Logic

Chen-Yu Wei

Wumpus World

Performance

Gold +1000, death -1000, -1 per step, -10 for using the arrow

Environment

Perceive stench if adjacent to wumpus

Perceive breeze if adjacent to pit

Perceive glitter if in the square of gold

Can grab gold if in the square of gold

Can shoot and kill wumpus if you’re facing it (shooting uses up the only arrow)

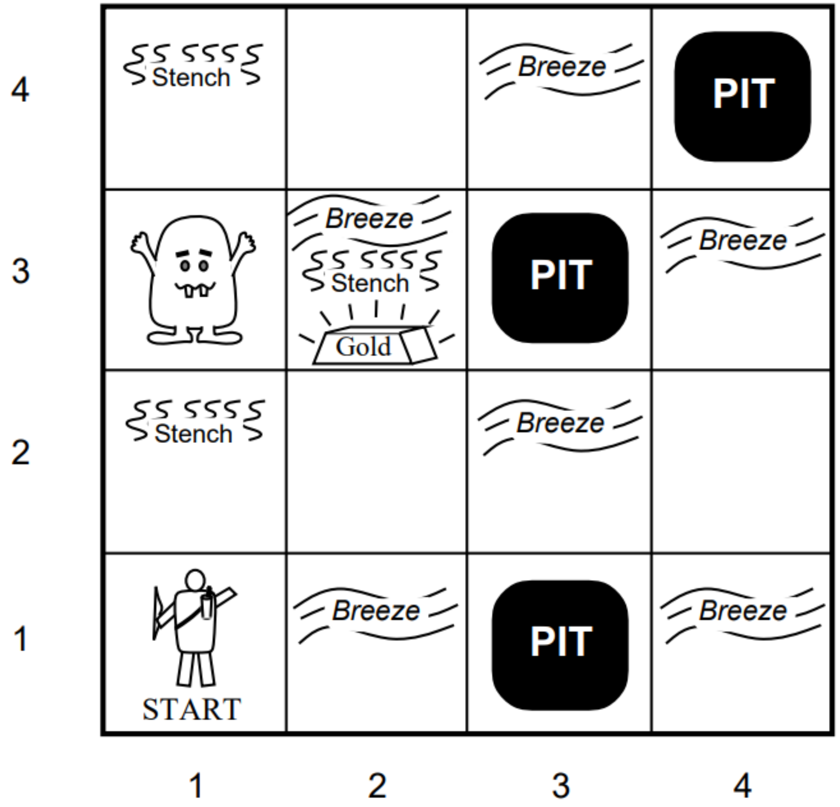
Die if entering a square with pit or living wumpus

Actions

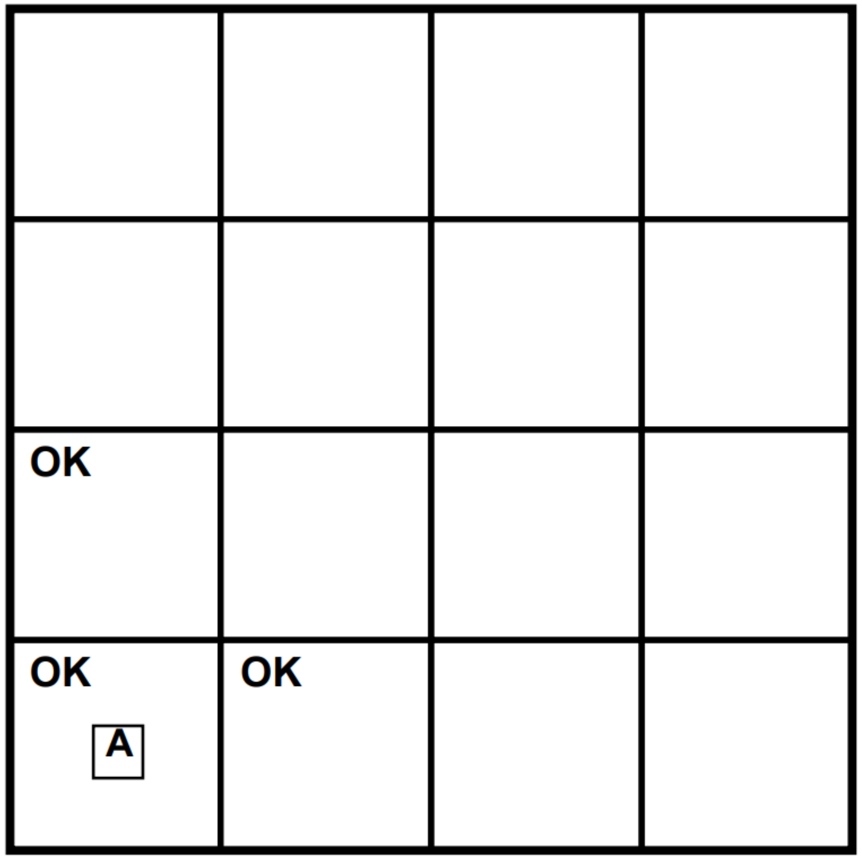
Left turn, right turn, forward, grab, shoot

