1
  $\neg P_{1,1}$
- There is no breeze at 1,1
  $\neg B_{1,1}$
- There is a breeze at 2,1
  $B_{2,1}$

Simple entailment algorithm

```
def TruthTable-Entails(KB, α):
   symbols = proposition-symbols in KB and α
   return TT-Check-All(KB, α, symbols, {})

def TT-Check-All(KB, α, symbols, model):
   if empty(symbols):
      if pl-true(kb, model):  # Does the KB entail the model?
         return pl-true(α, model)  # Does α entail the model?
      else return True  # return true when KB does not hold
    else:
      # recursively enumerate the models
      (s, others) = (first(symbols), rest(symbols))
      return TT-Check-ALL(KB, α, rest, model U {s=True}) and
             TT-Check-ALL(KB, α, rest, model U {s=False})
```

pl-true(a,b) is true if sentence a holds in model b

Concrete example:

- Knowledge base: $\neg P \vee Q, Q \Rightarrow M$
- Does $\neg P \vee Q, Q \Rightarrow M \vDash P \vee Q \vee M$?

P	Q	M	$\neg P \vee Q$	$Q \Rightarrow M$	$P \vee Q \vee M$
F	F	F	T	T	F
F	F	T	T	T	T
F	T	F	T	F	T
F	T	T	T	T	T
T	F	F	F	T	T
T	F	T	F	T	T
T	T	F	T	F	T
T	T	T	T	T	T

Concrete example:

- Knowledge base: $\neg P \lor Q$
- Does $\neg P \lor Q \vDash \neg(P \land \neg Q) \lor R$?

P	Q	R	$\neg P \lor Q$	$\neg(P \land \neg Q) \lor R$
F	F	F	T	T
F	F	T	T	T
F	T	F	T	T
F	T	T	T	T
T	F	F	F	F
T	F	T	F	T
T	T	F	T	T
T	T	T	T	T

```
def TruthTable-Entails(KB= ¬P ∨ Q, Q ⇒ M, α = P ∨ Q ∨ M):
   symbols = proposition-symbols in KB and α   # P, Q, M
   return TT-Check-All(KB, α, {P, Q, M}, {})

def TT-Check-All(KB, α, symbols, model):
   if empty(symbols):
      if pl-true(kb, model):   # Does model hold on left side of entailment?
         return pl-true(α, model)   # Is α entail the model?
      else return True   # entailment not affected when KB does not hold
    else:
      # recursively enumerate the models
      (s, others) = (first(symbols), rest(symbols))
      return TT-Check-ALL(KB, α, rest, model U {s=True}) and
             TT-Check-ALL(KB, α, rest, model U {s=False})
```

```
TT-Check-All(KB, α, {Q, M}, {P=T})
TT-Check-All(KB, α, {Q, M}, {P=F})
```

```
eventually for every model, e.g. {P=T,Q=T,M=T}
if pl-true(KB, {P=T,Q=T,M=T})
   return pl-true(P ∨ Q ∨ M,{P=T,Q=T,M=T})
   else return True
```

Simple entailment algorithm

Summary of model checking

- Recursively generates all possible combinations of truth values for every symbol.
- Checks if the knowledge base holds true for a world model (symbol value assignment)
  - If so, returns true/false showing whether sentence α also holds on model.
  - Otherwise returns true as we don't care about whether α holds outside the KB (implies).

Can we prove things without enumerating everything?

or, a refresher course in theorem proving…

Concepts

- Logical equivalence:
  if they are true in the same set of models

$$\alpha \equiv \beta$$

Alternative definition:
Equivalent if they entail one another:

$$\alpha \vDash \beta \Leftrightarrow \beta \vDash \alpha$$

Concepts

- Validity – Sentences are *valid* (called tautologies) if they are true in all models.
  e.g.

$$(a \wedge b) \vee \neg a \vee \neg b$$

a	b	$a \wedge b$	$\neg a$	$\neg b$	$(a \wedge b) \vee \neg a \vee \neg b$
F	F	F	T	T	T
F	T	F	T	F	T
T	F	F	F	T	T
T	T	T	F	F	T

- Deduction theorem:

$$\text{For arbitrary sentences } \alpha \text{ and } \beta, \alpha \vDash \beta \text{ iff } (\alpha \Rightarrow \beta) \text{ is valid.}$$

So if we can show $(\alpha \Rightarrow \beta)$ is a tautology, then we know $\alpha \vDash \beta$

Concepts

- Satisfiability – There exists some model such that a sentence is true.

