Lecture 5

Logical Agents
Propositional Logic

Marco Chiarandini

Deptartment of Mathematics & Computer Science
University of Southern Denmark

Slides by Stuart Russell and Peter Norvig

Course Overview

- ✔ Introduction
  - ✔ Artificial Intelligence
  - ✔ Intelligent Agents
- ✔ Search
  - ✔ Uninformed Search
  - ✔ Heuristic Search
- ✔ Adversarial Search
  - ✔ Minimax search
  - ✔ Alpha-beta pruning
- Knowledge representation and Reasoning
  - Propositional logic
  - First order logic
  - Inference
- Uncertain knoweldge and Reasoning
  - Probability and Bayesian approach
  - Bayesian Networks
  - Hidden Markov Chains
  - Kalman Filters
- Learning
  - Decision Trees
  - Maximum Likelihood
  - EM Algorithm
  - Learning Bayesian Networks
  - Neural Networks
  - Support vector machines

Outline

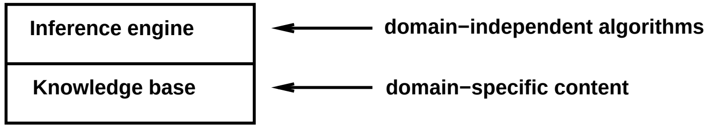
◇ Knowledge-based agents
◇ Wumpus world
◇ Logic in general—models and entailment
◇ Propositional (Boolean) logic
◇ Equivalence, validity, satisfiability
◇ Inference rules and theorem proving
  – forward chaining
  – backward chaining
  – resolution

Outline

1. Knowledge-based Agents
   Wumpus Example
2. Logic in General
3. Propositional Logic
   Equavalence and Validity
4. Inference in PL
   Proof by Resolution
   Proof by Model Checking

Knowledge bases

Knowledge base = set of sentences in a **formal** language



Declarative approach to building an agent (or other system):
Tell it what it needs to know

Then it can Ask itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level
i.e., **what they know**, regardless of how implemented

Or at the implementation level
i.e., data structures in KB and algorithms that manipulate them

A simple knowledge-based agent

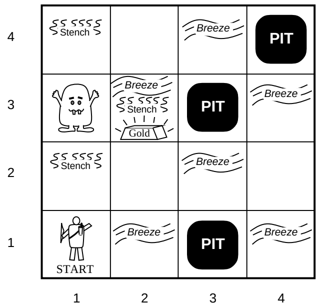
```
function KB-Agent( percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time

   Tell(KB, Make-Percept-Sentence( percept, t))
   action ← Ask(KB, Make-Action-Query(t))
   Tell(KB, Make-Action-Sentence(action, t))
   t ← t + 1
   return action
```

The agent must be able to:
- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

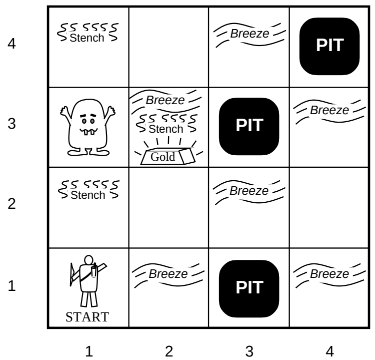
Wumpus World PEAS description

Performance measure
gold +1000, death -1000
-1 per step, -10 for using the arrow
Environment
Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter iff gold is in the same square
Shooting kills wumpus if you are facing it
Shooting uses up the only arrow
Grabbing picks up gold if in same square
Releasing drops the gold in same square
Actuators $LeftTurn$, $RightTurn$,
$Forward$, $Grab$, $Release$, $Shoot$
Sensors $Breeze$, $Glitter$, $Smell$

