Artificial Intelligence

Augustin Cosse.

Fall 2020

October 16, 2020

So far

- Simple Reflex, Random agents, Utility based, Goal based
- Improvement through Search Methods (uninformed (DFS, BFS), informed (BS, A*)).
- Logical Reasoning, Propositional logic (including syntax and semantics)
- Propositional inference (Part I)

Validity and Soundness

- An inference can be valid without being sound

All animals live on the moon.
All humans are animals.
Therefore, all humans live on the Moon.

- Another desirable property is Completeness.
- An inference algorithm is complete if it can derive any sentence that is entailed

Completeness

- Example of incomplete language : Zermelo - Fraenkel set theory with the axiom of choice (ZFC)
- The continuum hypothesis

  There is no set whose cardinality is strictly between that of the integers and the real numbers.

  cannot be proved or disproved within ZFC
- Example of incomplete theorem prover : Prolog
- Incompleteness of Prolog arises because of its unbounded depth-first search strategy

Resolution based Theorem Prover, CNF and completeness

- So far we have discussed soundness and validity of inference rules but we have not discussed completeness
- We will now introduce a simple inference rule which always yields a complete inference algorithm when coupled with a complete search method
- The idea is known as resolution

Resolution based Theorem Prover, CNF and completeness

- Consider the following example from the WUMPUS world. We denote the fact that there is a Breeze in (m, n) by $B_{m,n}$

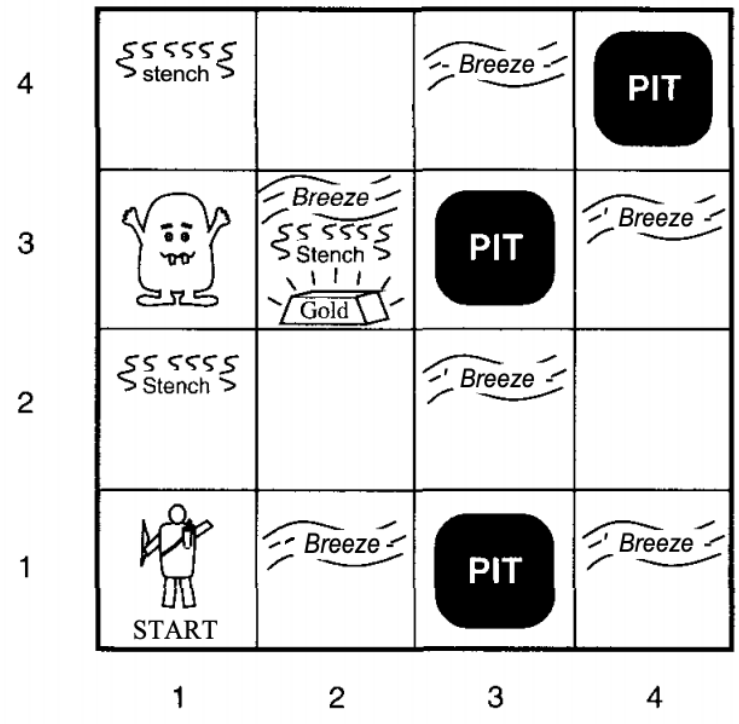


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- We start with a simple version of the resolution rule. When the agent comes back from (2,1) to (1,1) and then moves to (1,2), it gets the percept $R_{11} : \neg B_{1,2}$ (added to the KB)

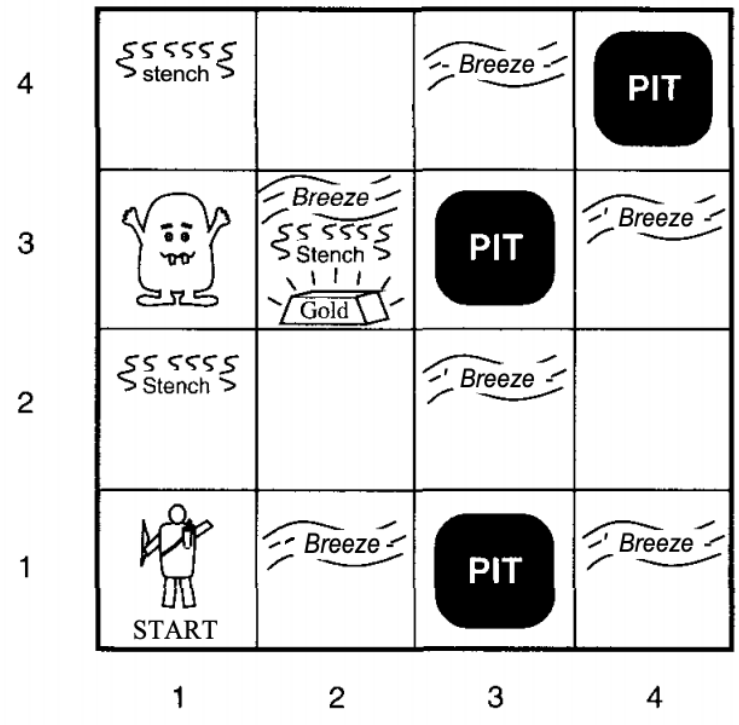


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- We also have the equivalence $R_{12} : B_{1,2} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{1,3})$

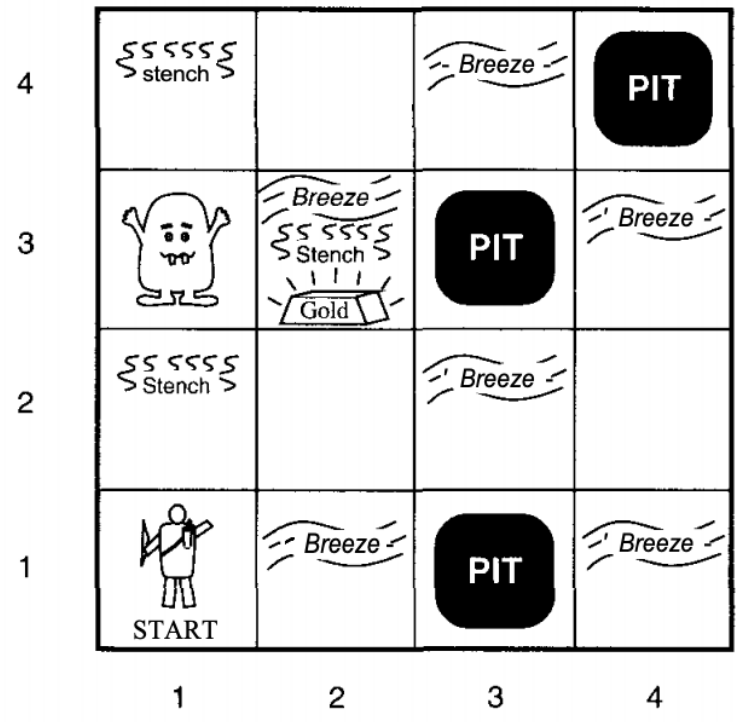


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- From the KB, we can also derive the absence of a pit on (2,2) and (1,3). Hence we have $R_{13} : \neg P_{2,2}$ and $\neg P_{1,3}$