- Sometimes, it is easier to show that something is valid by showing that its contradiction is not satisfiable:

- which leads to:

$$\alpha \text{ is valid iff } \neg\alpha \text{ is not satisfiable}$$
$$(\text{If no model satisfies } \neg\alpha,\ \text{then } \alpha \text{ must be true})$$

Proof by contradiction
(also known as proof by refutation)

$\alpha \vDash \beta$ iff $(\alpha\ \wedge \neg\beta)$ is not satisfiable
Remember, $\alpha\ \vDash \beta$ iff $\alpha \Rightarrow \beta\ \equiv \neg\alpha \vee \beta$

Inference rules

- Notation for rewriting logic

$$\frac{Sentence_1, Sentence_2}{Sentence_3}$$ means sentences 1 and 2 imply 3

- Rules
  - *Modus ponens*

    $$\frac{\alpha \Rightarrow \beta, \alpha}{\beta} \qquad \frac{\text{breeze} \Rightarrow \text{pit in neighboring cavern, breeze}}{\text{pit in neighboring cavern}}$$

  - And elimination

    $$\frac{\alpha \wedge \beta}{\alpha} \qquad \frac{\text{glitter} \wedge \text{breeze} \wedge \text{stench}}{\text{glitter}}$$

Notation and inference rules

- Rules
  - Biconditional elimination

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

$$\frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Note that we can also write these as equivalences

$$\text{e.g. } \alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$$

Inference rules

- Commutativity, associativity, and distributivity for
  $\wedge$ and $\vee$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$$

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$$

Inference rules

- Double-negation elimination

$$\neg(\neg\alpha) \equiv \alpha$$

- Contraposition

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$$

- Implication elimination

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$$

- DeMorgan's rule

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$$

Proof by hand

- Knowledge base

$$\neg P_{1,1},\quad B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1},\quad B_{2,1} \Leftrightarrow P_{1,1} \vee P_{2,2} \vee P_{3,1}$$

- Percepts (added to the knowledge base)

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

- Does the KB entail the lack of a pit at 1,2?

$B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1}$ from KB

$(B_{1,1} \Rightarrow P_{1,2} \vee P_{2,1}) \wedge (P_{1,2} \vee P_{2,1} \Rightarrow B_{1,1})$ biconditional elim.

$P_{1,2} \vee P_{2,1} \Rightarrow B_{1,1}$ and elimination

$\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$ contrapositive

$\neg(P_{1,2} \vee P_{2,1})$ from percept $\neg B_{1,1}$

$\neg P_{1,2} \wedge \neg P_{2,1}$ DeMorgan′s Rule

$\neg P_{1,2}$ and elimination ■ $KB \vDash (\alpha = \neg P_{1,2})$

Using inference

- Typically more efficient as we do not have to enumerate every possible value.
- Can be formulated as a search:
  - Initial State – Initial knowledge base
  - Actions – Inference rules applied to sentences
  - Results – Action result is the sentence rewritten by the inference rule
  - Goal – State containing the sentence we are trying to prove

Suppose we learn something new

- Suppose $KB \vDash \alpha$
- What if we learn β?

$$KB \wedge \beta \vDash \alpha$$

Propositional logic is a *monotonic* system, new knowledge will not change what is entailed; in other words: *propositional logic will not change its mind.*

Automated theorem proving

- Searching on logic rules works, but is complicated.
- Can we do something simpler?

- The resolution rule exploits disjunctive clause pairs where the literal is positive in one clause and negative in another:

$$\frac{l_1 \vee l_2 \vee \cdots \vee l_i \vee \cdots \vee l_{k-1} \vee l_k, m_1 \vee m_2 \vee \cdots \vee m_j \vee \cdots \vee m_n}{l_1 \vee l_2 \vee \cdots \vee l_{i-1} \vee l_{i+1} \ldots \vee l_{k-1} \vee l_k \vee m_1 \vee m_2 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

where $l_i \equiv \neg m_j$, e.g. $\mathrm{l_i} = \mathrm{P}, \mathrm{m_j} = \neg P$

example:

$$\frac{P_{1,1} \vee P_{3,1}, \neg P_{1,1} \vee \neg P_{2,2}}{P_{3,1} \vee \neg P_{2,2}}$$

Automated theorem proving

The resolution rule can produce duplicates:

$$\frac{A \vee B, A \vee \neg B}{A \vee A}$$

so we remove these by *factoring*

$$\frac{A \vee B, A \vee \neg B}{A}$$

We will assume that all results are factored.

