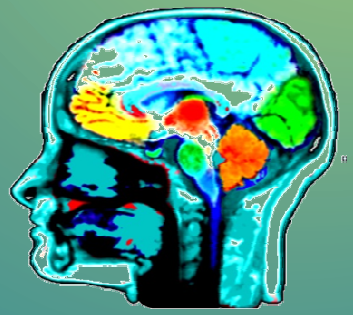
Introduction
To
Artificial Intelligence

Isfahan University of Technology (IUT)

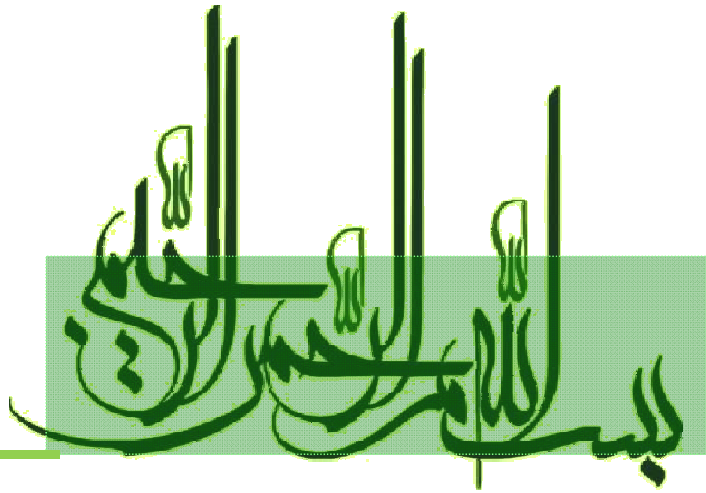
1402



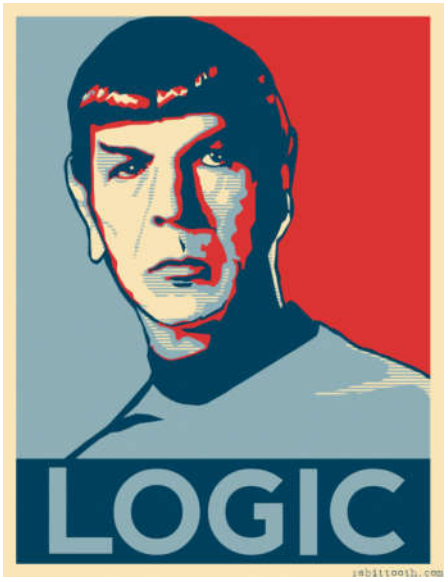
Propositional Logic

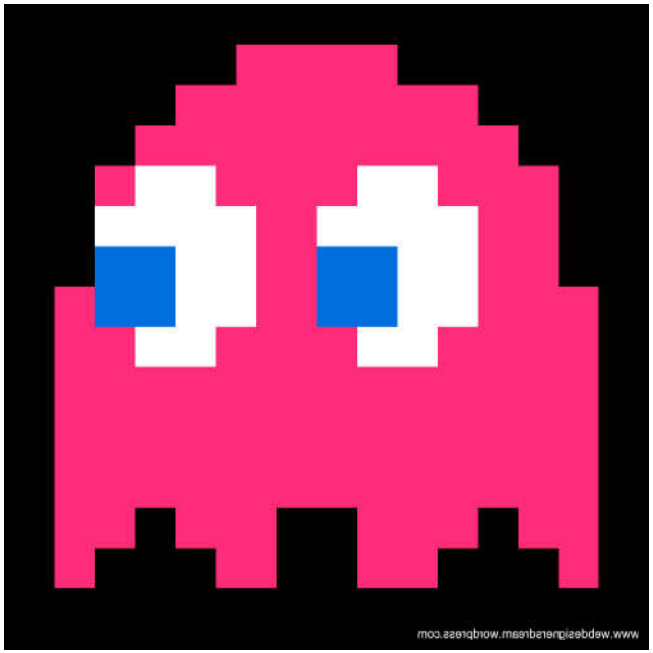
Dr. Hamidreza Hakim
hakim@iut.ac.ir

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.]