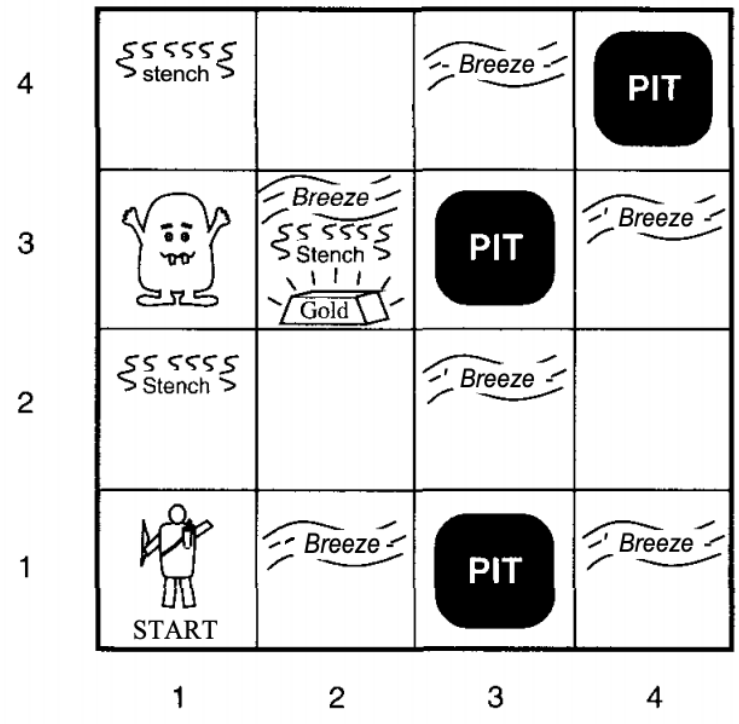


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- Let R_3 denote the proposition $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$ which we had in the KB and $R_5 : B_{2,1}$.

- By using Modus Ponens ($\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$) and Biconditional elimination $\alpha \Leftrightarrow \beta \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$, we can get $B_{2,1} \Rightarrow P_{1,1} \vee P_{2,2} \vee P_{3,1}$ and hence we can add $R_{1,5} : P_{1,1} \vee P_{2,2} \vee P_{3,1}$ (i.e True) to the knowledge base

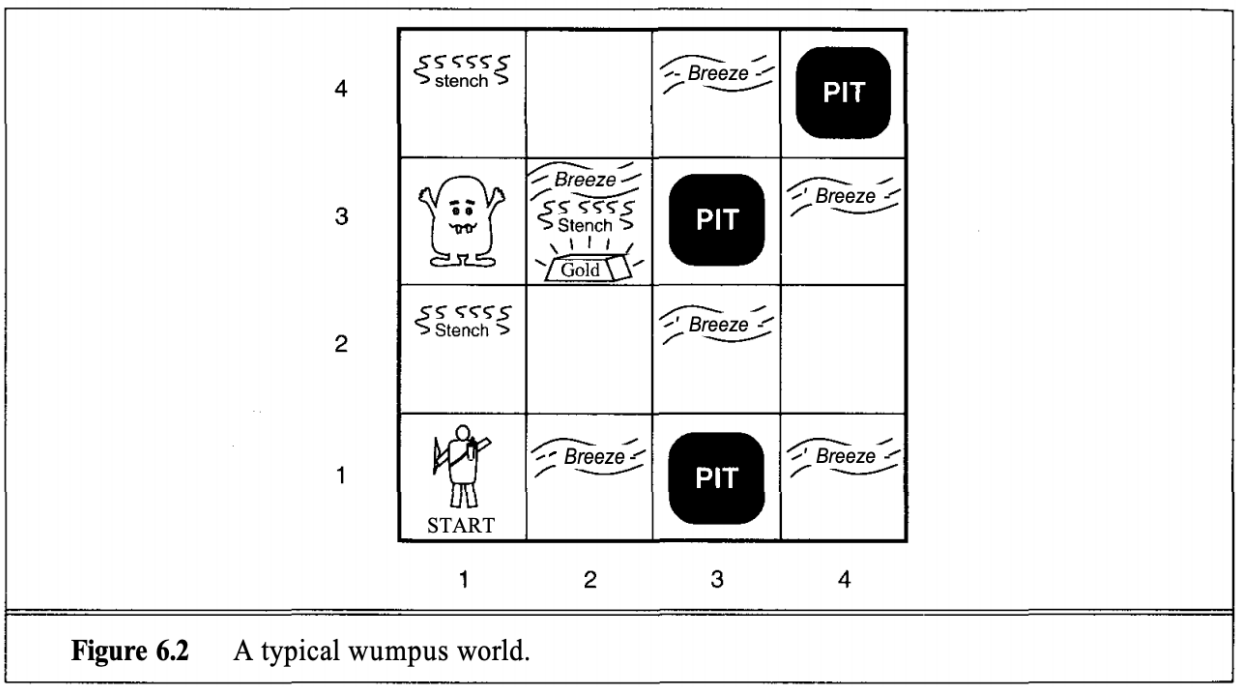


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- Now comes the application of our resolution rule: the proposition (single symbol = literal) $\neg P_{2,2}$ stored in $R_{1,3}$ resolves with the litteral $P_{2,2}$ which appears in $R_{1,5} : P_{1,1} \vee P_{2,2} \vee P_{3,1}$ to give the resolvent $R_{1,6} : P_{1,1} \vee P_{3,1}$

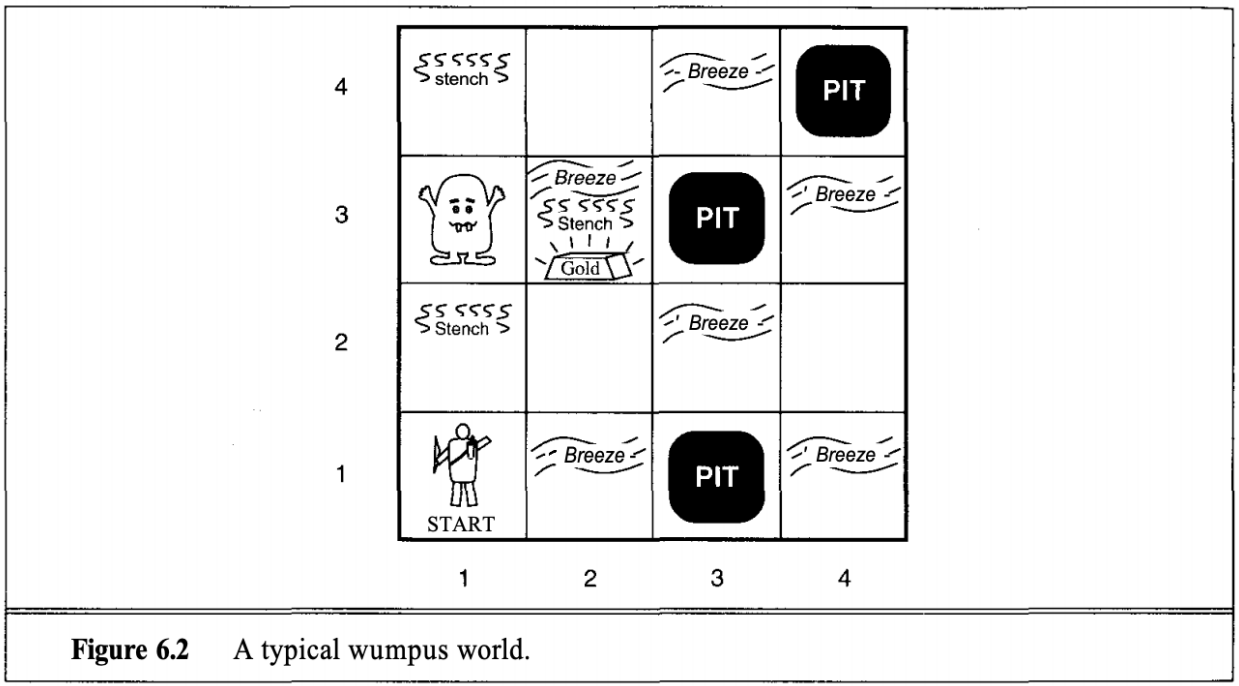


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- In English, the reasoning we make is that if there is a pit in one of $[1,1], [2,2]$ or $[3,1]$ and the pit is not in $[2,2]$ then it must be in either $[1,1]$ or $[3,1]$.