Sensors

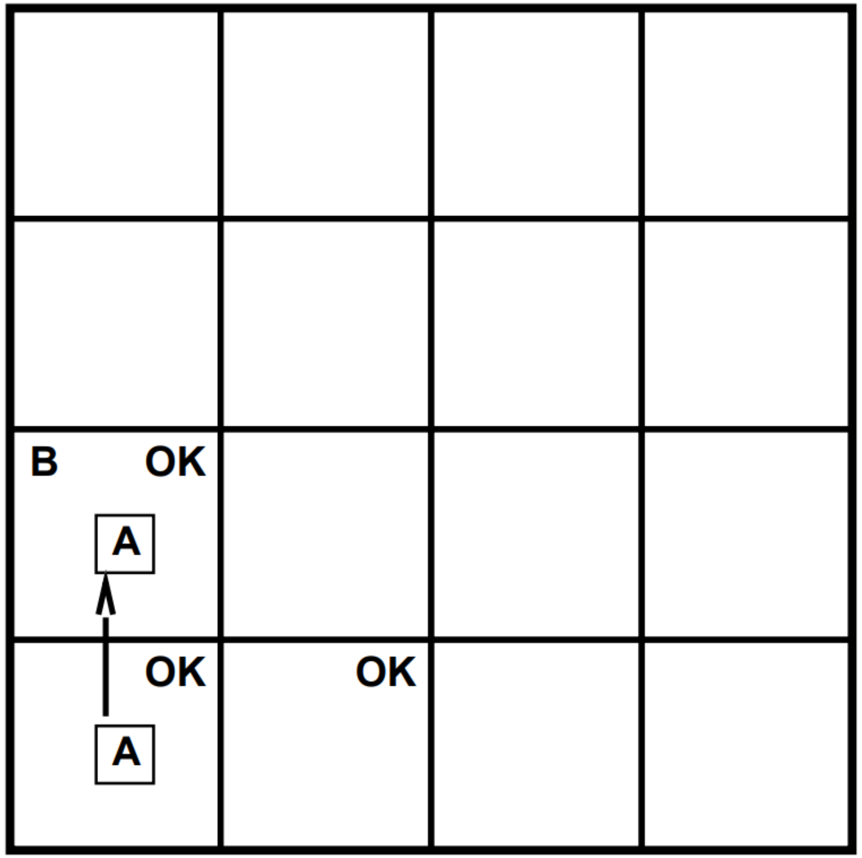
Breeze, glitter, smell



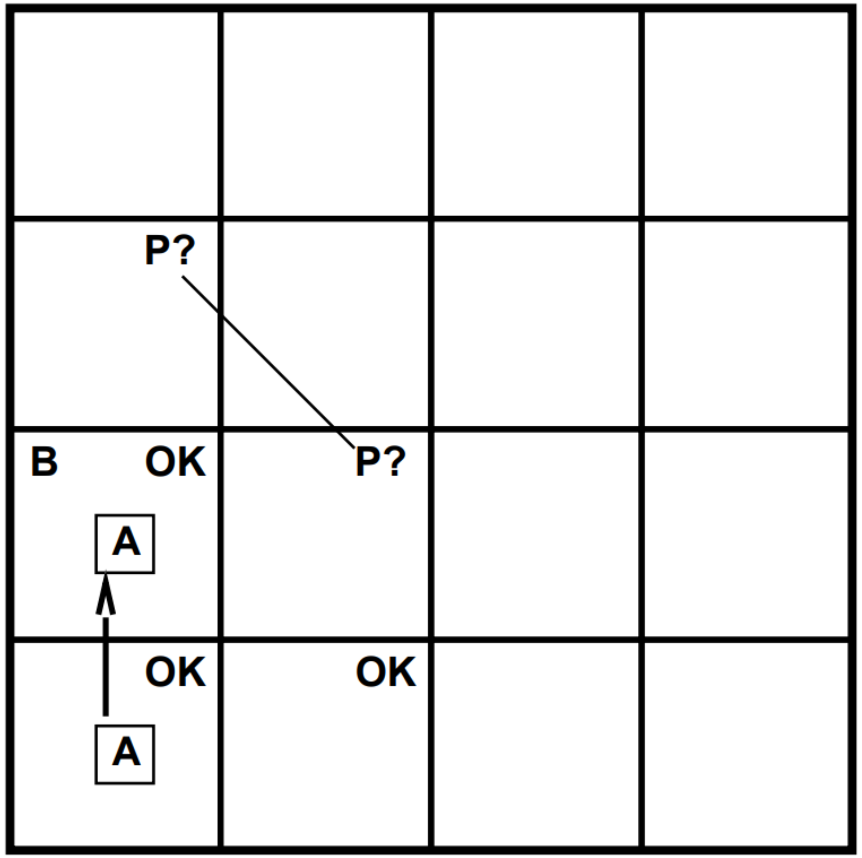
Exploring a wumpus world



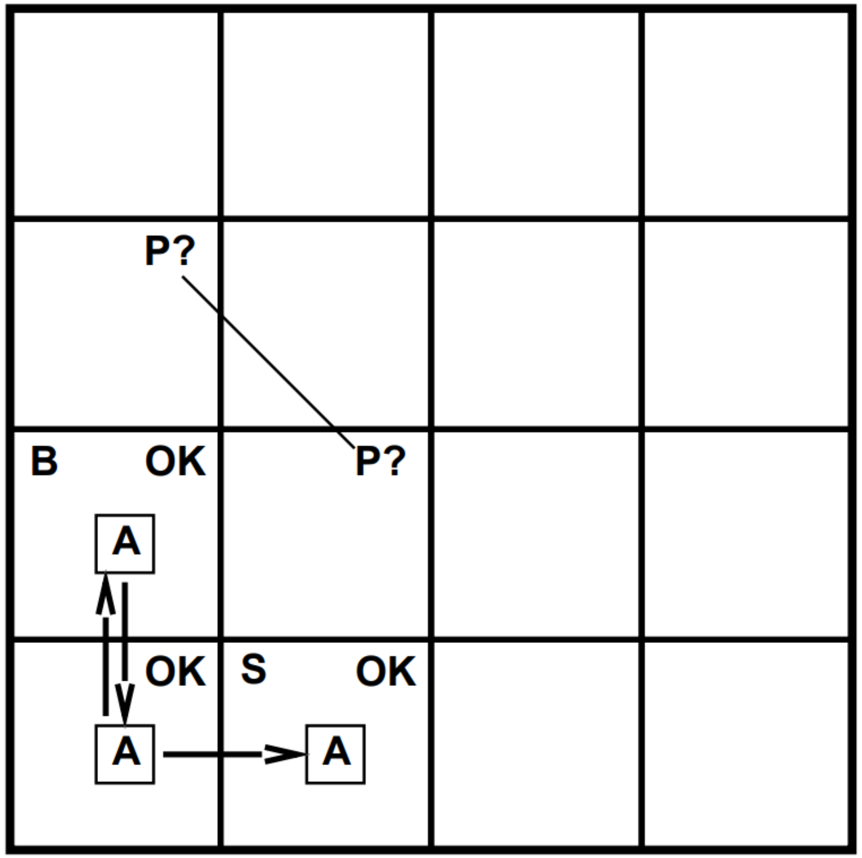
Exploring a wumpus world



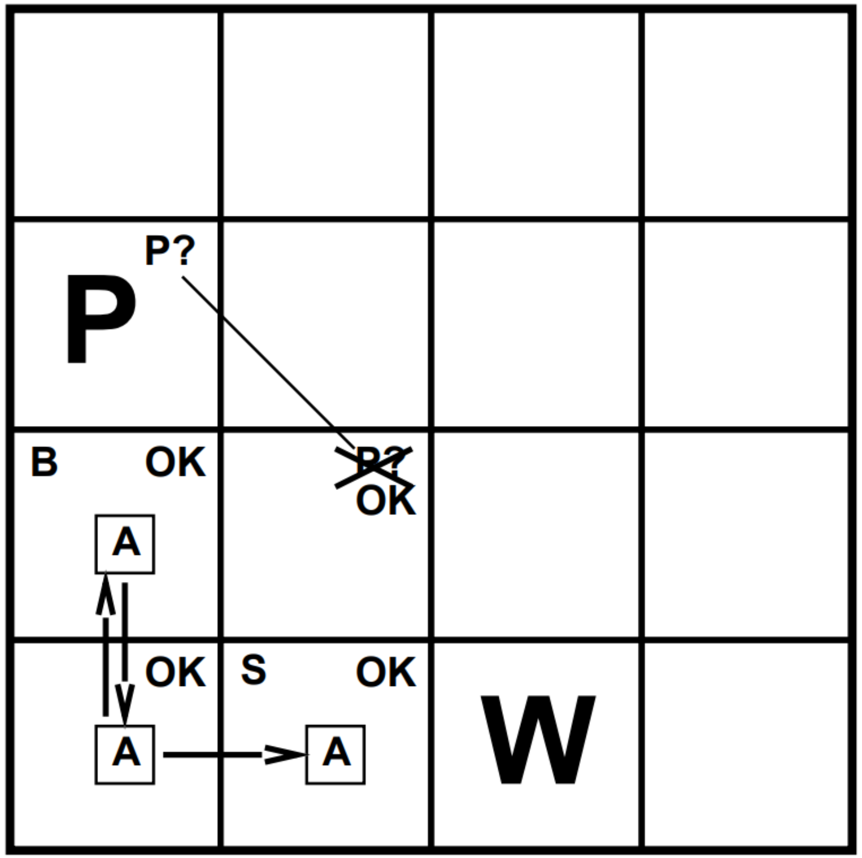
Exploring a wumpus world



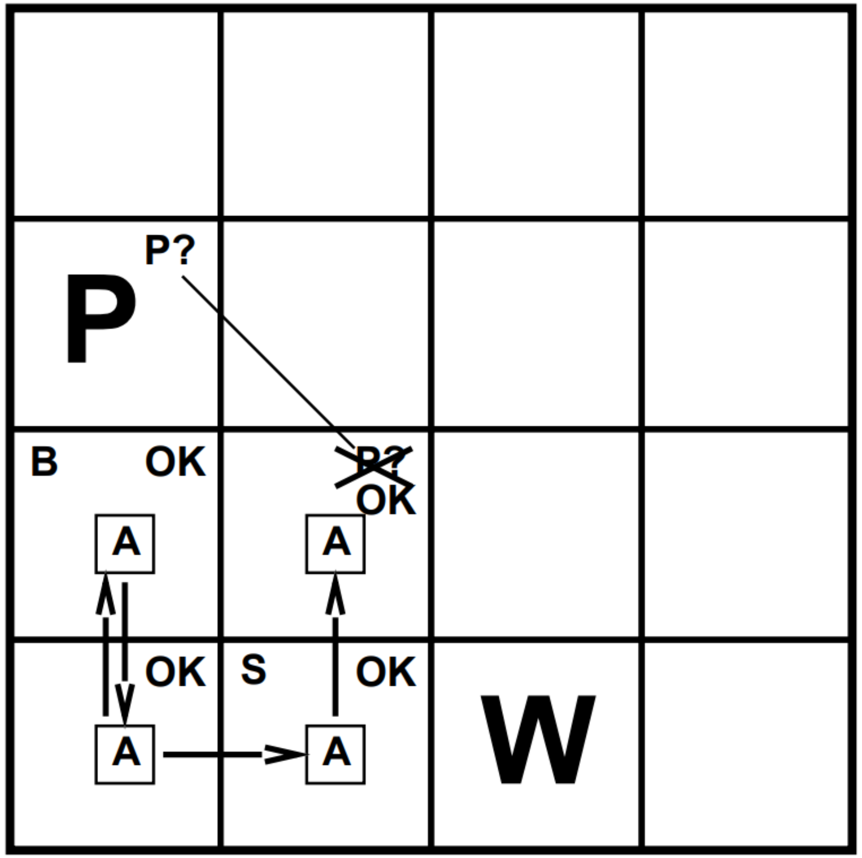
Exploring a wumpus world



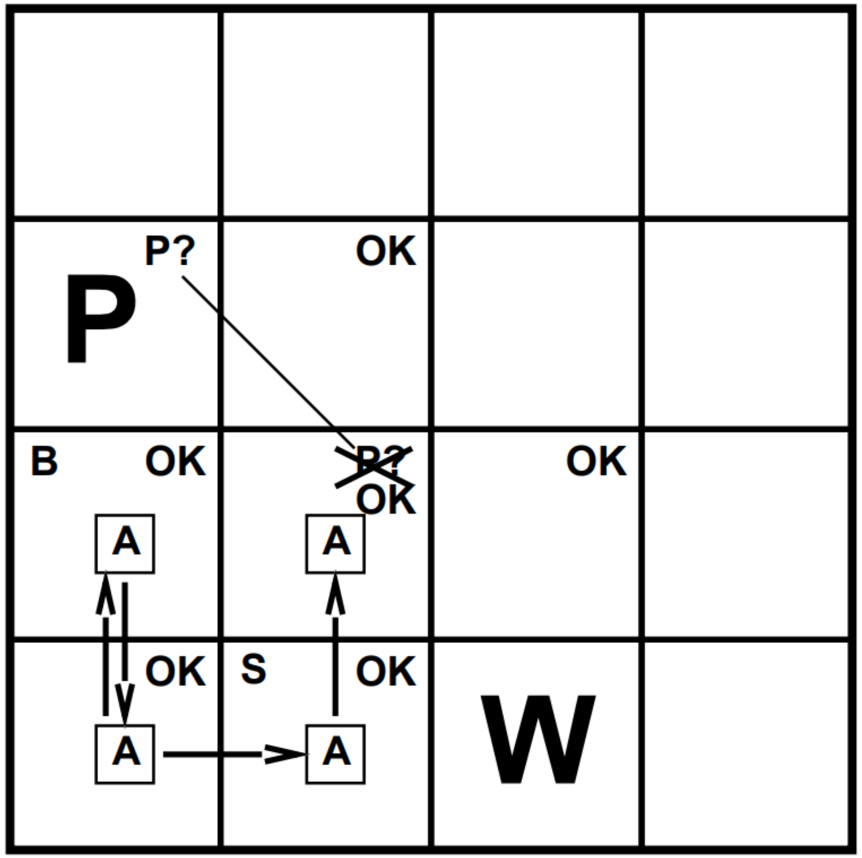
Exploring a wumpus world



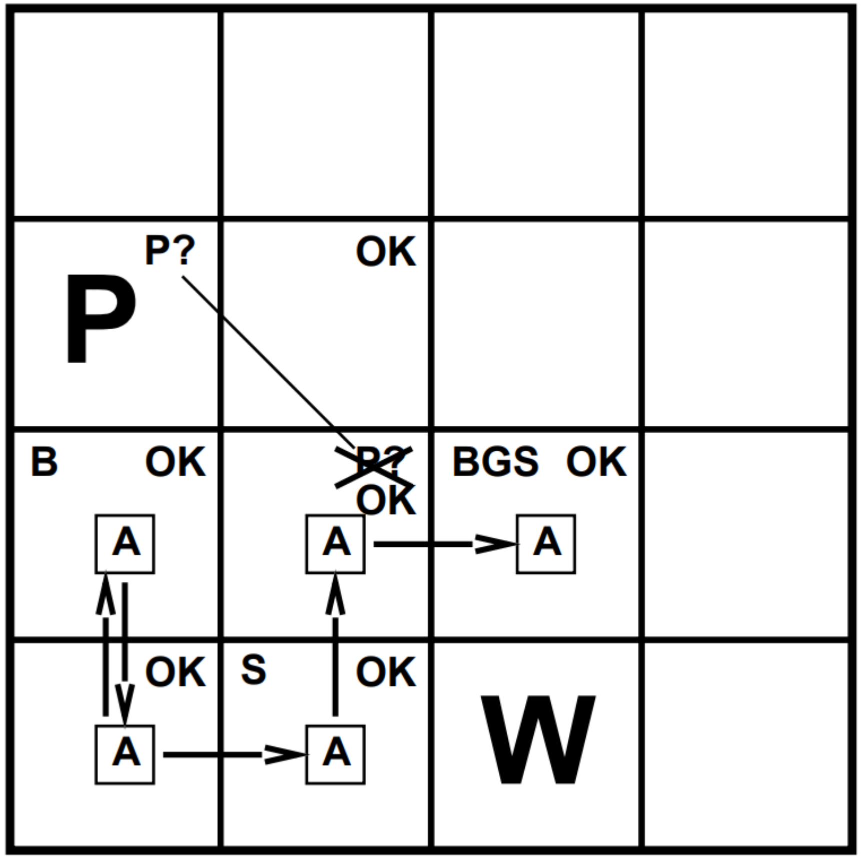
Exploring a wumpus world



Exploring a wumpus world



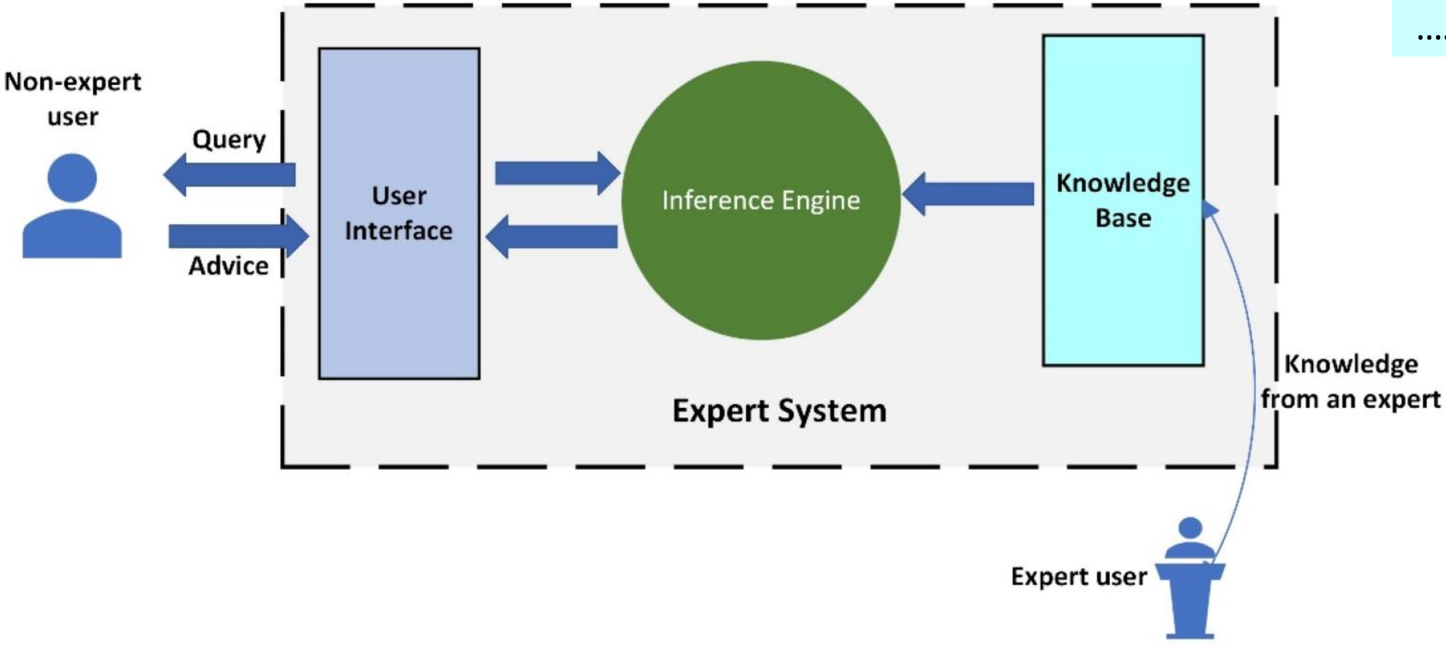
Exploring a wumpus world



Systems with Logical Reasoning

- Knowledge base
  - Consists of some prior knowledge
- Inference engine
  - Derive new knowledge or make some claims
- User Interaction
  - **Tell** information
  - **Ask** question

Example: Expert System



Knowledge base