Conjunctive normal form

All propositional logic sentences can be transformed to conjunctions of disjunctive clauses

Remove implications:

$$\alpha \Leftrightarrow \beta \rightarrow (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$$
$$\alpha \Rightarrow \beta \rightarrow \neg\alpha \vee \beta$$

Associate negations with literals only

$$\neg(\neg\alpha) \rightarrow \alpha$$
$$\neg(\alpha \vee \beta) \rightarrow \neg\alpha \wedge \neg\beta$$
$$\neg(\alpha \wedge \beta) \Rightarrow \neg\alpha \vee \neg\beta$$

Conjunctive normal form

- There are times when conjunctions cannot be eliminated: $\neg(A \vee B) \equiv \neg A \wedge B$
- As all sentences in the knowledge base must be true (conjunctions), we can treat these as separate sentences:
  - $\neg A$
  - B

Breeze/Pit Example

$$B_{1,1} \Leftrightarrow (P_{1,2} \lor P_{2,1})$$
$$\big(B_{1,1} \Rightarrow (P_{1,2} \lor P_{2,1})\big) \land \big((P_{1,2} \lor P_{2,1}) \Rightarrow B_{1,1}\big)$$
$$\big(\neg B_{1,1} \lor (P_{1,2} \lor P_{2,1})\big) \land \big(\neg(P_{1,2} \lor P_{2,1}) \lor B_{1,1}\big)$$
$$\big(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}\big) \land \big((\neg P_{1,2} \land \neg P_{2,1}) \lor B_{1,1}\big)$$
$$\big(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}\big) \land \big(\neg P_{1,2} \lor B_{1,1}\big) \land \big(\neg P_{2,1} \lor B_{1,1}\big)$$

Resolution algorithm

- Permits us to show KB $\models \alpha$ using only the resolution rule.
- All sentences must be in conjunctive normal form
- Resolution algorithm
  - Lets us show that something is satisfiable, that a model exists where a proposition is true
  - We want to show KB $|= \alpha$, but the resolution algorithm only shows us satisfiability.
  - Fortunately, $KB \models \alpha \Leftrightarrow KB \wedge \neg\alpha$
    so if we cannot satisfy α, we have a proof by contradiction.

Resolution algorithm

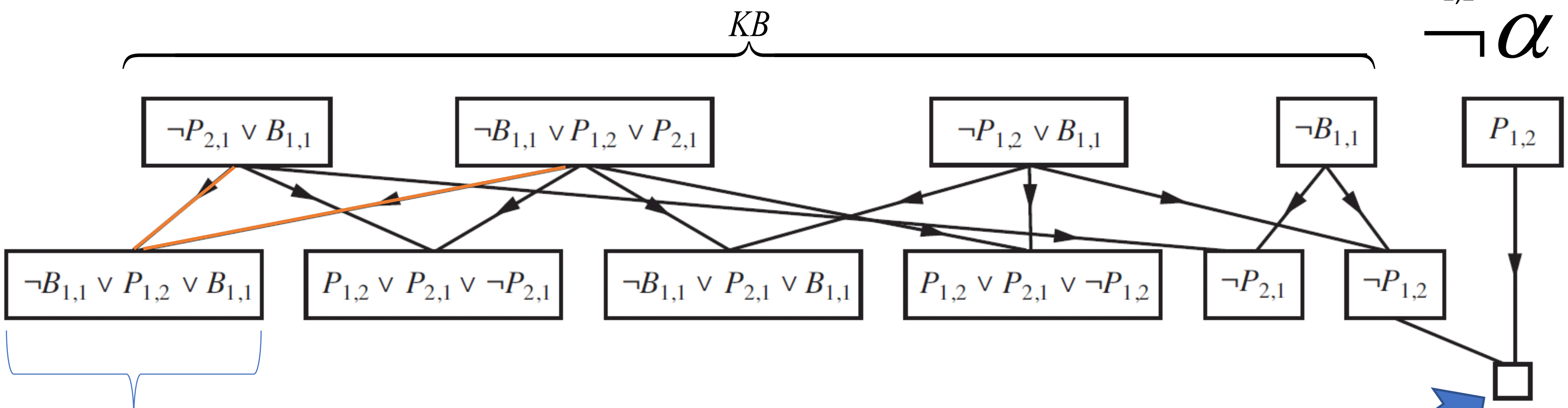
```
def PL-Resolution(KB, α)
    clauses = set of conjunctive normal form clauses of KB ∧ ¬α

    new = {}
    while True:
        for each distinct pair with complementary literals C_i, C_j ∈ clauses:
            resolvents = PL-Resolve(C_i, C_j)
            if ∅∈resolvents, return True  #could not satisfy, contradiction
            new = new U resolvents
        if new ⊆ clauses, return False   # everything satisfied
        clauses = clauses U new
```