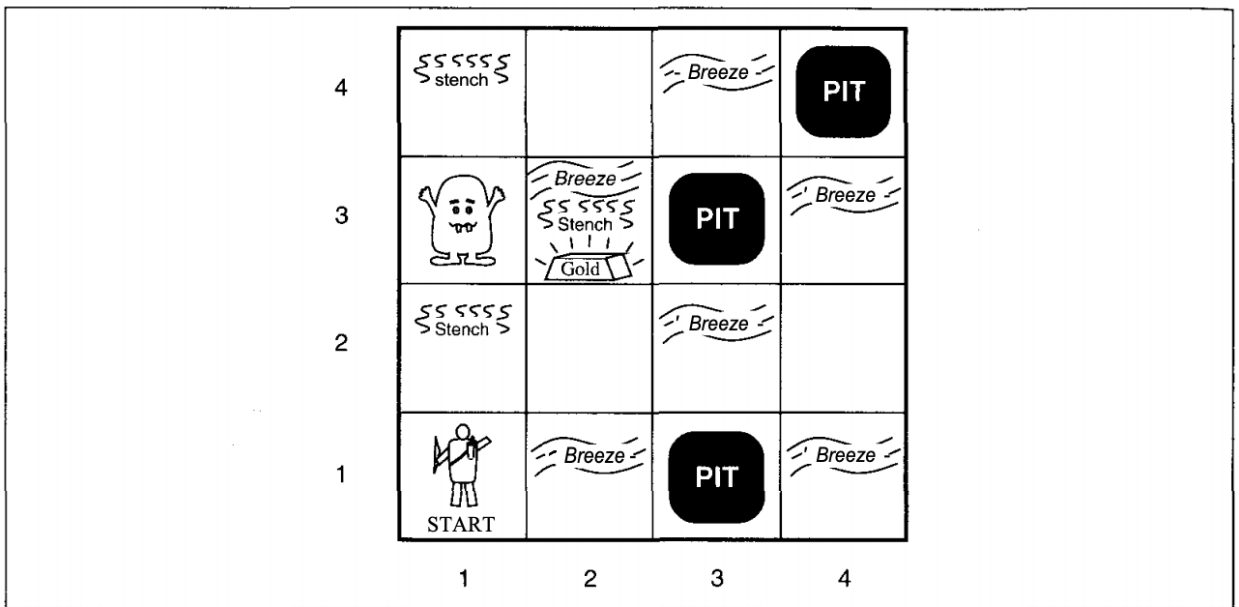


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- Another example is the combination of $\neg P_{1,1}$ and $P_{1,1} \vee P_{3,1}$ to give $P_{3,1}$
- In English, if there is a pit in $[1,1]$, or $[3,1]$, and it is not in $[1,1]$ then it is in $[3,1]$

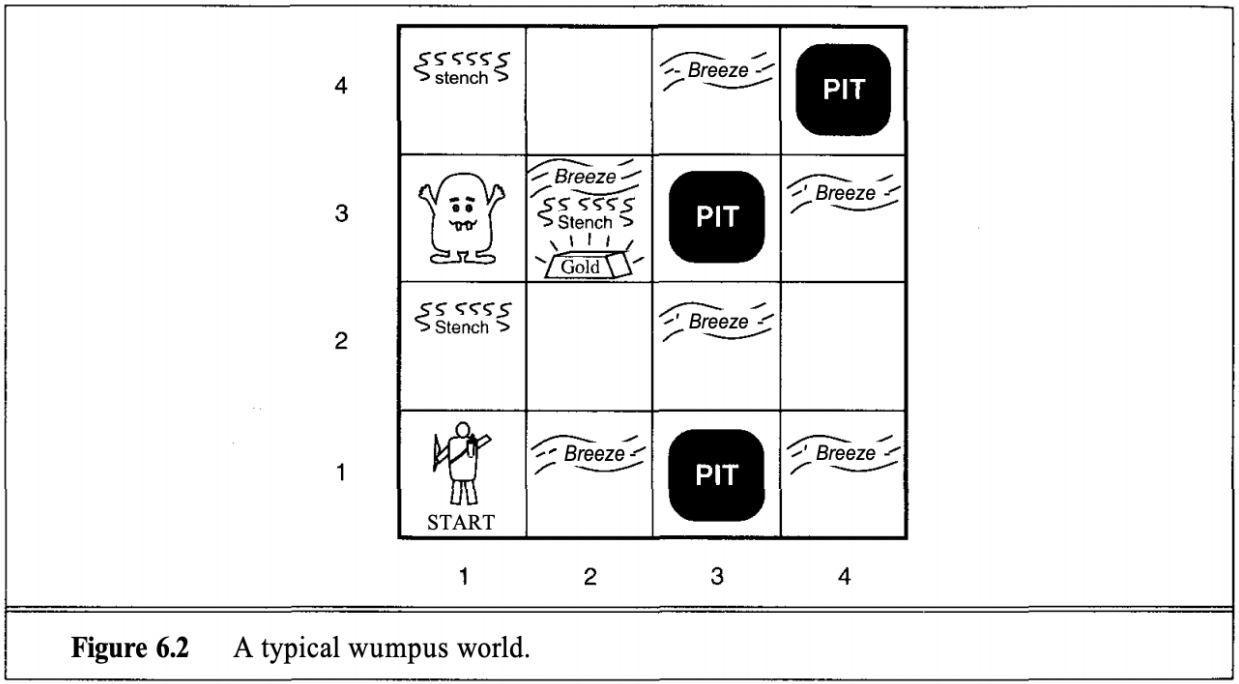


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- The steps that we covered are examples of the so-called unit resolution inference rule. If ℓ_i and m are complementary literals,