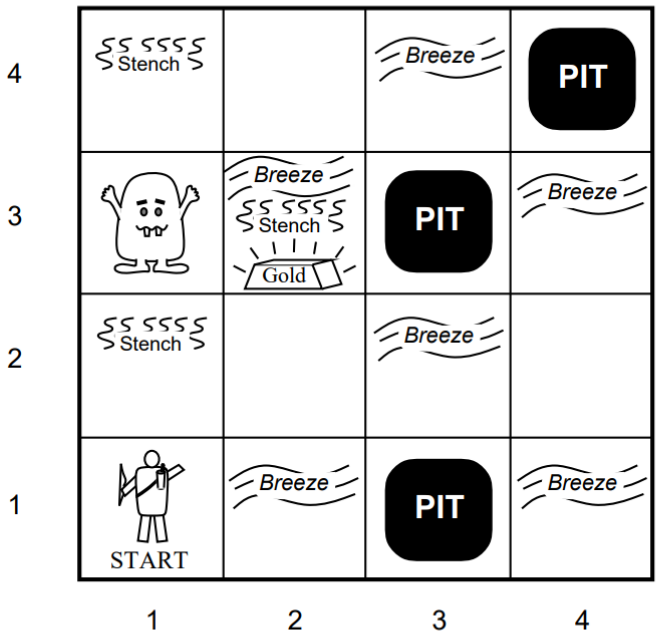
If **`has_hair`**, then **`mammal`**.
If **`mammal`** and **`has_hooves`**, then **`ungulate`**.
If **`has_feathers`**, then **`bird`**.
If **`mammal`** and **`carnivore`** and **`has_dark_spots`**, then **`cheetah`**.
If **`mammal`** and **`carnivore`** and **`has_black_stripes`**, then **`tiger`**.
If **`bird`** and **`does_not_fly`** and **`has_long_neck`**, then **`ostrich`**.
……

User interaction

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.6)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- go.
Does the animal have hair? yes.

Does the animal eat meat? |: no.

Does the animal have pointed teeth? |: no.

Does the animal have hooves? |: yes.

Does the animal have a long neck? |: yes.

Does the animal have long legs? |: yes.

I guess that the animal is: giraffe
true.