There is no pit

$$\alpha = \neg P_{1,2}$$

Try to prove that there is a pit $P_{1,2}$



$$\frac{\neg P_{2,1} \lor B_{1,1}, \neg B_{1,1} \lor P_{1,2} \lor P_{2,1}}{\neg B_{1,1}, \lor P_{1,2} \lor B_{1,1}}$$

$$\frac{\neg P_{1,2}, P_{1,2}}{\emptyset}$$

Contradiction!

We sometimes denote this ϵ

Applications of the resolution rule, Fig. 7.13

Is the resolution theorem complete?

RC(S) – The resolution closure of S, is the set of all clauses derivable by repeated application of the resolution rule to S and its derivatives.

The ground resolution theorem states:

clauses S are unsatisfiable $\rightarrow \epsilon \in RC(S)$

Completeness of ground-resolution theorem

If the conditional:

$$S \text{ is unsatisfiable} \rightarrow \epsilon \in RC(S)$$

is true, then its contrapositive must also be true

$$\epsilon \notin RC(S) \rightarrow S \text{ is satisfiable}$$

Assume that we have a set, RC(S), that does not contain the empty clause. Can we construct a model (values for proposition symbols) that is true?

Let's try…

Ground-resolution theorem completeness

There are a finite number of proposition symbols in RC(S). For convenience, rename to: $P_1, P_2, P_3, \ldots, P_k$

$$
\text{for } i = 1 \text{ to } k
$$

$$
P_i = \begin{cases} \mathit{false} & \begin{array}{l}\text{when some clause in } RC(S) \text{ contains } \neg P_i \\ \text{and literals } P_1, P_2, \ldots, P_{i-1} \text{ all evaluate to false}\end{array} \\ \\ \mathit{true} & \text{otherwise} \end{cases}
$$

Ground theorem completeness

$$RC(S) = \{\neg P_3 \vee P_1, \neg P_1 \vee P_2 \vee P_3, P_2 \vee P_1, \neg P_1, P_2$$
$$\ldots \text{other clauses due to resolution}\}$$

- Example construction
  (This is only to paint broad strokes, it is based on the clauses shown above which are incomplete.)

$$P_1 = \mathit{false} \text{ as } \neg P_1$$
$$P_2 = \mathit{true} \text{ No } \neg P_2 \text{ that requires } \mathit{false} \text{ for clause to be true over } i = 1, 2$$
$$P_3 = \mathit{false} \; \neg P_3 \vee P_1$$

Ground theorem completeness

- Suppose RC(S) was not satisfiable
- At some step i, we could no longer satisfy the first i propositions.
- Up to this point, we were fine, so any $P_{1 \ldots i-1}$ in the clause with P_i must have evaluated to false
  - If P_i, we would assign true
  - If $\neg P_i$, would assign false
- Either case would be fine and we would proceed as normal.

Ground theorem completeness

- Hence there must be clauses of the form

$$false \vee false \vee \ldots \vee P_i \text{ and } false \vee false \vee \ldots \vee \neg P_i$$

which are not satisfiable.