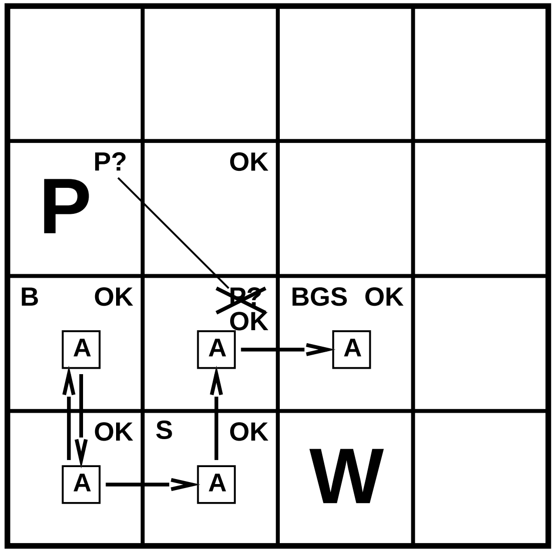


Wumpus world – Properties

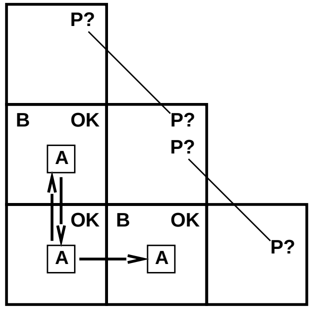
Fully vs Partially observable??
No—only local perception
Deterministic vs Stochastic??
Deterministic—outcomes exactly specified
Episodic vs Sequential??
sequential at the level of actions
Static vs Dynamic??
Static—Wumpus and Pits do not move
Discrete vs Continous??
Discrete
Single-agent vs Multi-Agent??
Single—Wumpus is essentially a natural feature



Exploring a wumpus world

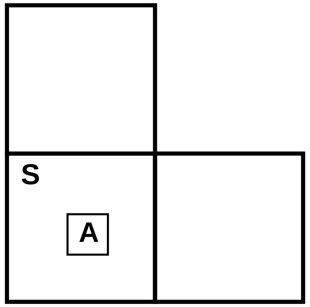


Other tight spots



Breeze in (1,2) and (2,1)
$\implies$ no safe actions

Assuming pits uniformly distributed,
(2,2) has pit w/ prob 0.86, vs. 0.31



Smell in (1,1)
$\implies$ cannot move
Can use a strategy of coercion:
shoot straight ahead
wumpus was there $\implies$ dead $\implies$ safe
wumpus wasn't there $\implies$ safe

Outline

Logic in general

Logics are formal languages for representing information
such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the "meaning" of sentences;
i.e., define truth of a sentence in a world

E.g., the language of arithmetic
$x+2 \geq y$ is a sentence; $x2+y>$ is not a sentence
$x+2 \geq y$ is true iff the number $x+2$ is no less than the number y
$x+2 \geq y$ is true in a world where $x=7,\ y=1$
$x+2 \geq y$ is false in a world where $x=0,\ y=6$

Entailment

Entailment means that one thing **follows from** another:

$KB \models \alpha$

Knowledge base KB entails sentence α
if and only if
α is true in all worlds where KB is true

E.g., the KB containing "OB won" and "FCK won"
entails "Either OB won or FCK won"

E.g., $x + y = 4$ entails $4 = x + y$

Entailment is a relationship between sentences (i.e., **syntax**)
that is based on **semantics**
Key idea: brains process **syntax** (of some sort)
trying to reproduce this mechanism

Models

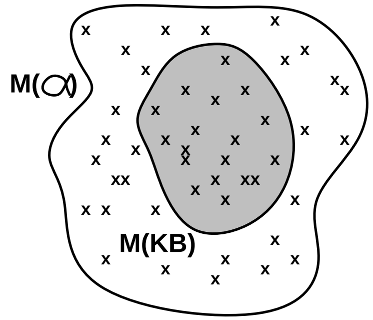
Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated

We say m is a model of a sentence α if α is true in m

$M(\alpha)$ is the set of all models of α

Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$

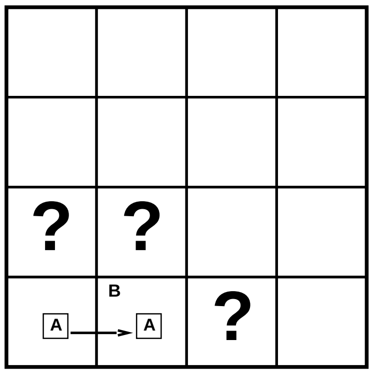
E.g. KB = OB won and FCK won
α = OB won



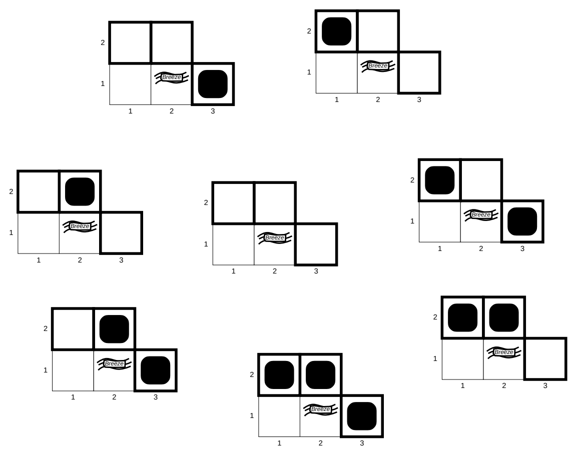
Entailment in the wumpus world

Situation after detecting nothing in [1,1],
moving right, breeze in [2,1]

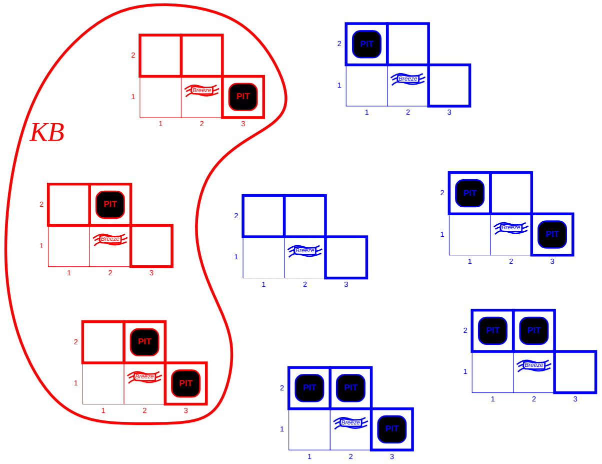
Consider possible models for ?s
assuming only pits
3 Boolean choices $\implies$ 8 possible models