?- 
```

Example: wumpus world



Knowledge base

Perceive stench if adjacent to wumpus

Perceive breeze if adjacent to pit

Perceive glitter if in the square of gold

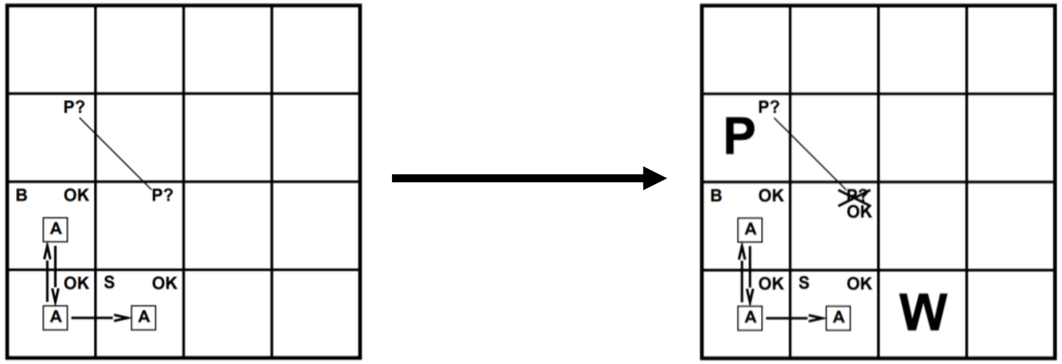
…

User interaction

Tell the logic system whether stench, breeze, glitter is perceived

Ask for the next action

Inference Engine



Ingredients of Propositional Logic

Sentence

Knowledge base consists of “sentences”

Inference algorithm derives new “sentences” and add them to the knowledge base

Example:

KB = { “Rain→Wet”, “Rain” }

Inference algorithm derives a new sentence “Wet” based on KB

Now KB becomes

KB = {“Rain→Wet”, “Rain”, “Wet” }

Ingredients of Logic – Syntax

Define what are valid sentences.

E.g., syntax in **python**:

“ for x in range(10): ”	Valid	
“ for x range(10): ”	Invalid	(the python interpreter cannot understand)

E.g. syntax in **math**:

“ $x + y = 5$”	Valid
“ $x\ 5 = y +$ ”	Invalid

Ingredients of Logic – Syntax

Syntax in **propositional logic**:

- A proposition symbols X is a sentence
  (a propositional symbol is a Boolean variable)
- If α is a sentence then $\neg\alpha$ is a sentence
- If α and β are sentences then $\alpha \wedge \beta$ is a sentence
- If α and β are sentences then $\alpha \vee \beta$ is a sentence
- If α and β are sentences then $\alpha \Rightarrow \beta$ is a sentence
- If α and β are sentences then $\alpha \Leftrightarrow \beta$ is a sentence

The $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ symbols have no meaning here. Their meanings are specified by the "semantics" of logic (discussed next).

Ingredients of Logic – Semantics

Let’s first define “models”. A model is a configuration of the world.

In propositional logic, a model is an **assignment of truth values** to propositional symbols.

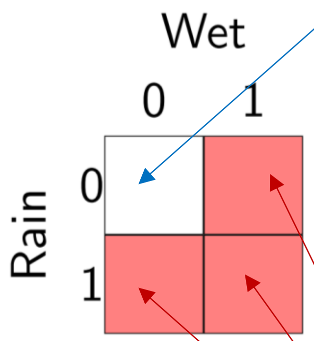
E.g., There are four possible models in the raining example:

		Wet	
		0	1
Rain	0		
	1		

Ingredients of Logic – Semantics

$f = \text{Rain} \vee \text{Wet}$

models where the sentence f is false



P	Q	(P v Q)
T	T	T
T	F	T
F	T	T
F	F	F

models where the sentence f is true

Ingredients of Logic – Semantics

P	~P
T	F
F	T

P	Q	(P ^ Q)
T	T	T
T	F	F
F	T	F
F	F	F

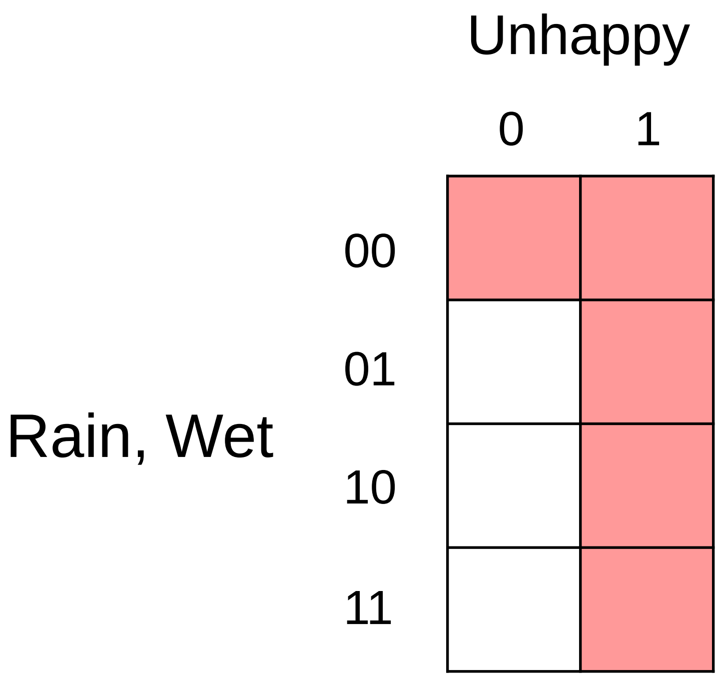
P	Q	(P v Q)
T	T	T
T	F	T
F	T	T
F	F	F

P	Q	(P =>Q)
T	T	T
T	F	F
F	T	T
F	F	T

P	Q	(P ⇔Q)
T	T	T
T	F	F
F	T	F
F	F	T

Ingredients of Logic – Semantics

f: (Rain ∨ Wet) ⇒ Unhappy

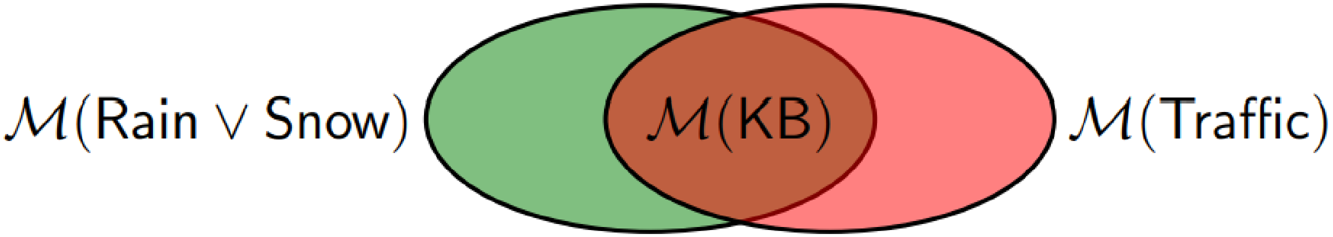


$\mathcal{M}(f)$: the set of models where sentence f is true.

Ingredients of Logic – Knowledge Base

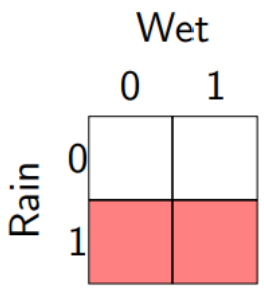
Knowledge base = a collection of sentences

Let $\text{KB} = \{\text{Rain} \vee \text{Snow}, \text{Traffic}\}$.

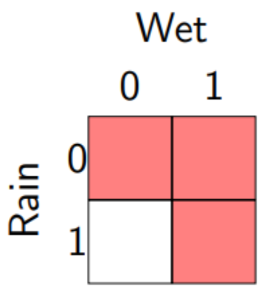


Ingredients of Logic – Knowledge Base

$\mathcal{M}(\text{Rain})$



$\mathcal{M}(\text{Rain} \rightarrow \text{Wet})$



Adding more formulas to the knowledge base:

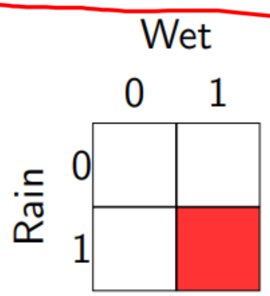
$$\text{KB} \longrightarrow \text{KB} \cup \{f\}$$

Shrinks the set of models:

$$\mathcal{M}(\text{KB}) \longrightarrow \mathcal{M}(\text{KB}) \cap \mathcal{M}(f)$$

KB

$\mathcal{M}(\{\text{Rain}, \text{Rain} \rightarrow \text{Wet}\})$



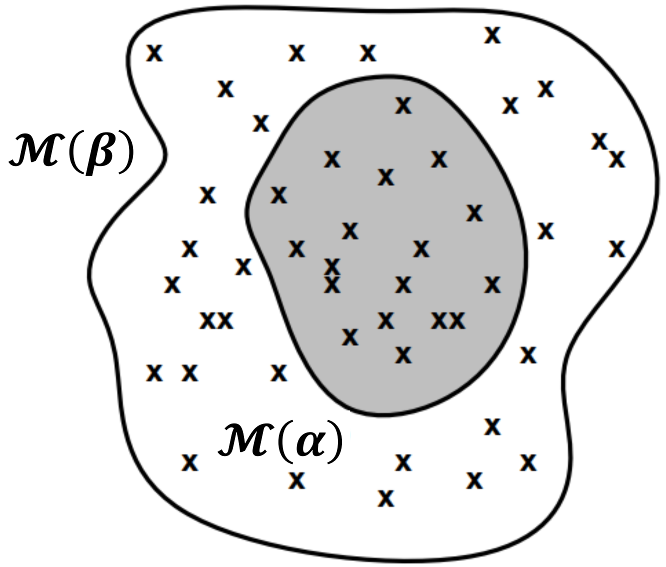
$\alpha = \text{Wet}$

Recap: Propositional Logic

- **Sentence:** propositional symbols, or their negations ($\neg$), or their combinations through $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$.
- **Models:** An assignment of truth values to propositional symbols.
- **Knowledge base:** a set of sentences
- $\mathcal{M}(f)$: the set of models where sentence f is true.

Entailment

- Sentence α **entails** sentence β means that (in high level) sentence β follows logically from sentence α
- Denoted as $\alpha \vDash \beta$
- $\alpha \vDash \beta$ if and only if $\mathcal{M}(\alpha) \subset \mathcal{M}(\beta)$
- **Example:** Rain ∧ Snow ⊨ Snow



Inference Algorithms

- Given KB and α, the algorithm tries to derive sentence α.
- If an algorithm $\mathcal{A}$ is able to derive α from KB, we write KB $\vdash_{\mathcal{A}} \alpha$
  - This is different from KB $\vDash \alpha$,
- Soundness (correctness)
  - The algorithm can only derive α when α is entailed by KB.
  - In other words: If KB $\vdash_{\mathcal{A}} \alpha$, then KB $\vDash \alpha$
- Completeness
  - For any α that KB entails, the algorithm is able to derive α.
  - If other words: If KB $\vDash \alpha$, then if KB $\vdash_{\mathcal{A}} \alpha$

A (Simple) Inference Algorithm: Model Checking