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \ldots \vee \ell_k}$$

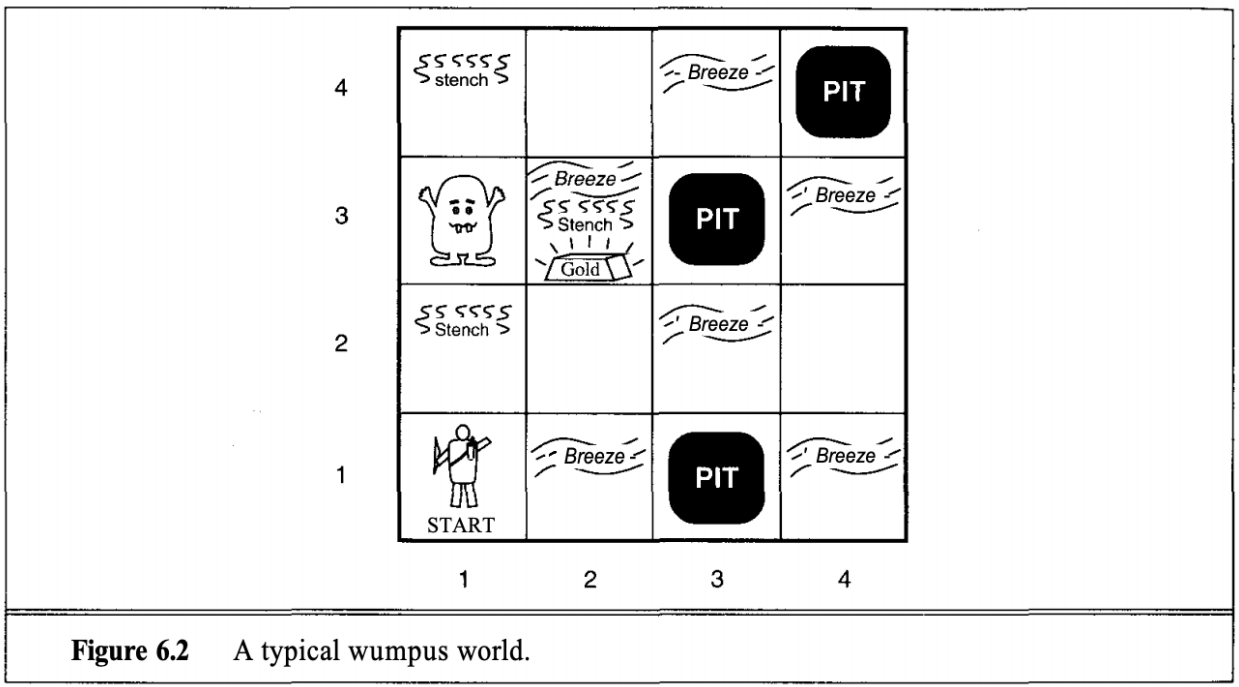


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- The general resolution rule reads as (Here ℓ_i and m_j are complementary litterals)

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m_1 \vee \ldots \vee m_n}{\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

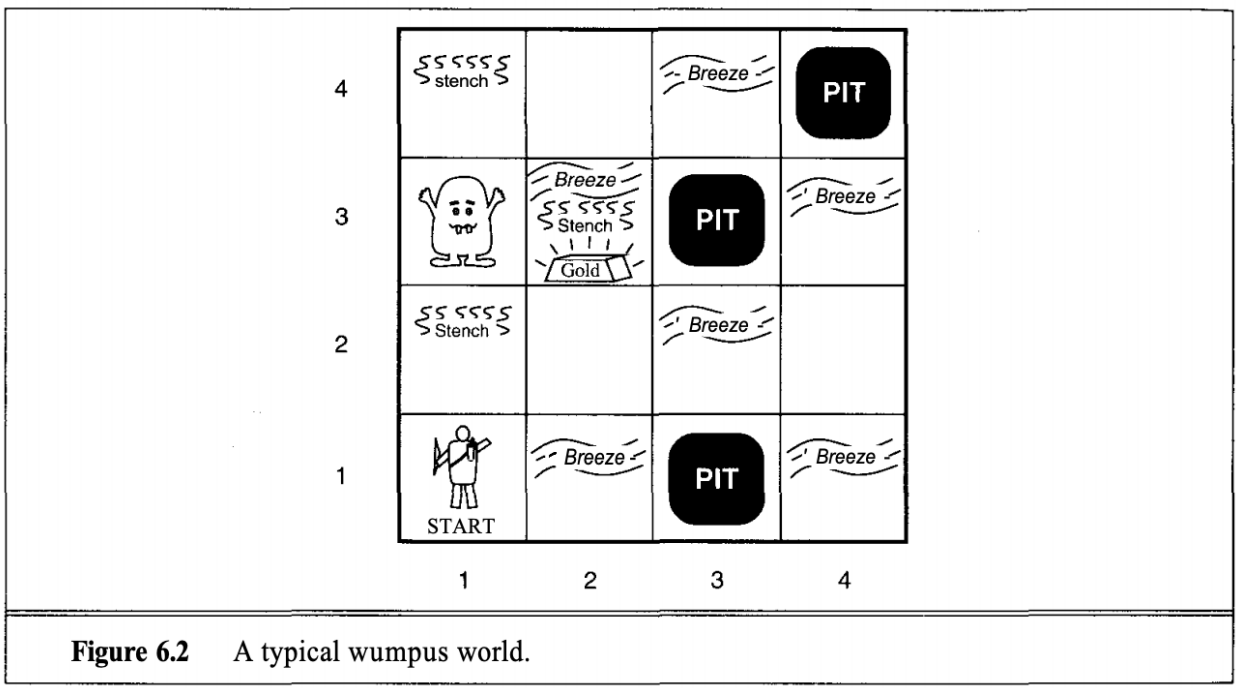


Figure 6.2 A typical wumpus world.

Resolution based Theorem Prover, CNF and completeness

- As an example, we have

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

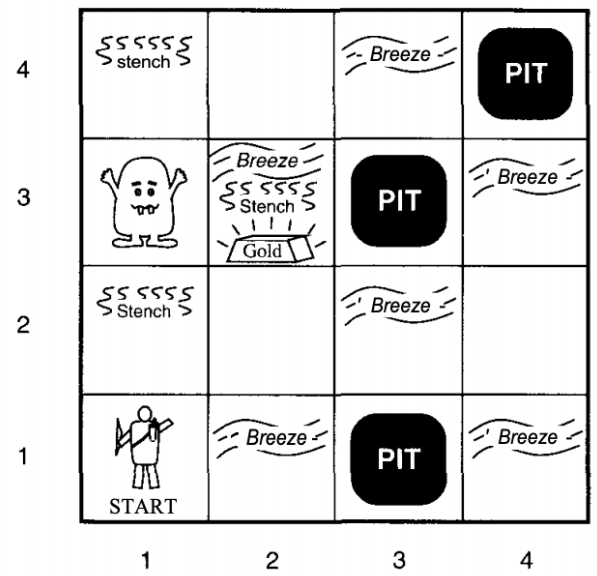


Figure 6.2 A typical wumpus world.

Resolution, technical aspects

- The resulting clause should contain only one copy of each literal. As an example, resolving $(A \vee B)$ with $A \vee \neg B$ gives $A \vee A$ which must be reduced to A.
- Removing multiple copies of literals is called factoring.
- The soundness of the resolution rule can be easily seen. if ℓ_i is true then m_j is false and $m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots m_n$ must be true since $m_1 \vee \ldots \vee m_n$ is among the premises.
- Similarly, if ℓ_i is false, then $\ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k$ is true since $\ell_1 \vee \ldots \vee \ell_k$ is among the premises. Since ℓ_i or m_j is necessarily true, the resulting clause is necessarily true.

Resolution and Conjunctive rules

- The interesting aspect of Resolution rules is that they can be used to generate complete inference procedures
- A resolution based theorem prover can, for any sentence α and β from propositional logic decide whether $\alpha \vDash \beta$.

Resolution and Conjunctive rules

- The resolution rule only applies to clauses (that is disjunction of literals) and so it would seem to be relevant only to knowledge bases and queries consisting of clauses.
- One could then wonder how it can lead to a complete inference procedure for all of propositional logic
- In fact every sentence of Propositional logic is logically equivalent to a conjunction of clauses.
- A sentence expressed as a conjunction of clauses is said to be in conjunctive normal form

Resolution and Conjunctive rules

- To convert a proposition to CNF, we rely on the following 4 steps
  - Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$
  - Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$
  - CNF requires the negation to only appears within each literal, so we move $\neg$ inwards by repeating the equivalences

$$\neg(\neg\alpha) \equiv \alpha \quad \text{double negation elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta), \quad \text{De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta), \quad \text{De Morgan}$$

  - Once we have a sentence containing only conjunctions and disjunctions, we apply the distributivity law, distributing $\vee$ over $\wedge$ wherever possible.