Wumpus models

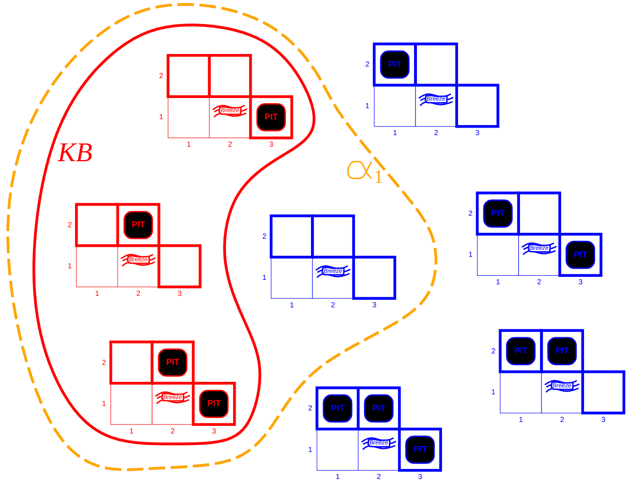


Wumpus models



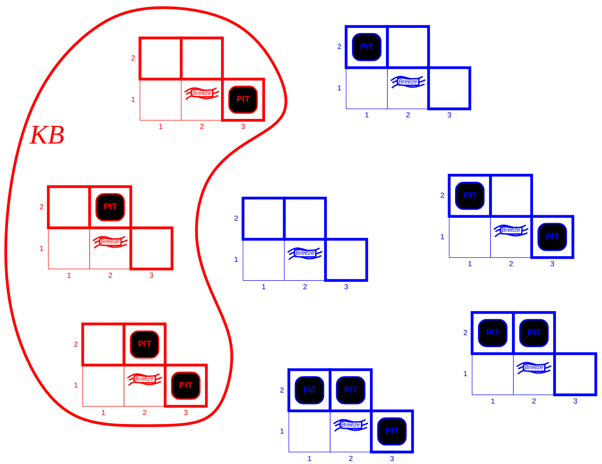
KB = wumpus-world rules + observations

Wumpus models



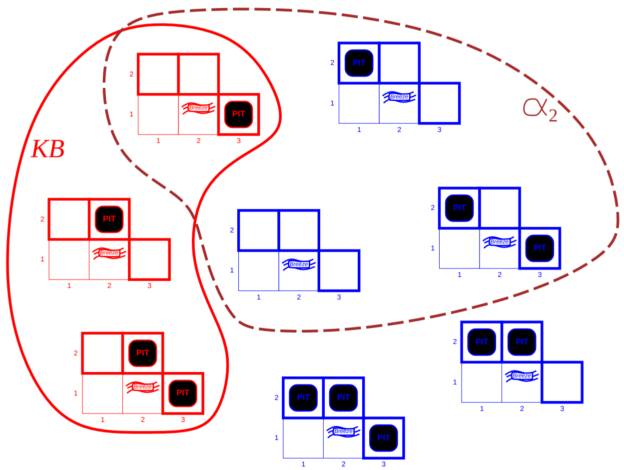
KB = wumpus-world rules + observations
α_1 = "[1,2] is safe", $KB \models \alpha_1$, proved by model checking

Wumpus models



KB = wumpus-world rules + observations

Wumpus models



KB = wumpus-world rules + observations
α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment = needle in haystack; inference = finding it

Soundness: i is sound if
whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if
whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Outline

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)
If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)
If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
If S_1 and S_2 are sentences, $S_1 \implies S_2$ is a sentence (implication)
If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.	$P_{1,2}$	$P_{2,2}$	$P_{3,1}$
	$true$	$true$	$false$

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false		
$S_1 \wedge S_2$	is true iff	S_1	is true **and**	S_2	is true
$S_1 \vee S_2$	is true iff	S_1	is true **or**	S_2	is true
$S_1 \implies S_2$	is true iff	S_1	is false **or**	S_2	is true
i.e.,	is false iff	S_1	is true **and**	S_2	is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \implies S_2$	is true **and**	$S_2 \implies S_1$	is true

Simple recursive process evaluates an arbitrary sentence, e.g.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.

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

"Pits cause breezes in adjacent squares"

Wumpus world sentences

Let $P_{i,j}$ be true if there is a pit in $[i,j]$.
Let $B_{i,j}$ be true if there is a breeze in $[i,j]$.

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

"Pits cause breezes in adjacent squares"

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

"A square is breezy **if and only if** there is an adjacent pit"

Truth tables for inference

$KB \vdash \alpha$

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	R_1	R_2	R_3	R_4	R_5	KB
false	*false*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*false*	*false*
false	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*false*	*true*	*false*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	*true*	*false*	*false*	*false*	*false*	*false*	*true*	*true*	*false*	*true*	*true*	*false*
false	*true*	*false*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*false*	*true*	*true*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*true*	*false*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
true	*true*	*true*	*true*	*true*	*true*	*true*	*false*	*true*	*true*	*false*	*true*	*false*

Enumerate rows (different assignments to symbols),
if KB is true in row, check that α is too

Inference by enumeration

Depth-first enumeration of all models is sound and complete

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 $\leftarrow$ 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*
  else do
    $P \leftarrow$ First(*symbols*); *rest* $\leftarrow$ Rest(*symbols*)
    return TT-Check-All(*KB*, α, *rest*, Extend(P, $true$, $model$)) **and**
      TT-Check-All(*KB*, α, *rest*, Extend(P, $false$, $model$))

$O(2^n)$ for n symbols; problem is **co-NP-complete**

Logical equivalence

Two sentences are logically equivalent iff true in same models:
$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

$$
\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{commutativity of } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{associativity of } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \implies \beta) & \equiv & (\neg\beta \implies \neg\alpha) & \text{contraposition} \\
(\alpha \implies \beta) & \equiv & (\neg\alpha \vee \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \implies \beta) \wedge (\beta \implies \alpha)) & \text{bicond. elimination} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{De Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{distributivity of } \vee \text{ over } \wedge
\end{array}
$$

Validity and satisfiability

A sentence is valid if it is true in **all** models,
  e.g., $True$, $A \vee \neg A$, $A \implies A$, $(A \wedge (A \implies B)) \implies B$

Validity is connected to inference via the Deduction Theorem:
  $KB \models \alpha$ if and only if $(KB \implies \alpha)$ is valid