```
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 }))
```

A (Simple) Inference Algorithm: Model Checking

Model Checking (KB, α):

Let $\mathcal{M}$ be the set of all possible models
($|\mathcal{M}| = 2^N$ if there are N propositional symbols in KB $\cup$ $\{\alpha\}$)

For $m \in \mathcal{M}$:

If KB is True in m and α is False in m : **return** False

return True

Theorem Proving

Idea: Instead of checking all models, will just perform manipulations on the sentence level.

Inference Rules

$\alpha \to \beta \equiv \neg \alpha \vee \beta$

$\neg(\alpha_1 \wedge \cdots \wedge \alpha_k)$

$\equiv (\neg \alpha_1) \vee (\quad) \cdots \vee (\neg \alpha_k)$

- Modus Ponens (Latin for *mode the affirms*)

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_k, \quad (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k) \Rightarrow \beta}{\beta}$$

← premises

← conclusion

or $$\frac{\alpha_1, \alpha_2, \ldots, \alpha_k, \quad (\neg\alpha_1 \vee \neg\alpha_2 \vee \cdots \vee \neg\alpha_k \vee \beta)}{\beta}$$

- And Eliminations

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k}{\alpha_i}$$

Standard Logical Equivalence

(can be applied in any steps in the inference algorithm)

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

Inference Rules

Example: KB = {Rain ⇒ Wet, Wet ⇒ Unhappy, Rain}, α = Unhappy.

Applying Modus Ponens on KB
(i.e., try to **match** sentences in KB with premises α and β)

$$\frac{\text{Rain, Rain} \Rightarrow \text{Wet}}{\text{Wet}}$$

Modus Ponens:

$$\frac{\alpha_1, \ldots, \alpha_k, \quad (\alpha_1 \land \cdots \land \alpha_k) \Rightarrow \beta}{\beta}$$

KB = {Rain ⇒ Wet, Wet ⇒ Unhappy, Rain, Wet}
Applying Modus Ponens on KB

$$\frac{\text{Wet}, \quad \text{Wet} \Rightarrow \text{Unhappy}}{\text{Unhappy}}$$

Forward Inference

Input: KB, α, $\mathfrak{I} =$ a set of inference rule

If $\alpha \in$ KB: **return** True

Repeat:

Choose a set of sentences $\alpha_1, \dots, \alpha_k \in$ KB such that

$$\frac{\alpha_1, \alpha_2, \dots, \alpha_k}{\beta}$$

matches a rule in $\mathfrak{I}$, and $\beta \notin$ KB.

If $\beta = \alpha$: **return** True

If such $(\alpha_1, \alpha_2, \dots, \alpha_k, \beta)$ does not exist: **return** False

Add β to KB.

Forward Inference

- Forward inference is a search problem
  - What are the states, actions, successor function, and goal test?
  - Algorithms introduced for search problems can be applied here.

- Is the forward inference algorithm sound?
  - Yes, as long as all inference rules you use are sound

- Is forward inference complete?

Forward Inference

Example:

KB = {Rain ⇒ Wet,   Rain ∨ Shine,   Wet ∨ Shine ⇒ Happy}

α = Happy

Use Forward Inference algorithm with $\mathfrak{I} =$ {Modus Ponens}

- Can KB entail α?
- Can the algorithm derive α from KB?

$$\frac{P, \quad P \Rightarrow Q}{Q}$$

$$\frac{P, \quad \neg P \lor Q}{Q}$$

Forward Inference with Modus Ponens is **sound** but **not complete**

A Sound and Complete Algorithm?

Fact 1. If KB only consists of **Horn clauses,**

then Forward Inference with **Modus Ponens** is sound and complete.

Fact 2. In general, Forward Inference with **Resolution** is sound and complete.

Horn Clauses + Modus Ponens is Complete

Horn clause: sentence that have the following forms

$$X_1 \wedge X_2 \wedge \cdots \wedge X_{k-1} \Rightarrow X_k$$

$$\equiv$$

$$\neg X_1 \vee \neg X_2 \vee \cdots \vee \neg X_{k-1} \vee X_k$$

Disjunction with only one positive symbol
(Definite clause)

or

$$X_1 \wedge X_2 \wedge \cdots \wedge X_k \Rightarrow \text{False}$$

$$\equiv$$

$$\neg X_1 \vee \neg X_2 \vee \cdots \vee \neg X_k$$

Disjunction with no positive symbol
(Goal clause)

Horn Clauses + Modus Ponens is Complete

KB

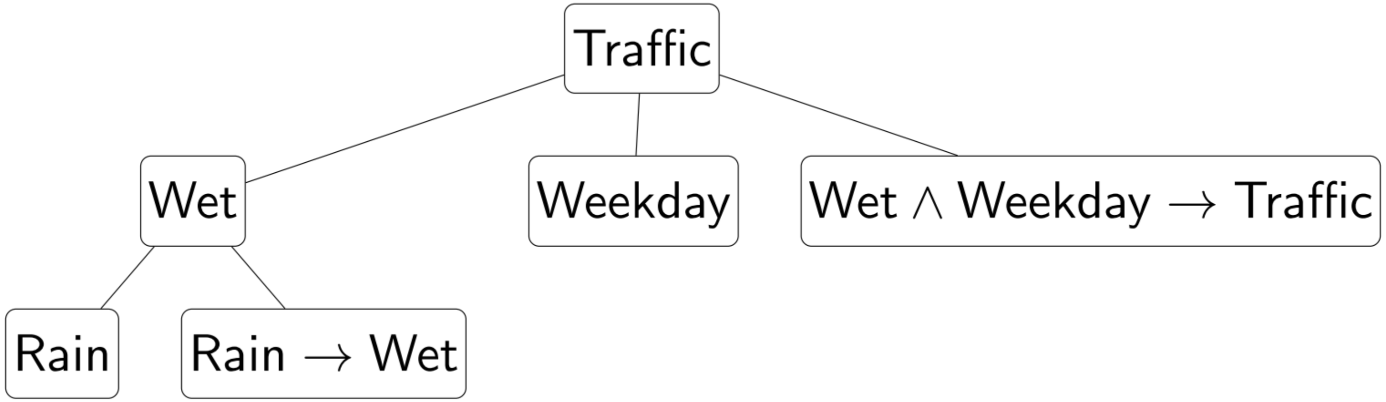
Rain

Weekday

Rain $\rightarrow$ Wet

Wet $\wedge$ Weekday $\rightarrow$ Traffic

Traffic $\wedge$ Careless $\rightarrow$ Accident



Intuition: The inference procedure of horn clauses is *direct*, in the sense that there is no branching.

Horn clause: Rain ∧ Snow → Dark ∧ Traffic

R ∧ S → D

R ∧ S → T

Non- horn clause: Wet → $\underbrace{\text{Rain} \vee \text{Snow}}$

Has to branch into the cases ¬Rain, ¬Snow etc.

A pseudocode for Forward Inference with Modus Ponens (this algorithm is also called **Forward Chaining**). This pseudocode assumes that all sentences are definite clauses (but it's easy to extend it to handle goal clauses as well).

The time complexity is linear in the "**size of KB**", i.e., the sum of the lengths of all sentences in KB.