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

Resolution and Conjunctive rules

- Consider the proving of the proposition $B_{11} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ from the Wumpus World
- Turning this proposition to CNF can be done through the following steps
  - ($\Leftrightarrow$ elimination):
    $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
  - ($\Rightarrow$ elimination): $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,2})$
  - (moving $\neg$ inwards):
    $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
  - ($\vee$ distribution over $\wedge$):
    $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\not{P}_{2,1} \vee B_{1,1})$

Resolution algorithm

- Inference procedures based on resolution work by using the principle of contradiction. I.e. to show $KB \models \alpha$, we show $(KB \wedge \neg\alpha)$ is unsatisfiable.
- The algorithm starts by converting the proposition $KB \wedge \neg\alpha$ into a CNF
- Then the resolution rule is applied to the resulting clauses
- Each pair of clauses that contains complementary literals is resolved to produce a new clause which is added to the set of clauses
- The process continues until either
  - There are no new clauses that can be added, in which case KB does not entail α,
  - or two clauses resolve to yield the empty clause, in which case KB entails α

Resolution algorithm

input : knowledge base KB, propositional sentence α, the query $KN \models \alpha$

clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$

new $\leftarrow \{\}$;

while *(1)* **do**

  for *each pair of clauses* C_i, C_j *in* `clauses` **do**

    resolvents $\leftarrow$ `PL-Resolve`(C_i, C_j)

    if `resovents` *contains the empty clause* **then**

      `return` true

    end

    new $\leftarrow$ new $\cup$ resolvents

  end

  if `new` $\subseteq$ `clauses` **then**

    `return` false

  end

end

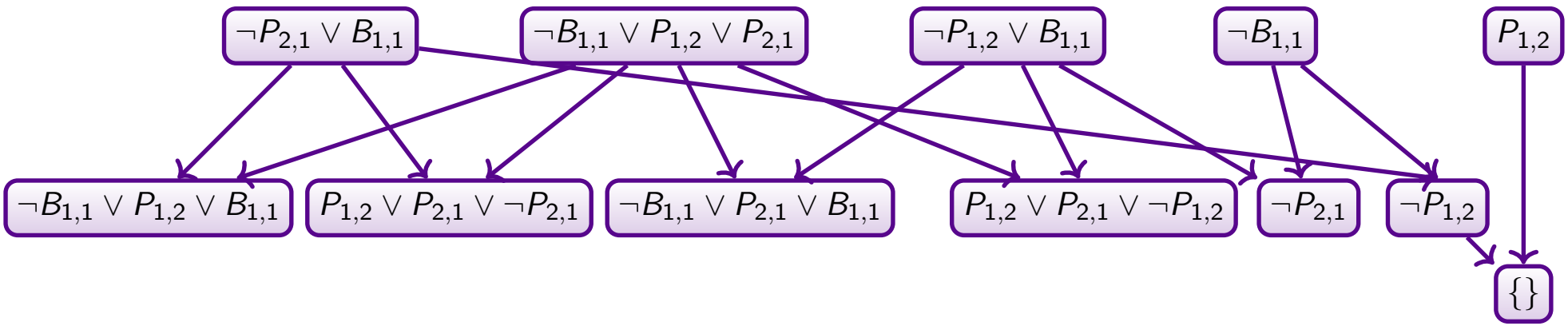
Algorithm 1: Propositional resolution algorithm

Resolution algorithm

- As an example, let us go back to the Wumpus World.
- Let us assume that our knowledge base is given by

$$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{1,2})) \wedge \neg B_{1,1}$$

- The second row in the figure below shows clauses obtained by resolving pairs of clauses from the first row



$\neg P_{1,2}$ is shown to follow from the four clauses on the top row.

Resolution algorithm: completeness

- For a set of clauses S, we introduce the resolution closure of S: $CL(S)$, as the set of all closes that can be derived by repeated application of the resolution rule to clauses in S or their derivatives.
- The resolution closure is exactly what `PL-Resolve`(C_i, C_j) computes. It is stored in the final state of the variable $'$`clause`$'$
- Because of the factoring step which removes multiple copies of a literal, there can only be finitely many distinct clauses that can be generated from the symbols $P_1, \ldots P_k$ that appear in S
- As a consequence `PL-Resolution` always terminates

Resolution algorithm: completeness

- We have the following completeness theorem (known as Ground Resolution Theorem):

  If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause.

Horn clauses

- The completeness of resolution makes it a very important inference method
- In many practical situations, however, the full power of resolution is not needed
- Some real world knowledge bases satisfy certain restrictions on the form of sentences they contain, which makes it possible to use a more restricted and efficient inference algorithm
- One such restricted form is the definite clause which is a disjunction of literals of which exactly one is positive. $(\neg L_{1,1} \vee \neg \texttt{Breeze} \vee B_{1,1})$ is a definite clause whereas $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1})$ is not.

Horn clauses

- Slightly more general is the Horn clause which is a disjunction of literals of which at most one is positive.
- All definite clauses are Horn clauses, as are clauses with no positive literals (called goal clauses)
- Horn clauses are closed under resolution (if you resolve two Horn clauses you get back a Horn clause)

Horn clauses

- Knowledge bases containing only definite clauses are interesting for 3 reasons:

  - Every definite clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal. For example the definite clause $(\neg L_{1,1} \vee \neg \texttt{Breeze} \vee B_{1,1})$ can be written as the implication $(L_{1,1} \wedge \texttt{Breeze}) \Rightarrow B_{1,1}$. In Horn form, the premise is called the body and the conclusion is called the head.

  - Inference with Horn clauses can be done through the forward and backward chaining algorithms. Those algorithms form the basis for logic programming

  - Deciding entailment with Horn clauses can be done in time that is linear in the size of the Knowledge base

Forward and Backward chaining

- The Forward chaining algorithm determines if a single proposition symbol q (the query) is entailed by a given knowledge base of definite clauses
- It begins from the known facts (positive literals) in the knowledge base.
- If all the premises of an implication are known, then its conclusion is added to the set of known facts
- For example if $L_{1,1}$ and $\texttt{Breeze}$ are known, and $(L_{1,1} \wedge \texttt{Breeze}) \Rightarrow B_{1,1}$ is in the knowledge base, then $B_{1,1}$ can be added

```
Function PL_ForwardChaining(KB, query q):
    input  : KB (initial state of the env.)
             q the query (propositional symbol)
    count (count[c] = num. of symbols in c's premises)
    inferred (inferred[s] is set to false for all symbols)
    agenda (queue of symbols known to be true in KB)
    while agenda is not empty do
        p ← Pop(agenda)
        if p = q then
            return True
        end
        if inferred[p] is False then
            inferred[p]← True
            for each clause c in KB where p is in c.Premises do
                decrement count[c]
                if count[c] = 0 then
                    add c.Conclusion to agenda
                end
            end
        end
    end
    return false
```

Algorithm 2: Propositional Forward Chaining

Forward and Backward chaining

- The agenda keeps track of the 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
- When a symbol p from the agenda is processed, the count is reduced by 1 for each implication in whose premises p appears
- If a count reaches 0, all the premises of the implication are known, so its conclusion can be added to the agenda

Forward and Backward chaining

- Consider the example below. The KB consists of a set of Horn clauses with the literals A and B as known facts .
- The knowledge base can be represented as an **AND-OR** graph

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

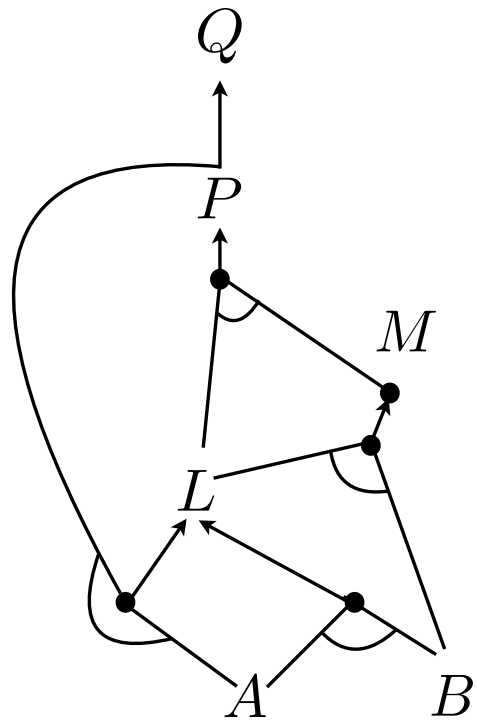
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chaining