A sentence is satisfiable if it is true in **some** model
  e.g., $A \vee B$, C

A sentence is unsatisfiable if it is true in **no** models
  e.g., $A \wedge \neg A$

Satisfiability is connected to inference via the following:
  $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable
i.e., prove α by *reductio ad absurdum*

Outline

Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules
- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.
- Typically require translation of sentences into a normal form

Model checking
truth table enumeration (always exponential in n)
improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
heuristic search in model space (sound but incomplete)
e.g., min-conflicts-like hill-climbing algorithms

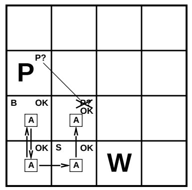
Resolution

Conjunctive Normal Form (CNF—universal)
conjunction of $\underbrace{\textbf{disjunctions} \text{ of } \textbf{literals}}_{\textbf{clauses}}$
E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Resolution inference rule (for CNF): complete for propositional logic

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

where ℓ_i and m_j are complementary literals. E.g.:



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

Resolution is sound and complete for propositional logic

Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \implies \beta) \wedge (\beta \implies \alpha)$.

   $(B_{1,1} \implies (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \implies B_{1,1})$

2. Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

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

3. Move $\neg$ inwards using de Morgan's rules and double-negation:

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

4. Apply distributivity law ($\vee$ over $\wedge$) and flatten:

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

Resolution algorithm

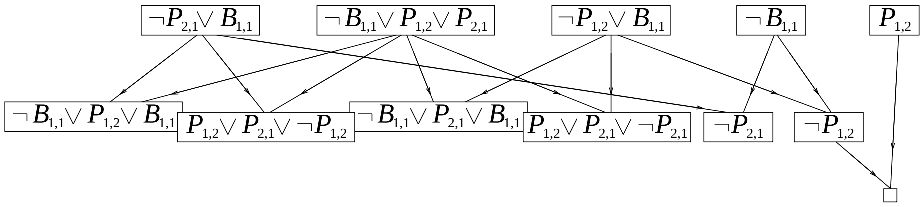
Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

```
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 ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
      for each Ci, Cj in clauses do
         resolvents ← PL-Resolve(Ci, Cj)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

Resolution example

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



Completeness of Resolution

Theorem

Ground Resolution Theorem
If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clauses

Proof. by contraposition $RC(S)$ does not contain empty clause $\implies$ S is satisfiable.
Construct a model for S with sutiable ttruth values for $P_1, \ldots, P_k$ as follows

- assign false to P_i if there is a clause in $RC(S)$ containing literal $\neg P_i$ and all its other literals being false under the current assignment
- otherwise, assign P_i true.

Forward and backward chaining

Horn Form (restricted)
KB = **conjunction** of **Horn clauses**
Horn clause =
- ♢ proposition symbol; or
- ♢ (conjunction of symbols) $\implies$ symbol

E.g., $C \wedge (B \implies A) \wedge (C \wedge D \implies B)$

Modus Ponens (for Horn Form): complete for Horn KBs

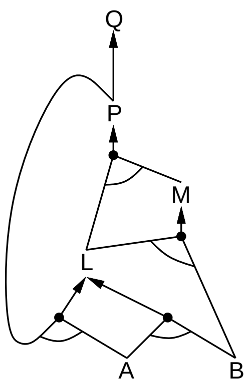
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \implies \beta}{\beta}$$

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in **linear** time

Forward chaining

Idea: fire any rule whose premises are satisfied in the KB,
add its conclusion to the KB, until query is found

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

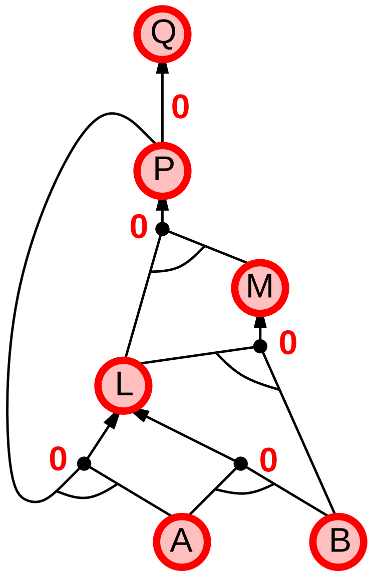


Forward chaining algorithm