```
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
```

Figure 7.15 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

General Case: Resolution is Complete

Resolution

$$\frac{\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_k \vee p, \quad \neg p \vee \beta_1 \vee \beta_2 \vee \cdots \vee \beta_m}{\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_k \vee \beta_1 \vee \beta_2 \vee \cdots \vee \beta_m}$$

Example

$$\frac{\text{Rain} \vee \text{Shine}, \quad \neg\text{Rain} \vee \text{Wet}}{\text{Shine} \vee \text{Wet}}$$

Converting Sentences to CNF Before Applying Resolution

Conjunctive Normal Form (CNF)

Example: $(A \lor B \lor \neg C) \land (\neg B \lor D)$

Converting Sentences to CNF: Example

Initial formula:

$$(\mathsf{Summer} \rightarrow \mathsf{Snow}) \rightarrow \mathsf{Bizzare}$$

Remove implication ($\rightarrow$):

$$\neg(\neg\mathsf{Summer} \vee \mathsf{Snow}) \vee \mathsf{Bizzare}$$

Push negation ($\neg$) inwards (de Morgan):

$$(\neg\neg\mathsf{Summer} \wedge \neg\mathsf{Snow}) \vee \mathsf{Bizzare}$$

Remove double negation:

$$(\mathsf{Summer} \wedge \neg\mathsf{Snow}) \vee \mathsf{Bizzare}$$

Distribute $\vee$ over $\wedge$:

$$(\mathsf{Summer} \vee \mathsf{Bizzare}) \wedge (\neg\mathsf{Snow} \vee \mathsf{Bizzare})$$

Converting Sentences to CNF: General Rules

Conversion rules:

- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \wedge (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \vee g}$
- Move $\neg$ inwards: $\frac{\neg(f \wedge g)}{\neg f \vee \neg g}$
- Move $\neg$ inwards: $\frac{\neg(f \vee g)}{\neg f \wedge \neg g}$
- Eliminate double negation: $\frac{\neg\neg f}{f}$
- Distribute $\vee$ over $\wedge$: $\frac{f \vee (g \wedge h)}{(f \vee g) \wedge (f \vee h)}$

Resolution-Based Inference Algorithm

Note that $\text{KB} \vDash \alpha$ is equivalent to $\mathcal{M}(\text{KB} \wedge \neg \alpha)$ = empty set

KB’ ← KB U $\{\neg \alpha\}$

Convert all sentences in KB’ to CNF

Repeatedly apply Resolution Rule until

1) False is derived → return $\text{KB} \vDash \alpha$
2) No new sentence can be derived → return $\text{KB} \nvDash \alpha$

Resolution-Based Inference Algorithm

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 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*
        $clauses \leftarrow clauses \cup new$

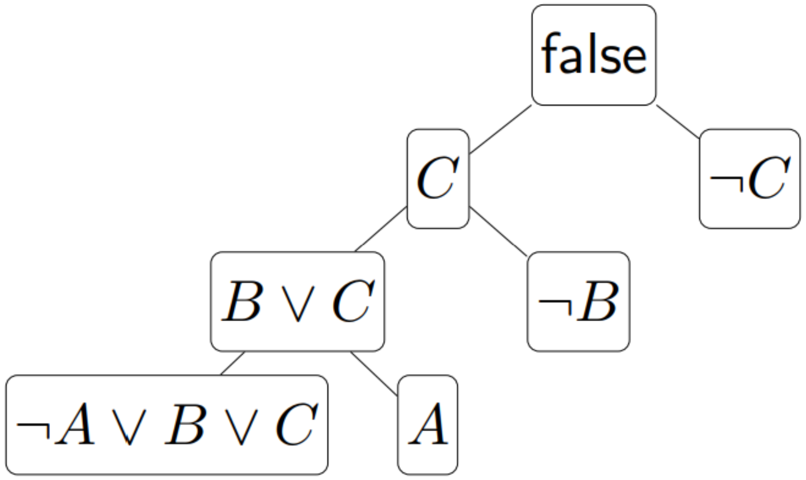
Resolution-Based Inference Algorithm

$$\mathrm{KB}' = \{A \to (B \vee C), A, \neg B, \neg C\}$$

Convert to CNF:

$$\mathrm{KB}' = \{\neg A \vee B \vee C, A, \neg B, \neg C\}$$

Repeatedly apply **resolution** rule:



Conclusion: KB **entails** f

Time Complexity

- Modus Ponens

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_k, \quad (\alpha_1 \land \alpha_2 \land \cdots \land \alpha_k) \Rightarrow \beta}{\beta}$$

Each rule application adds sentence with **one** propositional symbol
→ **linear time**

- Resolution

$$\frac{\alpha_1 \lor \alpha_2 \lor \cdots \lor \alpha_k \lor p, \quad \neg p \lor \beta_1 \lor \beta_2 \lor \cdots \lor \beta_m}{\alpha_1 \lor \alpha_2 \lor \cdots \lor \alpha_k \lor \beta_1 \lor \beta_2 \lor \cdots \lor \beta_m}$$

Each rule application adds sentence with **many** propositional symbol
→ **exponential time**

Recap

	Modus Ponens	Resolution
Sound?	Yes	Yes
Complete?	No	Yes
Complete for horn clauses?	Yes	Yes
Time complexity	linear	exponential

Homework 3

Choice problems deadline: **11:59PM, Oct. 9 (No late submission!)**

Programming problems deadline: 11:59PM, Oct. 23

Question 1: Logic Warmup

- Practice working with the class **Expr** which will be used to represent propositional sentences in the later questions.
- Example. Create an Expr instance that represents the conjunction of the following four expressions.

$$C \leftrightarrow (B \vee D)$$

$$A \rightarrow (\neg B \wedge \neg D)$$

$$\neg (B \wedge \neg C) \rightarrow A$$

$$\neg D \rightarrow C$$

Question 2: Logic Workout

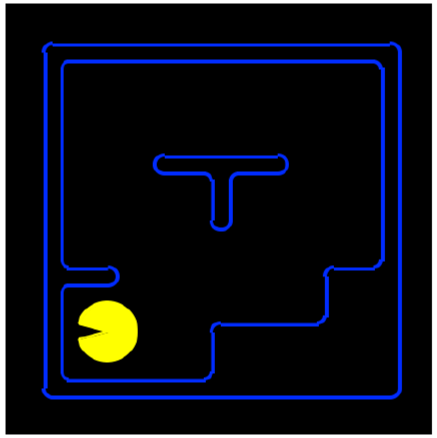
- Express sentences using Conjunctive Normal Form (CNF)
- Example. Express the function **ExactOne**([A, B, C, …]) as a CNF

Question 3: PAC Physics and Satisfiability

- Express the physics of PACMAN using propositional logics. The rules include
  - PAC must be at exactly one position that is not a wall
  - PAC takes exactly one of the four actions in every round
  - If PAC is at (x,y) and takes WEST actions at time t-1 and there is no wall at (x-1, y), then PAC is at (x-1,y) at time t
  - …

axioms

Question 4: Path Planning with Logic

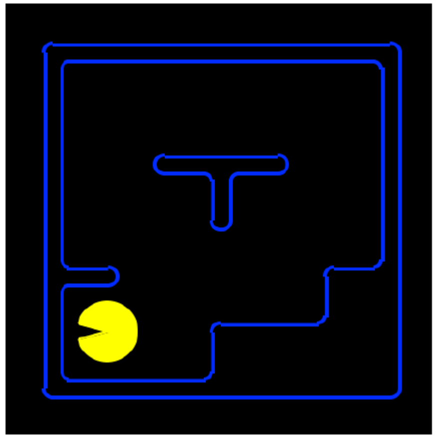


For t = 1, 2, …

- Check if there exists there is a feasible assignment for $(x_1, y_1, a_1, x_2, y_2, a_2, \ldots, x_t, y_t)$ where (x_t, y_t) is goal.
- If so, return the path
- KB ← KB ∪ {a_t is one of NSEW}
- KB ← KB ∪ {if a_t is N and (x_t, y_t+1) is not wall, then $(x_{t+1}, y_{t+1})=(x_t, y_t+1)$, … }

(All symbols represent “binary” variables, so the pseudocode above only provides rough ideas but not exactly how you implement it)

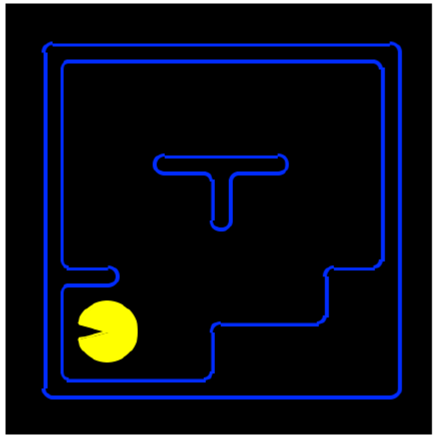
Question 5: Eating All Food



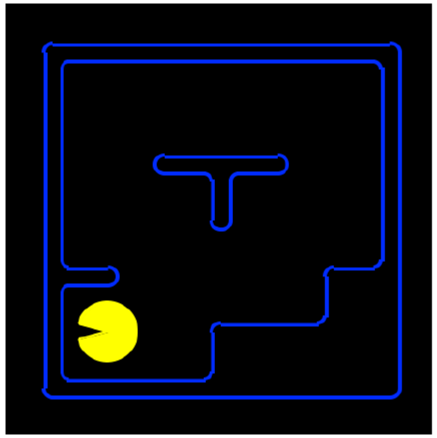
- Similar to previous question, but now with additional symbols representing whether there is food in a location
- Sentences related to the PACMAN position (x,y) and whether there is food at (x,y) should be added to KB.

Question 6: Localization

- PACMAN does not know its own position
- But it has a sensor that tells whether there is wall on NSEW
- Given a sequence of actions and sensor signals, can you tell the position of the PACMAN?



Question 7: Mapping



- PACMAN knows its initial position
- It has a sensor that tells whether there is wall on NSEW
- Given a sequence of actions and sensor signals, reconstruct the positions of the walls

Question 8: Simultaneous Localization and Mapping

- PACMAN knows its initial position
- The sensor only tells **how many walls** (0 to 3) are around it
- Given a sequence of actions and sensor signals, reconstruct the positions of the walls and the position of the PACMAN

Midterm Exam

Types of Questions

- Multiple choice questions (like in the homework)
- Questions where you may need to provide steps. For example,
  - Which nodes are pruned under alpha-beta pruning in a specific game tree