- In **AND-OR** graphs, multiple links joined by an arc indicate a conjunction (every link must be proved)
- While multiple links without an arc indicate a disjunction (any link can be proved)

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

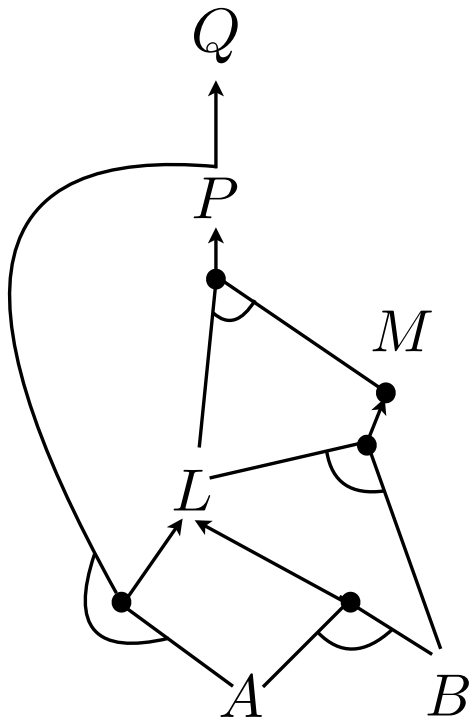
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chaining

- Forward chaining works by starting from the known leaves (here A and B)
- Inference then propagates up the graph as far as possible
- Whenever a conjunction appears, Forward Propagation waits until all the conjuncts are known before proceeding

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

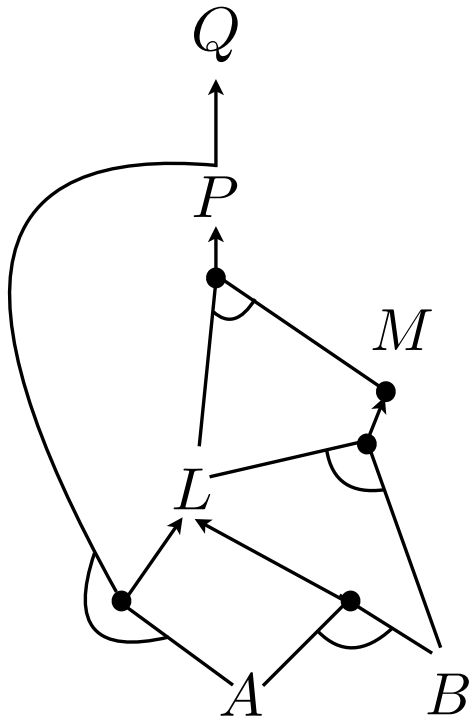
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chaining

- Forward chaining is an example of the general concept of Data-driven reasoning (=reasoning in which the focus of attention starts with the known data)

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

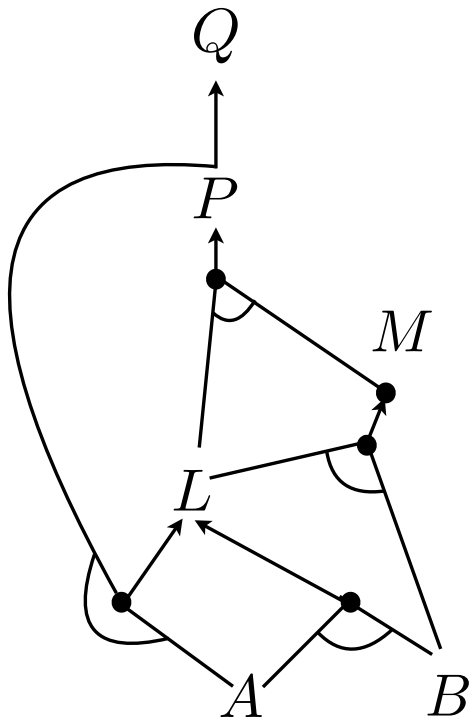
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chaining

- The Backward chaining, as its name indicates works backward from the query.
- If the query is known to be true, then no work is needed

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

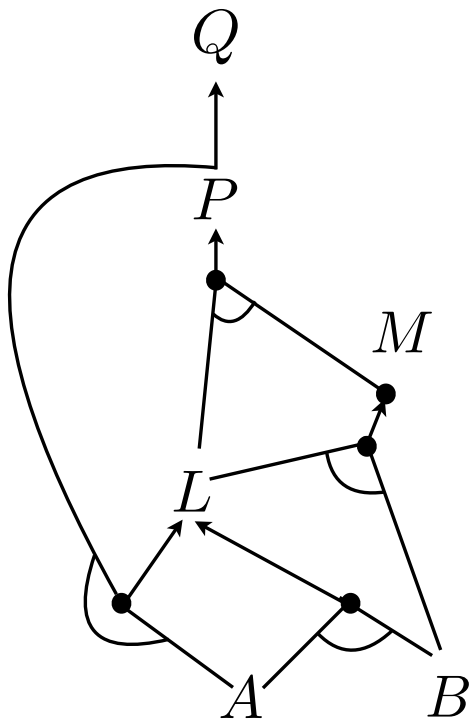
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chainging

- Otherwise, the algorithm finds those implications in the knowledge base whose conclusion is q. If all the premises of one of those implications can be proved true (by backward chaining), then q is true

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

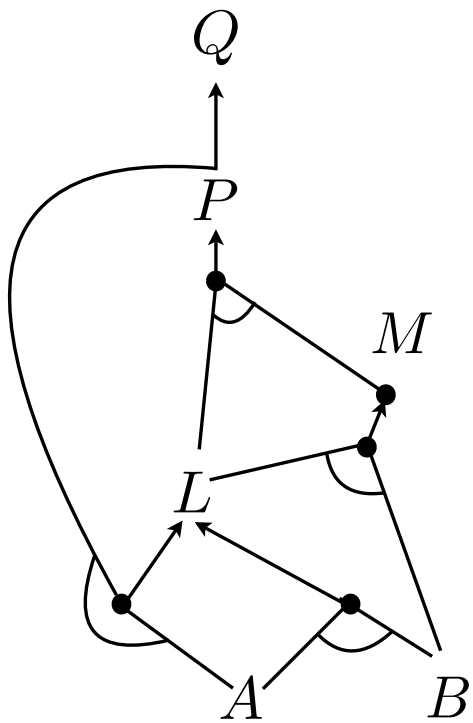
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chainging

- When applied to the query Q below, the algorithm works downwards until it reaches a set of known literals, A and B that form the basis for a proof.
- As for forward chaining, efficient implementations can run in linear time

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

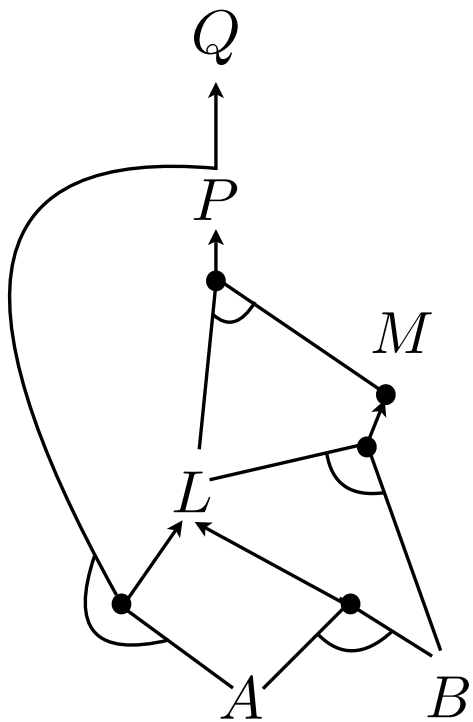
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Forward and Backward chainging