بسم الله الرحمن الرحيم

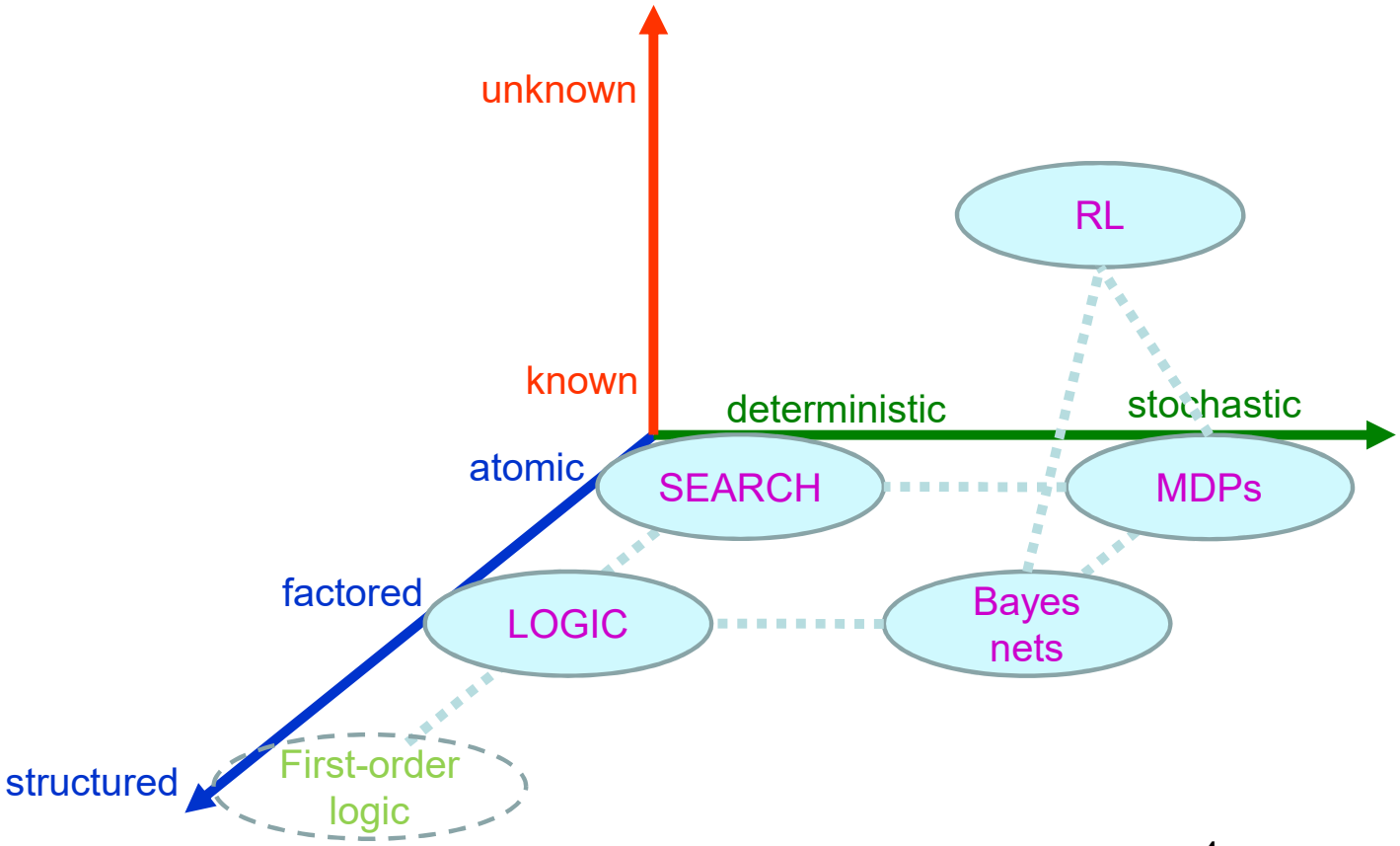
Propositional Logic





[These slides were created by Stuart Russell and Dawn Song for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Outline of the course