  - Argue why a particular heuristic function is consistent
- Resources from other universities:
  - https://inst.eecs.berkeley.edu/~cs188/sp24/resources/
  - https://inst.eecs.berkeley.edu/~cs188/su24/resources/
  - https://stanford.edu/~cpiech/cs221/handouts/practiceMidterms.html

Midterm Review

- Next Thursday (Oct 10)

First-Order Logic

Limitations of Propositional Logic

Alice and Bob both know logic.

AliceKnowsLogic ∧ BobKnowsLogic

Every student knows logic.

AliceIsStudent → AliceKnowsLogic
BobIsStudent → BobKnowsLogic

⋮

Every student knows some algorithm.

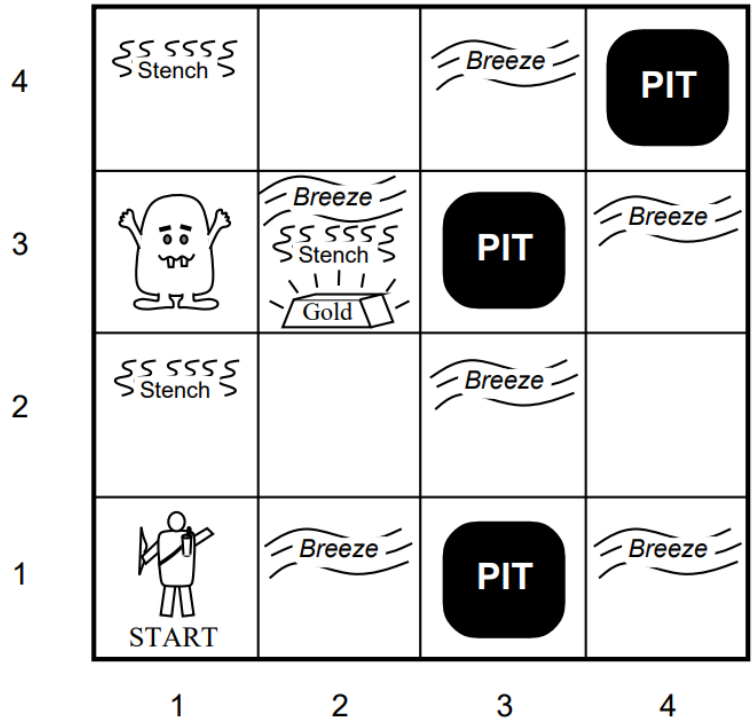
AliceIsStudent → AliceKnowsBFS ∨ AliceKnowsDFS ∨ …
BobIsStudent → BobKnowsBFS ∨ BobKnowsDFS ∨ …

⋮

First-order logic: $\forall x\ \text{IsStudent}(x) \rightarrow (\exists y\ \text{IsAlgorithm}(y) \land \text{Knows}(x,y))$

Limitations of Propositional Logic

If you're adjacent to a pit, then you can feel breeze



Propositional logic:

$P_{1,1} \Rightarrow B_{1,2} \wedge B_{2,1}$

$P_{1,2} \Rightarrow B_{1,1} \wedge B_{2,2} \wedge B_{1,3}$

$\vdots$

First-order logic:

$\forall w,x,y,z \quad Adjacent(Grid(w,x), Grid(y,z)) \wedge Pit(Grid(w,x)) \Rightarrow Breeze(Grid(y,z))$

Limitations of Propositional Logic

Propositional logic is sometimes clunky. What is missing?

- **Objects** and **predicates**: propositions (e.g., AliceKnowsLogic) have more internal structure (Alice, Knows, Logic)
- **Quantifiers** and **variables**: *all* is a quantifier which applies to each person, don't want to enumerate them all.

First-Order Logic

Alice and Bob both know logic.

$\text{Knows}(\text{Alice}, \text{Logic}) \land \text{Knows}(\text{Bob}, \text{Logic})$

Every student knows logic.

$\forall x, \ \text{Student}(x) \Rightarrow \text{Knows}(x, \text{Logic})$

Syntax and Semantics of First-Order Logic

- **Terms** (refer to object)
  - Constant: Alice, Logic
  - Variable: x
  - Function of terms: Father(.)

Knows(Alice, Logic) $\wedge$ Knows(Bob, Logic)

$\forall$x, Student(x) $\Rightarrow$ Knows(x, Logic)

$\forall$x, HasBloodType(x, 'AB') $\Rightarrow$ $\neg$HasBloodType(Father(x), 'O') $\wedge$ $\neg$HasBloodType(Mother(x), 'O')

- **Predicate:** Knows, Student, HasBloodType
- **Atomic sentence:** Predicate(Terms, …)
- **Complex sentence:**

  $\neg$ sentence
  sentence $\wedge$ sentence
  sentence $\vee$ sentence
  sentence $\Rightarrow$ sentence
  sentence $\Leftrightarrow$ sentence
  Quantifier variable, sentence

Syntax and Semantics First-Order Logic

- **Quantifiers**
  - Universal quantifier $\forall$ (think conjunction): $\forall x\ P(x)$ is like $P(A) \wedge P(B) \wedge \ldots$
  - Existential quantifier $\exists$ (think disjunction): $\exists x\ P(x)$ is like $P(A) \vee P(B) \vee \ldots$
- Properties of quantifiers
  - $\neg\forall x\ P(x)$ is equivalent to $\exists x\ \neg P(x)$
  - $\forall x \exists y\ Know(x,y)$ and $\exists y \forall x\ Know(x,y)$ are different

Translation from Natural Language to FOL

Every student knows logic.

$$\forall x \;\; \text{Student}(x) \Rightarrow \text{Knows}(x, \text{Logic})$$

Some student knows logic.

$$\exists x \;\; \text{Student}(x) \wedge \text{Knows}(x, \text{Logic})$$

What about $\exists x \;\; \text{Student}(x) \Rightarrow \text{Knows}(x, \text{Logic})$?

Translation from Natural Language to FOL

There is some course that every student has taken.

$$\exists y\, \mathsf{Course}(y) \wedge [\forall x\, \mathsf{Student}(x) \rightarrow \mathsf{Takes}(x, y)]$$

Every even integer greater than 2 is the sum of two primes.

$$\forall x\, \mathsf{EvenInt}(x) \wedge \mathsf{Greater}(x, 2) \rightarrow \exists y\, \exists z\, \mathsf{Equals}(x, \mathsf{Sum}(y, z)) \wedge \mathsf{Prime}(y) \wedge \mathsf{Prime}(z)$$

If a student takes a course and the course covers a concept, then the student knows that concept.

$$\forall x\, \forall y\, \forall z\, (\mathsf{Student}(x) \wedge \mathsf{Takes}(x, y) \wedge \mathsf{Course}(y) \wedge \mathsf{Covers}(y, z)) \rightarrow \mathsf{Knows}(x, z)$$

Inference in First-Order Logic

- Convert everything to propositional logic

- Modus ponens
  - Sound
  - Complete for Horn clauses

- Resolution
  - Sound and complete

Recall: **Horn clause in PL**

$$\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k \Rightarrow \beta$$
$$\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k \Rightarrow \text{False}$$

Each α_i is a propositional symbol

Horn clause in FOL

$$\forall x_1, \ldots, \forall x_n \quad \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k \Rightarrow \beta$$
$$\forall x_1, \ldots, \forall x_n \quad \alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k \Rightarrow \text{False}$$

Each α_i is an atomic sentence
(which may involve universal quantifier)

Forward Inference with Modus Ponens

$$\frac{\alpha_1', \alpha_2', \dots, \alpha_k', \quad \forall x_1, \dots, \forall x_n, (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k) \Rightarrow \beta}{\beta'}$$

where $\alpha_1', \alpha_2', \dots, \alpha_k', \beta', \alpha_1, \dots, \alpha_k, \beta$ are atomic sentences, and

$(\alpha_1, \alpha_2, \dots, \alpha_k, \beta)$ and $(\alpha_1', \alpha_2', \dots, \alpha_k', \beta')$ can be **unified** through a **substitution** from variable to terms.

make them look the same

Forward Inference with Modus Ponens

Take(Alice, CS4710)
Covers(CS4710, Logic)
∀x,y,z Take(x,y) ∧ Covers(y,z) ⇒ Knows(x,z)

Knowledge Base

$$\frac{\alpha_1', \alpha_2', \ldots, \alpha_k', \quad \forall x_1, \ldots, \forall x_n, (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k) \Rightarrow \beta}{\beta'}$$

$$\frac{\text{Take(Alice, CS4710)}, \quad \text{Covers(CS4710, Logic)}, \quad \forall x,y,z\ \text{Take}(x,y) \wedge \text{Covers}(y,z) \Rightarrow \text{Knows}(x,z)}{?}$$

Substitution:
x / Alice
y / CS4710
z / Logic

? = Knows(x,z) applying the substitution
= Knows(Alice,Logic)

Forward Inference with Modus Ponens

∀w Take(Alice, w)
Covers(CS4710, Logic)
∀x,y,z Take(x,y) ∧ Covers(y,z) ⇒ Knows(x,z)

Knowledge Base

$$\frac{\alpha_1', \alpha_2', \ldots, \alpha_k', \quad \forall x_1, \ldots, \forall x_n, (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k) \Rightarrow \beta}{\beta'}$$