- Backward chaining is an example of Goal-directed reasoning

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

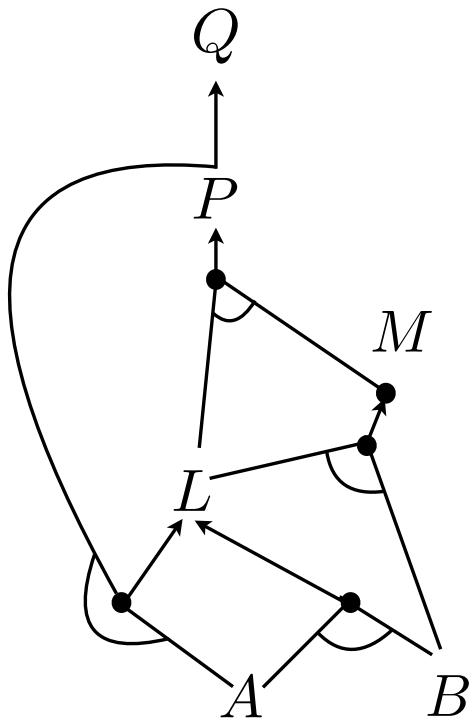
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Agents based on Propositional logic

- Note that so far, we have considered percepts only about space. In fact a percept asserts something only for the current time.
- This idea of associating proposition with time steps must apply to any aspect of the world that changes through time
- We use the word fluent (from the latin *fluens*, flowing) to refer to an aspect of the world that changes through time.
- Using this, we can connect the current stench and breeze percepts directly to the properties of the squares where they are experienced by relying on the location fluents, for any time step t and location $[x, y]$, we assert

$$L^t_{x,y} \Rightarrow (\text{Breeze}^t \Leftrightarrow B_{x,y})$$
$$L^t_{x,y} \Rightarrow (\text{Stench}^t \Leftrightarrow S_{x,y})$$

Agents based on Propositional logic

- Now we need additional axioms that allow the agent to keep track of the fluents such as $L^t_{x,y}$
- In other words, we need to write down a transition model of the Wumpus world as a set of logical sentences
- We can start by defining effect axioms that specify the outcome of an action at the next time step
- As an example, if the agent is in square $[1,1]$ and facing East at time 0 and goes Forward, the result is that the agent is in square $[2,1]$ and no longer is in $[1,1]$,

$$L^0_{1,1} \wedge \text{Facing East}^0 \wedge \text{Forward}^0 \Rightarrow (L^1_{2,1} \wedge \neg L^1_{1,1})$$

Agents based on Propositional logic

- Now imagine that at step 1 we make the query `Ask(KB,` $\texttt{HaveArrow}^1$`)` to determine whether the agent still has the arrow (note that we ask for the arrow at time $t = 1$).
- The answer to this query is false. The agent cannot prove that it has the arrow at time $t = 1$.
- The information has been lost because the effect axiom fails to state what remains unchanged as the result of an action
- The need to keep track of such facts is known as the frame problem (the name comes from the relation to the frame of reference in physics, the assumed stationary background)

Agents based on Propositional logic

- One solution to the frame problem would be to explicitely add all the frame axioms , explicitely asserting all the proposition that remain the same
- For each time t, we would then have

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

- The proliferation of frame axioms seems remarkably inefficient
- In a world with m different actions and n fluents, the set of frame axioms will be of size $O(mn)$

Agents based on Propositional logic

- This specific manifestation of the frame problem is sometimes known as the representational frame problem.
- Historically, the problem was a significant one for AI researchers
- The representational frame problem is important because the real world has very many fluents
- Fortunately for us humans, each action only changes no more than a small number k of those fluents. The world exhibits locality
- Solving the representational frame problem involves defining the transition model with a set of axioms of size $O(mk)$ rather than $O(mn)$ (m actions being affected by the k fluents)

Agents based on Propositional logic

- There is also an inferential frame problem which consists in projecting forward the result of a t steps plan of actions in time $O(kt)$ instead of $O(nt)$.
- While the representational FP means finding a succinct way of specifying all non-effects of actions, the inferential FP is the problem of effectively computing these non-effects. The inferential Frame Problem arises whenever the value of a fluent in one situation has to be derived from its value in previous situations.
- In the worst case scenario, every such fluent value needs to be carried stepwise from one situation to the other resulting in $O(nt)$ values for t time steps
- The solution to the problem involves changing one's focus from writing axioms about actions (see Forward^t in the above) to writing axioms about fluents

Agents based on Propositional logic

- For each Fluent, we have an axiom that defines the truth value of F^{t+1} in terms of Fluents (including F) at time t and the actions that may have occured at time t.
- The truth value of F^{t+1} can be set in one of two ways:
  - Either the action at time t causes F to be true at time $t+1$
  - or F was already true at time t and the action at time t does not cause it to be false

  $$F^{t+1} \Leftrightarrow \text{ActionCauses}F^t \vee (F^t \wedge \neg \text{ActionCausesNot}F^t)$$
- At each step, we thus only consider the k_1 fluents that are affected by the action and the k_2 fluents which were true at the previous step. An axiom of this form is called a successor-state axiom
- We can now encode the 'HaveArrow' fluent more simply as

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

Agents based on Propositional logic

- Solving the representational and inferential frame problems is a big step forward but a pernicious problem remains: we need to confirm that all the necessary preconditions of an action holds for it to have its intended effect
- I.e. we say that the Forward action moves the agent ahead unless there is a wall in the way, but there are many other unsual exceptions that could cause the action to fail: the agent might trip and fall, be stricken with a heart attack,...
- The need to specify all those exceptions is known as the qualification problem
- There is no complete solution to this problem within logic. System designers have to use good judgment in deciding how detailed they want to be in specifying their model, and what details they want to leave out.

Limitations of Propositional logic

- Recall Aristotle's celebrated syllogism

All men are mortal.
Socrates is a man.
Therefore, Socrates is mortal.

- Given A true, B true, we cannot prove the implication $A \wedge B \Rightarrow C$. Yet in plain English, this is clearly true

Combining the best of formal and natural languages

- We can adopt the foundations of Propositional logic, a declarative, compositional (meaning of a sentence is a function of the meaning of its parts) semantic and try to build a more expressive logic borrowing representational ideas from natural languages while avoiding its drawbacks.

- When you look at the syntax of natural language, the most obvious elements are nouns and noun phrases that refer to objects (squares, pits, Wumpuses,..), and verb and verb phrases that refer to relations among objects (is breezy, is adjacent to, shoots,..). Some of these relations are functions (in which there is only one value for a given input)

Combining the best of formal and natural languages

- It is relatively easy to list examples of objects, relations and functions:
  - Objects: people, houses, numbers, theories, colors, baseball games, table,...
  - Relations: these can be unary relations or properties such as red, round, bogus, prime,... or more general n-ary relations such as 'brother of', 'bigger than', 'inside', 'part of',...
  - Functions: 'one more than', 'father of',...
- The language of first order logic is built around objets and relations

Combining the best of formal and natural languages

- Remember the difference between ontological (what each language assumes about the nature of reality) commitment and epistemological (possible states of knowledge allowed by the language with respect to each fact) commitment?
- The principal difference between Propositional and First order logic lies in their respective ontological commitment
- Propositional logic assumes that there are facts that either hold or do not hold in the world. Each fact can be in either of two states: True or False
- First order logic assumes more: the world consists of certain objects with relations among them that do or do not hold.
- The epistemological commitment is however identical: a sentence represents a fact and the agent either believes the sentence to be true believes it to be false, or has no opinion

FOL Syntax (I)

- The basic syntactic elements of First Order Logic are the symbols that stand for objects, relations and functions
- The symbols come in three kind: constant Symbols, which stand for objects, predicate symbols which stand for relations, and function symbols, which stand for functions.
- We will adopt the convention that those symbols begins with uppercase letters, such as in 'Richard' and 'John', the predicate symbol 'Brother' and the function 'LeftLeg'
- Keep in mind that if Ω is the domain of the variables (set of all constants), a function takes zero or more arguments of that domain and returns another argument from that domain. A predicate takes zero or more arguments of that domain and returns a Boolean value.