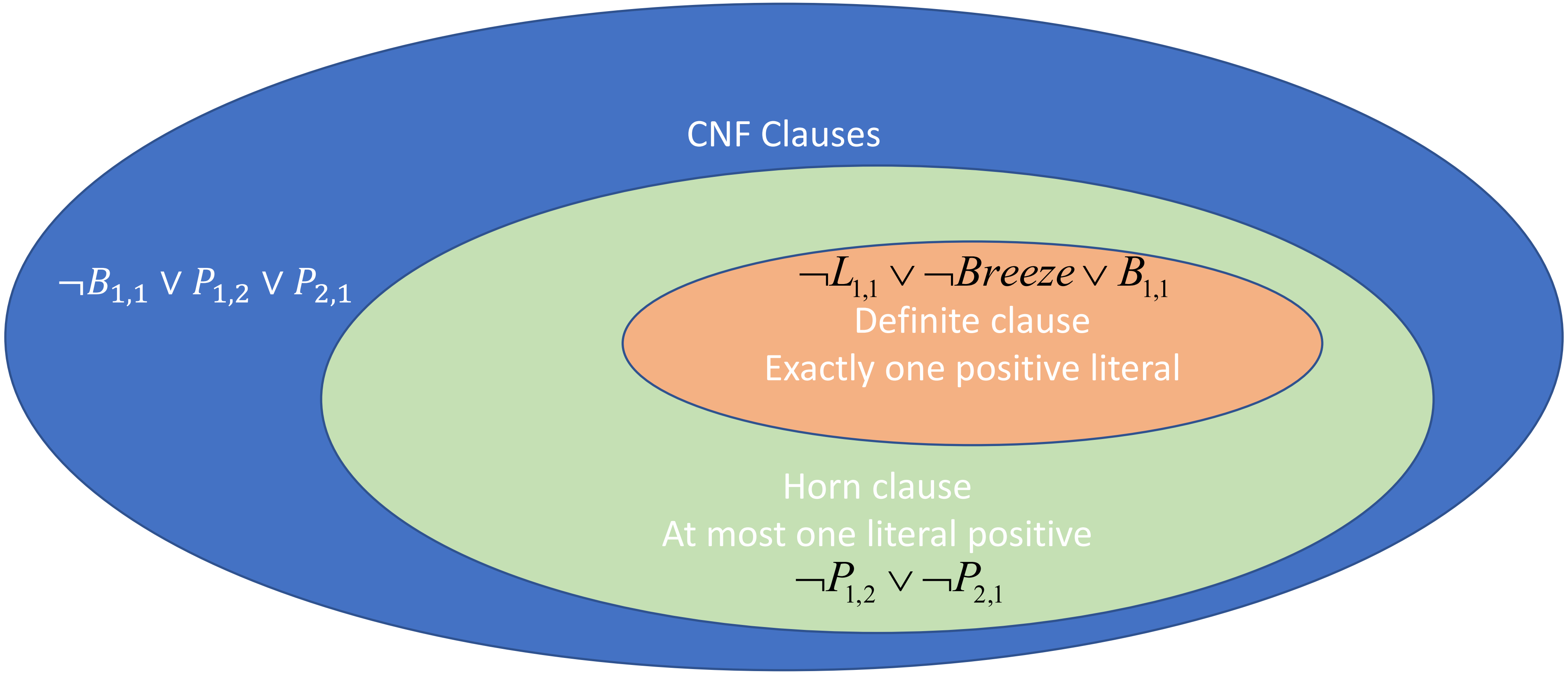
- But we know

$$\alpha \vee P_i \text{ and } \beta \vee \neg P_i \text{ resolve to } \alpha \vee \beta$$

which *must be in* RC(S), hence the failure would have had to occur earlier than i. So we must be able to satisfy S.

Restricted conjunctive normal form (CNF) clauses

Full power of resolution is not always needed



Restricted CNF clauses

- Definite and Horn clauses are interesting as there are more efficient resolution techniques.
- Many problems can be posed as Horn or definite clause problems:

$$\neg L_{1,1} \vee \neg Breeze \vee B_{1,1} \equiv (L_{1,1} \wedge Breeze) \Rightarrow B_{1,1}$$

- Horn clauses are usually written as implications and have names for clause parts.

$$\underbrace{Breeze}_{fact} \qquad \underbrace{(L_{1,1} \wedge Breeze)}_{body} \Rightarrow \underbrace{B_{1,1}}_{head}$$

$$\equiv True \Rightarrow Breeze$$

Why care about Horn clauses?

- Set of possible clauses in resolution closure of KB and α may be large
- We can prove Horn clause propositions in time that is linear with respect to the size of KB