```
function PL-FC-Entails?(KB, q) returns true or false
    inputs: KB, the knowledge base, a set of propositional Horn clauses
            q, the query, a proposition symbol
    local variables: count, a table, indexed by clause, initially number of
premises
                     inferred, a table, indexed by symbol, entries initially false
                     agenda, a list of symbols, initially symbols known in KB

    while agenda is not empty do
        p ← Pop(agenda)
        unless inferred[p] do
            inferred[p] ← true
            for each Horn clause c in whose premise p appears do
                decrement count[c]
                if count[c] = 0 then do
                    if Head[c] = q then return true
                    Push(Head[c], agenda)
    return false
```


Forward chaining example



Proof of completeness

FC derives every atomic sentence that is entailed by KB

- FC reaches a fixed point where no new atomic sentences are derived
- Consider the final state as a model m, assigning true/false to symbols
- Every clause in the original KB is true in m
  Proof: Suppose a clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in m
  Then $a_1 \wedge \ldots \wedge a_k$ is true in m and b is false in m
  Therefore the algorithm has not reached a fixed point!
- Hence m is a model of KB
- If $KB \models q$, q is true in **every** model of KB, including m

General idea: construct any model of KB by sound inference, check α

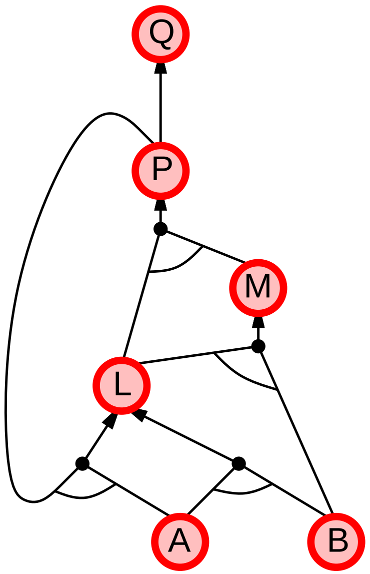
Backward chaining

Idea: work backwards from the query q:
  to prove q by BC,
    check if q is known already, or
    prove by BC all premises of some rule concluding q

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
  1) has already been proved true, or
  2) has already failed

Backward chaining example



Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD program?

Complexity of BC can be **much less** than linear in size of KB

Model Checking by Backtracking – DPLL

Davis Putnam Logeman and Loveland (1960-1962)

function DPLL-Satisfiable?(*s*) **returns** *true* or *false*
inputs: *s*, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of s
symbols ← a list of the proposition symbols in *s*
return DPLL(*clauses*, *symbols*, [ ])

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

if every clause in *clauses* is true in *model* **then return** *true*
if some clause in *clauses* is false in *model* **then return** *true*
P, *value* ← Find-Pure-Symbol(*symbols*, *clauses*, *model*)
if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, $[P = value|model]$)
P, *value* ← Find-Unit-Clause(*clauses*, *model*)
if *P* is non-null **then return** DPLL(*clauses*, *symbols*–*P*, $[P = value|model]$)
P ← First(*symbols*); *rest* ← Rest(*symbols*)
return DPLL(*clauses*, *rest*, $[P = true|model]$) **or** DPLL(*clauses*, *rest*, $[P = false|model]$)

Model Checking by Local Search

Walksat

function WalkSAT(*clauses*, *p*, *max-flips*) **returns** a satisfying model or *failure*
inputs: *clauses*, a set of clauses in propositional logic
p, the probability of choosing to do a "random walk" move, typically around 0.5
max-flips, number of flips allowed before giving up

model ← a random assignment of *true*/*false* to the symbols in *clauses*
for $i = 1$ **to** *max-flips* **do**
if *model* satisfies *clauses* **then return** *model*
clause ← a randomly selected clause from *clauses* that is false in *model*
with probability *p* flip the value in *model* of a randomly selected symbol from *clause*
else flip whichever symbol in *clause* maximizes the number of satisfied clauses
return *failure*

Summary

Logical agents apply inference to a knowledge base
to derive new information and make decisions

Basic concepts of logic:

- syntax: formal structure of sentences
- semantics: truth of sentences wrt models
- entailment: necessary truth of one sentence given another
- inference: deriving sentences from other sentences
- soundess: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power