Outline

1. Propositional Logic I
   - Basic concepts of knowledge, logic, reasoning
   - Propositional logic: syntax and semantics,
   - Wumpus example
   - Logic in general model and entaiolment
   - Inference by theorem proving
   - Inference by model checking

Agents that know things

Humans, it seems, know things; and what they know helps them do things.

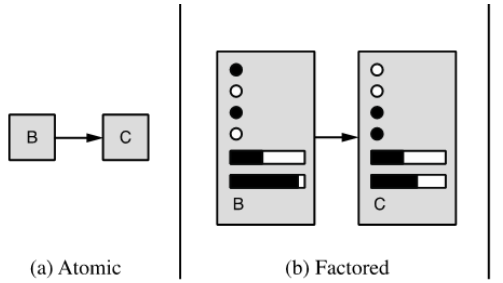
In AI, knowledge based agents use a process of **reasoning** over an **internal representation of knowledge** to decide what **actions** to take.

Agents that know things

The atomic representations:

- A route-finding agent doesn't know that it is impossible for a road to be a negative number of kilometers long
- An 8-puzzle agent doesn't know that two tiles cannot occupy the same space

They don't know general facts



Agents that know things

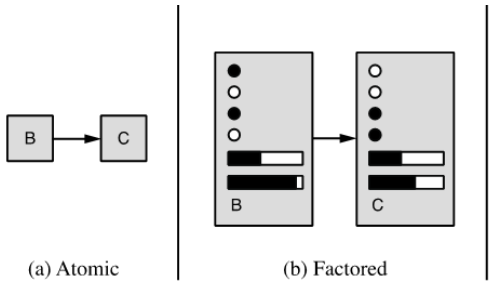
The Factor representations:

In CSP: states are represented as

assignments of values to variables;

enabling some parts of the agent to work in a domain-independent

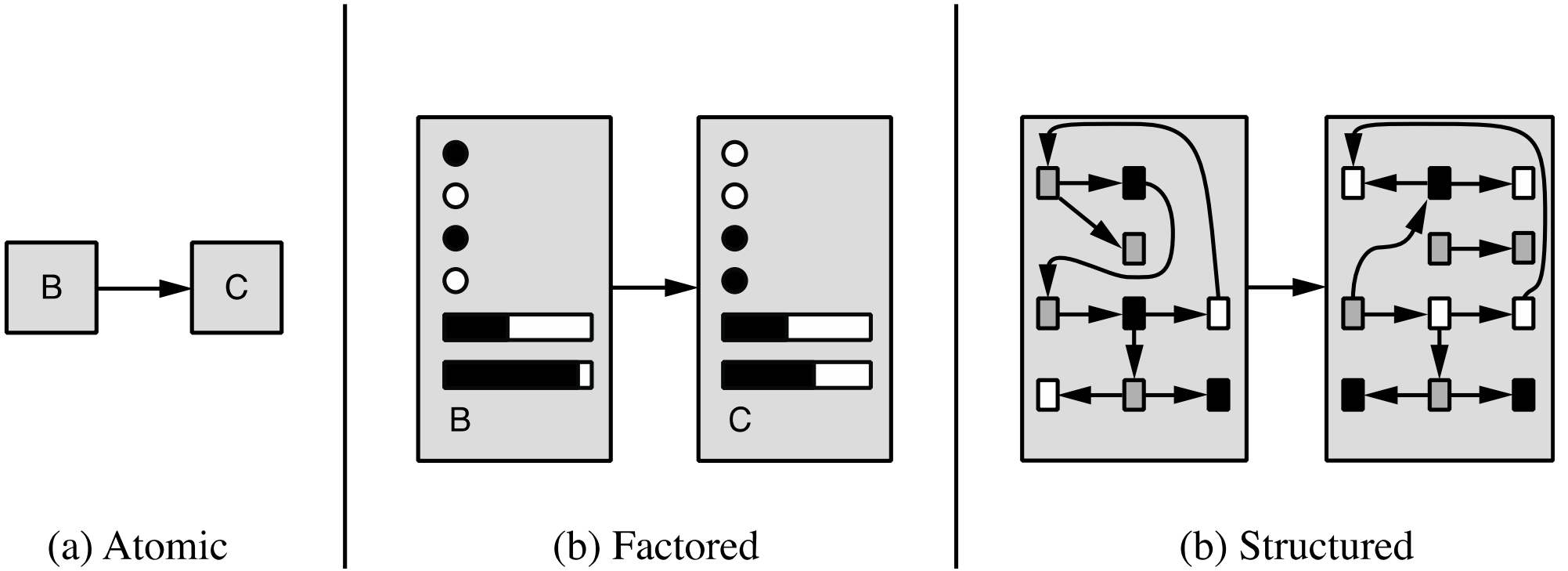
- we develop logic as a general class of representations to support knowledge-based agents