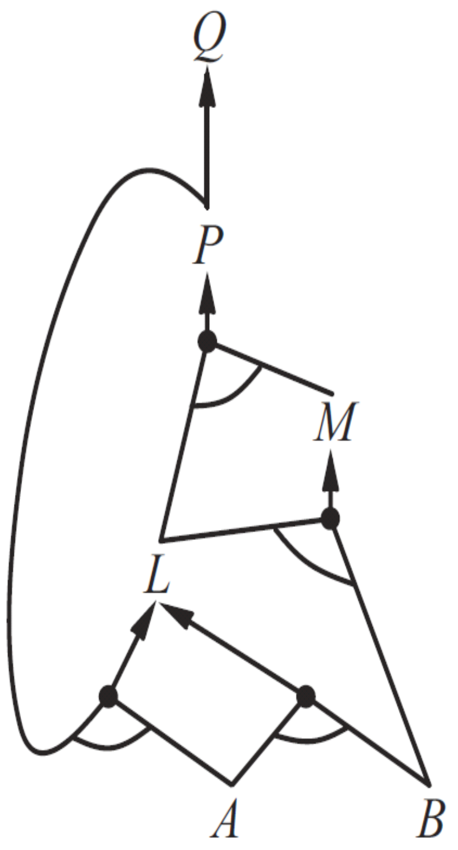
Road Runner fan art dohtem.com

Restricted CNF Clauses

- A Horn clause with no positive literal is called a *goal* clause.
- Efficient inference with Horn clauses is done through chaining.
  - forward-chaining – Data-driven, use known precepts to drive resolution of graph constructed from Horn clauses.
  - backward-chaining – Query driven, search and-or graph for goal.
- The language *Prolog* is a Horn clause processing language.

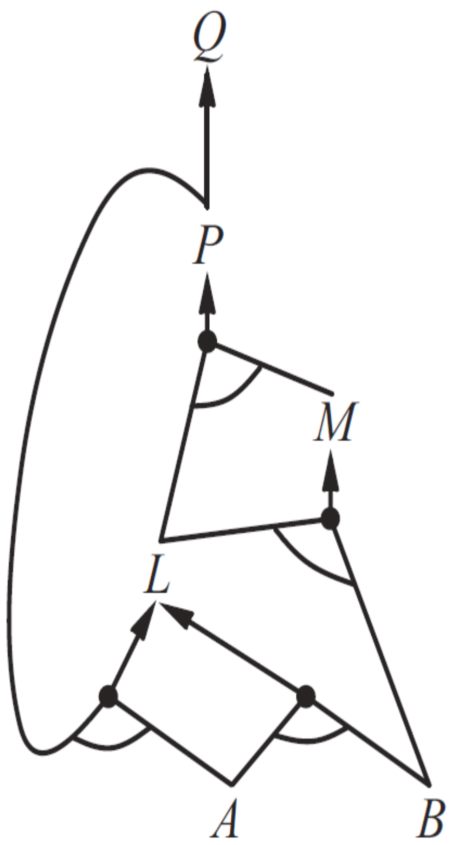
Horn clauses as graphs

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



Forward chaining

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

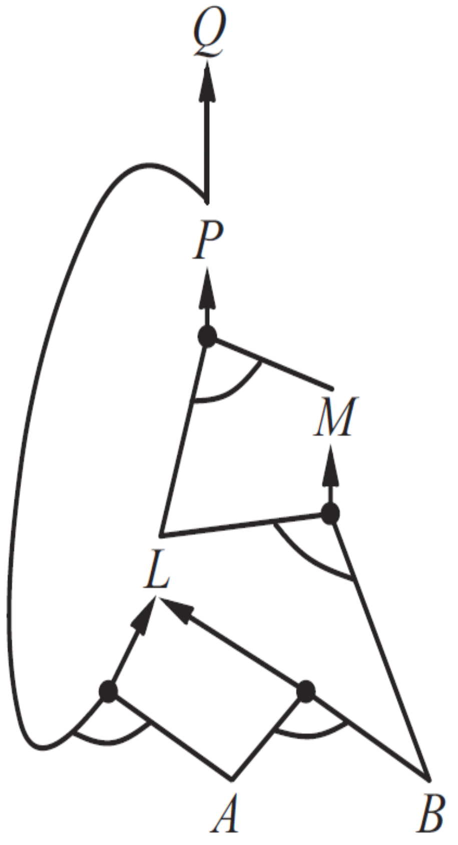


Prove L from A & B, continue propagating
Algorithm on p. 258, but you need only understand this intuitively.

Backward chaining

Goal directed reasoning – start with goal
Usually faster…

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



Start with query, e.g. Q and perform AND-OR search

Searching based on propositional model checks

- Recall $\alpha \vDash \beta \Leftrightarrow (\alpha \Rightarrow \beta \equiv \alpha \wedge \neg\beta)$
  so we work to satisfy $\neg\beta$ with respect to some knowledge base α.