$$\frac{\forall w\ Take(Alice, w), \quad Covers(CS4710, Logic), \quad \forall x,y,z\ Take(x,y) \wedge Covers(y,z) \Rightarrow Knows(x,z)}{?}$$

Substitution:
x / Alice
y / CS4710
z / Logic
w / CS4710

? = Knows(x,z) applying the substitution
= Knows(Alice,Logic)

Forward Inference with Modus Ponens

Take(Alice, CS4710)
∀v Covers(CS4710, v)
∀x,y,z Take(x,y) ∧ Covers(y,z) ⇒ Knows(x,z)

Knowledge Base

$$\frac{\alpha_1', \alpha_2', \dots, \alpha_k', \quad \forall x_1, \dots, \forall x_n, (\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_k) \Rightarrow \beta}{\beta'}$$

$$\frac{\text{Take(Alice, CS4710), } \forall v \text{ Covers(CS4710, v),} \quad \forall x,y,z \text{ Take(x,y)} \wedge \text{Covers(y,z)} \Rightarrow \text{Knows(x,z)}}{?}$$

Substitution:

x / Alice
y / CS4710
z / v

Substitute variable with another variable

? = Knows(x,z) applying the substitution

= ∀v Knows(Alice,v)

Forward Inference with Modus Ponens

Input: KB, α

If $\alpha \in$ KB: **return** True

Repeat:

Choose a set of atomic sentences $\alpha'_1, \ldots, \alpha'_k$ and rule $\forall x_1, \ldots, \forall x_n, (\alpha_1 \land \alpha_2 \land \cdots \land \alpha_k) \Rightarrow \beta$ in KB such that

$$\alpha'_1, \alpha'_2, \ldots, \alpha'_k, \beta' \text{ matches } \alpha_1, \alpha_2, \ldots, \alpha_k, \beta$$

under variable substitution, and β' cannot be subsumed by any sentence in KB.

If $\beta' = \alpha$: **return** True

If such matching does not exist: **return** False

Add β' to KB.

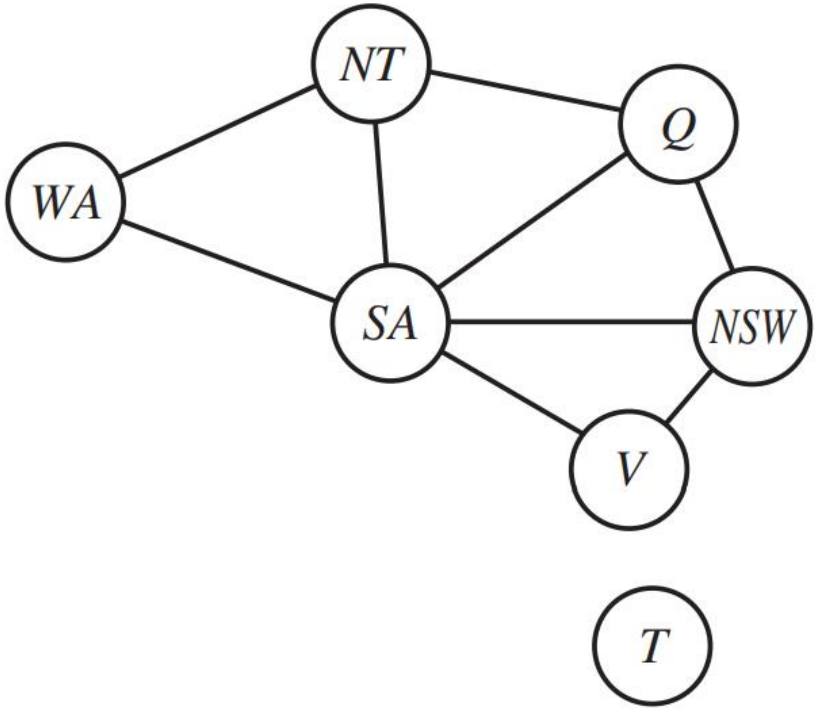
Forward Inference with Modus Ponens

Take(Alice, CS4710)
Take(Alice, CS1234)
Take(Alice, MU4321)
Take(Bob, CS4710)
Covers(DSA, Search)
Covers(LinearAlgebra, matrix)
$\forall x,y,z \quad Take(x,y) \wedge Covers(y,z) \Rightarrow Knows(x,z)$

How to find a matching between x,y,z and terms?

We can view of this problem as finding x,y,z that satisfies **constraints** Take(x,y) and Covers(y,z)
→ Constraint Satisfaction Problem (**CSP**)
→ The heuristics we discussed before can be used

CSP is a Single Horn Clause



(a)

$$
\begin{aligned}
&Diff(wa, nt) \land Diff(wa, sa) \land \\
&\quad Diff(nt, q) \land Diff(nt, sa) \land \\
&\quad Diff(q, nsw) \land Diff(q, sa) \land \\
&\quad Diff(nsw, v) \land Diff(nsw, sa) \land \\
&\quad Diff(v, sa) \;\Rightarrow\; Colorable()
\end{aligned}
$$

$$
\begin{array}{ll}
Diff(Red, Blue) & Diff(Red, Green) \\
Diff(Green, Red) & Diff(Green, Blue) \\
Diff(Blue, Red) & Diff(Blue, Green)
\end{array}
$$

(b)

Figure 9.5 (a) Constraint graph for coloring the map of Australia. (b) The map-coloring CSP expressed as a single definite clause. Each map region is represented as a variable whose value can be one of the constants *Red*, *Green* or *Blue*.

Inference with Resolution

- High-level Ideas
  - Convert everything to CNF
  - Repeatedly apply the resolution rule from KB $\cup \{\neg\alpha\}$

Conversion to CNF

Anyone who likes all animals is liked by someone.

Input:

$$\forall x\,(\forall y\,\mathsf{Animal}(y) \rightarrow \mathsf{Loves}(x, y)) \rightarrow \exists y\,\mathsf{Loves}(y, x)$$

Output:

$$(\mathsf{Animal}(Y(x)) \vee \mathsf{Loves}(Z(x), x)) \wedge (\neg\mathsf{Loves}(x, Y(x)) \vee \mathsf{Loves}(Z(x), x))$$

New to first-order logic:

- All variables (e.g., x) have universal quantifiers by default
- Introduce **Skolem functions** (e.g., $Y(x)$) to represent existential quantified variables

Conversion to CNF (1/2)

Input:

$\forall x\, (\forall y\, \mathsf{Animal}(y) \rightarrow \mathsf{Loves}(x, y)) \rightarrow \exists y\, \mathsf{Loves}(y, x)$

Eliminate implications (old):

$\forall x\, \neg(\forall y\, \neg\mathsf{Animal}(y) \vee \mathsf{Loves}(x, y)) \vee \exists y\, \mathsf{Loves}(y, x)$

Push $\neg$ inwards, eliminate double negation (old):

$\forall x\, (\exists y\, \mathsf{Animal}(y) \wedge \neg\mathsf{Loves}(x, y)) \vee \exists y\, \mathsf{Loves}(y, x)$

Standardize variables (**new**):

$\forall x\, (\exists y\, \mathsf{Animal}(y) \wedge \neg\mathsf{Loves}(x, y)) \vee \exists z\, \mathsf{Loves}(z, x)$

Conversion to CNF (2/2)

$\forall x\, (\exists y\, \mathsf{Animal}(y) \land \neg \mathsf{Loves}(x, y)) \lor \exists z\, \mathsf{Loves}(z, x)$

Replace existentially quantified variables with Skolem functions (**new**):

$\forall x\, [\mathsf{Animal}(Y(x)) \land \neg \mathsf{Loves}(x, Y(x))] \lor \mathsf{Loves}(\underline{Z(x)}, x)$

Distribute $\lor$ over $\land$ (old):

$\forall x\, [\mathsf{Animal}(Y(x)) \lor \mathsf{Loves}(Z(x), x)] \land [\neg \mathsf{Loves}(x, Y(x)) \lor \mathsf{Loves}(Z(x), x)]$

Remove universal quantifiers (**new**):

$[\mathsf{Animal}(Y(x)) \lor \mathsf{Loves}(Z(x), x)] \land [\neg \mathsf{Loves}(x, Y(x)) \lor \mathsf{Loves}(Z(x), x)]$

Resolution

Definition: resolution rule (first-order logic)

$$\frac{f_1 \vee \cdots \vee f_n \vee p, \qquad \neg q \vee g_1 \vee \cdots \vee g_m}{\mathsf{Subst}[\theta, f_1 \vee \cdots \vee f_n \vee g_1 \vee \cdots \vee g_m]}$$

where $\theta = \mathsf{Unify}[p, q]$.

Example: resolution

$$\frac{\mathsf{Animal}(Y(x)) \vee \mathsf{Loves}(Z(x), x), \qquad \neg\mathsf{Loves}(u, v) \vee \mathsf{Feeds}(u, v)}{\mathsf{Animal}(Y(x)) \vee \mathsf{Feeds}(Z(x), x)}$$

Substitution: $\theta = \{u/Z(x), v/x\}$.

Recap: FOL and PL

- First-order logic provides internal structures for propositions

Knows(Alice, Logic)
Knows(Bob, DFS)
Takes(Boyfriend(Alice), CS4710)

Predicate Function Constant

AliceKnowsLogic
BobKnowsDFS
AliceBoyfriendTakesCS4710

Propositional symbol

- First-order logic uses quantifiers ∀, ∃ to generalize an idea across different objects

Knowledge Representation using (first-order or other) logic

- Chapter 12 in https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf
  - Give some ideas how to create knowledge representations on general concepts such as events, time, physical objects etc, using first-order logic.
- Knowledge base and inference algorithms are important elements of **expert systems**
- DENDRAL (1968): Predict molecular structure based on spectrographic data
- MYCIN (1975): Diagnose blood infections
- XCON (1978): Select computer system components based on customer’s need
- Many companies built expert systems and software/hardware specialized for their purpose.