FOL Syntax (II)

- A term is a logical expression that refers to an object. Constant symbols are thus terms. It is not always convenient to have a distinct symbol to refer to each object. We might and will instead want to use function symbols.

- Just as in natural language we might use the sentence 'King John's left leg' instead of giving an explicit name to the considered leg. We will encode this through the (function) symbol 'LeftLeg(John)'

- The formal semantics is clear: for a function $f(t_1, \ldots, t_n)$, the function symbol f refers to some function of the model, the argument terms refer to some objects in the domain (let us call them $d_1, \ldots, d_n$), and the term as a whole refers to the object that is the value of the function F applied to those objects $d_1, \ldots, d_n$

FOL Syntax (III)

- Once we have introduce terms for referring to objects and predicate symbols to refer to relations, we can put them together to make atomic sentences that state facts.
- An atomic sentence is formed from a predicate symbol optionally followed by a parenthesized list of terms such as

$$\text{Brother}(\text{Richard}, \text{John})$$

(Richard the LionHeart is the brother of King John)

- Atomic sentences can have complex arguments such as below

$$\text{Married}(\text{Father}(\text{Richard}), \text{Mother}(\text{John}))$$

- An atomic sentence is true in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments

FOL Syntax (IV)

- We can then use logical connectives to build more complex sentences.
- Some examples include

$$\neg \text{Brother}(\text{LeftLeg}(\text{Richard}), \text{John})$$
$$\text{Brother}(\text{Richard}, \text{John}) \wedge \text{Brother}(\text{John}, \text{Richard})$$
$$\text{King}(\text{Richard}) \vee \text{King}(\text{John})$$
$$\neg \text{King}(\text{Richard}) \Rightarrow \text{King}(\text{John})$$

FOL Syntax (V)

- Once we have a logic that allows objects, it seems natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. Quantifiers let us do this.
- First Order Logic contains two standard quantifiers called universal and existential
- Recall the difficulty we had in propositional logic with the expression of general rules, 'All men are mortal'
- The first sentence 'All men are mortal' in Aristotle syllogism can now be written as

$$\forall\, x\ \mathrm{Human}(x) \Rightarrow \mathrm{Mortal}(x)$$

- The symbol x is called a variable

FOL Syntax (VI)

- Consider the sentence $\forall x\ \text{King}(x) \Rightarrow \text{Person}(x)$
- Intuitively the sentence $\forall x\ P$ where P is any logical expression, says that P is true for every object x.
- More precisely, we say that $\forall x\ P$ is true in a given model if P is true in all possible extended interpretations constructed from the interpretation given in the model. Possible extensions are for example given by

$$x \rightarrow \text{Richard the LionHeart}$$
$$x \rightarrow \text{King John}$$
$$x \rightarrow \text{Richard's left leg}$$
$$x \rightarrow \text{john's left leg}$$
$$x \rightarrow \text{the crown}$$

FOL Syntax (VII)

- In such a model the universal quantifier would then be equivalent to asserting the following sentences

  Richard the Lionheart is a king $\Rightarrow$ Richard LH is a person

  King John is a king $\Rightarrow$ King John is a person

  Richard's Left Leg is a king $\Rightarrow$ Richard's left leg is a person

  ...

- We know that the second implication is true because the premise is true.

- What about the remaining 2 sentences?

FOL Syntax (IX)

- Universal quantification makes statements about every objects, Similarly, we can make a statement about some object in the universe without naming it, by using an existential quantifier
- To say for example that King John has a crown on his head, we can write

$$\exists\, x\ \mathsf{Crown}(x) \wedge \mathsf{OnHead}(x, \mathsf{John})$$

FOL Syntax (X)

- The sentence $\exists x\ P$ says that P is true for at least 1 object x.
- More precisely, $\exists x\ P$ is true in a given model if P is true in at least one extended interpretation that assigns x to a domain element. That is if we again consider the model containing the objects

  $x \rightarrow$ Richard the LionHeart

  $x \rightarrow$ King John

  $x \rightarrow$ Richard's left leg

  ...

- At least one of the following sentences must be true
  - R. the Lionheart is a crown $\wedge$ R. the Lionheart is on John's head
  - King J. is a crown $\wedge$ King J. is on John's head
  - R.'s left leg is a crown $\wedge$ R. left leg is on J's head
  - ...

FOL Syntax (XII)

- In FOL, we will often want to express more complex sentences using multiple quantifiers.
- As an example, the statement 'Brothers are siblings' can be written as

$$\forall x,\ \forall y\ \text{Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$$

- The order of the quantifiers is also very important. As an example consider the two statements 'Everybody loves somebody' and 'There is someone that is loved by everyone'. Those two statements would be respectively encoded as

$$\forall x,\ \exists y\ \text{Love}(x, y), \quad \exists y, \forall x\ \text{Love}(x, y)$$

FOL Syntax (XIII)

- The two quantifiers are actually intimately connected, through negation.
- Asserting that everyone dislikes meat is the same as asserting that there does not exist anyone who likes meat. I.e.

  $\forall x,\ \neg \text{Likes}(x, \textit{meat}), \quad$ is equivalent to $\quad \neg\exists x,\ \text{Likes}(x, \text{meat})$

- Similarly, the sentence 'everyone likes icecream' can be encoded as the fact that no one does not like ice cream:

  $\forall x\ \text{Likes}(x, \text{IceCream}), \quad$ is equivalent to $\quad \neg\exists x\ \neg\text{Likes}(x, \text{IceCream})$

FOL Syntax (XIII)

- $\forall$ can be understood as a conjunction over the universe of objects and $\exists$ can be understood similarly as a disjunction
- As a consequence, we can apply DeMorgan Law which for example gives

$$\begin{array}{ll} \forall x, \neg P \equiv \neg \exists x\ P & \neg(P \vee Q) \equiv \neg P \wedge \neg Q \\ \neg \forall x,\ P \equiv \exists x\ \neg P & \neg(P \wedge Q) \equiv (\neg P \vee \neg Q) \\ \forall x,\ P \equiv \neg \exists x\ \neg P & P \wedge Q \equiv \neg(\neg P \vee \neg Q) \\ \exists x\ P \equiv \neg \forall x\ \neg P & P \vee Q \equiv \neg(\neg P \wedge \neg Q) \end{array}$$

FOL Syntax (XIII)

- First order logic also enables the use of the equal sign '=' to indicate that two terms refer to the same object. For example

$$\text{Father}(\text{John}) = \text{Henry}$$

- The equal sign can be used to state facts about given functions. it can also be used with the negation to insist on the fact that two terms are not the same. For example, if we wan to say that Richard has at least two brothers, we would write

$$\exists\, x, y\ \text{Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard}) \wedge \neg(x = y)$$