- In effect, this is a constraint satisfaction problem and many of the techniques used for CSPs can be applied here. These can be applied to all CNF clauses:
  - Backtracking based solutions
  - Minimum conflict search solutions

Backtrack search

- Complete algorithm given in text (not responsible as similar in spirit to CSP backtrackers).
- General strategies for backtrack search
  - Components – When possible, partition into subproblems.
  - Variable/value ordering
  - Intelligent backtracking (e.g. conflict directed backtracking)
  - random restarts
  - Indexing to quickly find clauses with variables is highly useful.

Minconflict Local Search

- Same ideas as for CSPs
  - Assign all variables.
  - Pick conflicted variable and change to reduce conflicts until:
    - goal is satisfied *or*
    - maximum number of steps has been reached (failure)
- Drawbacks
  - Failure to find solution does not mean one does not exist.
  - If steps set to ∞ and a solution does not exist, algorithm will never return…

∴ usually used when solution known to exist

under- and over-constrained

- Under-constrained logic problems have lots of solutions and we can solve them quickly.
- Over-constrained logic problems have no solution.
- The hard ones are in between the two…

  as the ratio of clauses to available symbols increases, there comes a point where it becomes more difficult.

Putting it all together
Logical Agents

- Begin with knowledge base.
  For the Wumpus world
  - big long list of pits, breezes, and stenches (oh my):

$$B_{1,1} \Leftrightarrow P_{1,2} \vee P_{2,1}$$
$$S_{1,1} \Leftrightarrow W_{1,2} \vee W_{2,1}$$
$$B_{2,2} \Leftrightarrow P_{2,1} \vee P_{2,3} \vee P_{1,2} \vee P_{3,2}$$
$$S_{2,2} \Leftrightarrow W_{2,1} \vee W_{2,3} \vee W_{1,2} \vee W_{3,2}$$
$$\ldots$$

Tedious…. first order logic provides
a better way to do this Chapters 8 & 9

  - what we know about starting square and that we have an arrow

$$\neg W_{1,1} \wedge \neg P_{1,1} \wedge HaveArrow$$

Logical Agents

- Wumpus world continued
  - what we know about the wumpus:
    - At least one $W_{1,1} \vee W_{1,2} \vee W_{1,3} \vee W_{1,4} \vee W_{2,1} \vee \ldots \vee W_{4,4}$
    - No more than one – for every pair of locations at least one of them must not have a wumpus

$$\neg W_{1,1} \vee \neg W_{1,2}$$
$$\neg W_{1,1} \vee \neg W_{1,3}$$
$$\neg W_{1,1} \vee \neg W_{1,4}$$
$$\neg W_{1,1} \vee \neg W_{2,1}$$
$$\neg W_{1,1} \vee \neg W_{2,2}$$
$$\ldots$$
$$\neg W_{4,4} \vee \neg W_{4,3}$$

Percepts

- How do we incorporate percepts?
  - We might observe *Stench* at time 6, but not at time 7.
  - Remember, KB cannot change its mind! PL is *monotonic.*
  - We can modify MAKE-PERCEPT-SENTENCE to create distinct literals with time names: $Stench^6$ $\neg Stench^7$

Percepts

- We now have two types of variables:
  - Timestamped variables called *fluents*, and
  - *atemporal* variables that do not vary with time.

Should HaveArrow be a fluent or atemporal variable?

Actions and transition models

- Suppose agent initially at 1,1 facing east. An *effect axiom* could be added to model moving forward:

$$L_{1,1}^{0} \wedge FacingEast^{0} \wedge Forward^{0} \Rightarrow (L_{2,1}^{1} \wedge \neg L_{1,1}^{1})$$

- Other effect axioms could be added for:
  - grab, shoot, climb, turnleft, turnright.

Forward young gold hunter

So we move forward…

$$L_{1,1}^0 \wedge FacingEast^0 \wedge Forward^0 \Rightarrow (L_{2,1}^1 \wedge \neg L_{1,1}^1)$$

the effect axiom lets us determine that we are now in $L_{2,1}$:

$$Ask(KB, L_{2,1}^1) = True$$

What about?

$$Ask(KB, HaveArrow^1)$$

The Frame Problem

Nothing made HaveArrow^1 true…

the effects axiom only handled what changed.

We could add *frame axioms* for each time *t:*

$$Forward^0 \Rightarrow (HaveArrow^0 \Leftrightarrow HaveArrow^1)$$
$$Forward^1 \Rightarrow (HaveArrow^1 \Leftrightarrow HaveArrow^1)$$
$$Forward^2 \Rightarrow (HaveArrow^2 \Leftrightarrow HaveArrow^3)$$
$$\ldots$$
$$Forward^0 \Rightarrow (WumpusAlive^0 \Leftrightarrow WumpusAlive^1)$$
$$Forward^1 \Rightarrow (WumpusAlive^1 \Leftrightarrow WumpusAlive^1)$$
$$Forward^2 \Rightarrow (WumpusAlive^2 \Leftrightarrow WumpusAlive^3)$$
$$\ldots$$

more generally...

$$Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrow^{t+1})$$
$$Forward^t \Rightarrow (WumpusAlive^t \Leftrightarrow WumpusAlive^{t+1})$$

The Frame Problem

- When there are m actions and n fluents, there are O(mn) frame axioms for each time step.

- A better way to do this is to write *successor-state axioms* that shift the focus to the fluent and actions that might affect its state*:*

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$$

that is, F^{t+1} is true iff we caused it or it was true we did not take an action to falsify F.

Example: $Food^{t+1} \Leftrightarrow GrowFood^t \vee (Food^t \wedge \neg EatFood^t)$

Successor-state axiom example

- Hunter is at $L_{1,1}$ at time t+1:

$$L_{1,1}^{t+1} \Leftrightarrow (L_{1,1}^{t} \wedge (\neg Forward^{t} \vee Bump^{t+1}))$$
$$\vee (L_{1,2}^{t} \wedge (South^{t} \wedge Forward^{t}))$$
$$\vee (L_{(2,1)}^{t} \wedge (West^{t} \wedge Forward^{t})$$

- Hunter has arrow at time t+1:

$$HaveArrow^{t+1} \Leftrightarrow (HaveArrow^{t} \wedge \neg Shoot)$$

Convenience axioms

- Just as we abstract code with subroutines,
  we can define axioms to make our lives simpler:

$$OK^t_{x,y} \Leftrightarrow \neg P_{x,y} \wedge \neg (W_{x,y} \wedge WumpusAlive^t)$$

a cave location is safe if there's neither a pit or a live Wumpus at that location.

Logical Agents

- Pure logic agents simply try to prove a goal
  - Wumpus example sentence contains clauses for:
    - Init^0 – assertions about initial state
    - Transition^1, … Transition^t, successor-state axioms for all possible actions times 1,2,…,t.
    - Goal: HaveGold^t ^ ClimbedOut^t
  - If a model satisfies the clause, we can look at the action variables at each time and we have our plan.
  - Otherwise not possible in t steps.

Caveat: Not for partially observable environments, solver will simply set variables of hidden information to satisfy the problem

Logical Agents

- Hybrid agents - Use combination of logic, search and rule-based actions.

Hybrid Wumpus agent (partial)

```
hybrid-wumpus-agent(perceptlist, t) {
  Tell(KB, Make-Percept-Sentence(perceptlist, t))
  if t > 0:
    Tell(KB, successor state axioms for time t)
  safe = {[x,y] : Ask(KB, Ok^t_xy) = True}   # list of known safe caves

  # check goals by priority
  if Ask(KB, Glitter^t) = True:
    plan = [Grab] + PlanRoute(current, {[1,1]}, safe) + [Climb]
  else:
    unvisited = {[x,y] : Ask(KB, L^t'_x,y) = false ∀t'≤t }   # places we haven't been
    plan = PlanRoute(current, unvisited∩safe, safe)
  if not plan and Ask(KB,HaveArrow^t) == True:
    # no glitter or way to a safe square
    possiblewumpus = {[x,y] : Ask(KB, ¬W_x,y)==false}
    plan = plan-shot(current, possiblewumpus, safe)
```

Hybrid Wumpus agent (partial)

```
if not plan:
    # Couldn't move anywhere or shoot, take a risk
    mightbesafe = {[x,y] : Ask(KB, ¬OK^t_{x,y}) == False}
    plan = PlanRoute(current, unvisited ∩ mightbesafe, safe)
if not plan:
    # Can't get anywhere safely.  Call it a day
    plan = PlanRoute(current, {[1,1]}, safe) + [Climb]

action = pop(plan)
Tell(KB, Make-Action-Sentence(action, t))
t = t + 1
return action

PlanRoute(current, goals, allowed)
    # Plan a path from current to any goal through the set
    # of allowed nodes
    problem = RouteProblem(current, goals, allowed)
    return A* graph search(problem)
```