Agents that know things

- Knowledge-based agents can accept new tasks in the form of explicitly described goals;
- they can achieve competence quickly by being told or learning new knowledge about the environment;
- they can adapt to changes in the environment by updating the relevant knowledge

Spectrum of representations



(a) Atomic (b) Factored (b) Structured

Search, game-playing

CSPs, planning, propositional logic, Bayes nets, neural nets

First-order logic, databases, logic programs, probabilistic programs

Agents that know things

- Agents acquire knowledge through perception, learning, language
  - Knowledge of the effects of actions ("transition model")
  - Knowledge of how the world affects sensors ("sensor model")
  - Knowledge of the current state of the world
- Can keep track of a **partially observable** world
- Can formulate plans to achieve goals

Knowledge, contd.

- The central component of a knowledge-based agent is its **knowledge base**, or KB.

  Knowledge base = set of sentences in a formal language

- Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world

  There are two operations TELL and ASK

- To add new sentences to the knowledge base
- To query what is known

Knowledge, contd.

- Declarative approach to building an agent (or other system):
  - ***Tell*** it what it needs to know (or have it ***Learn*** the knowledge)
  - Then it can ***Ask*** itself what to do—answers should follow from the KB
  - Both operations may involve **inference—that** is,

    deriving new sentences from old

A knowledge-based agent

function KB-AGENT(percept) **returns** an action

persistent: KB, a knowledge base

t, an integer, initially 0

1. TELL(KB, MAKE-**PERCEPT**-SENTENCE(percept, t))
2. action ← ASK(KB, MAKE-ACTION-QUERY(t))
3. TELL(KB, MAKE-**ACTION**-SENTENCE(action, t))

t←t+1

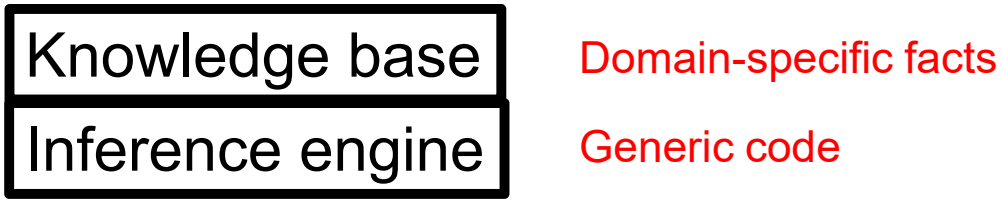
return action

A knowledge-based agent

- MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time.
- MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.
- MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed.

Knowledge, contd.

- Agents can be viewed at the ***knowledge level***
  i.e., what they ***know***, regardless of how implemented
- A single inference algorithm can answer any answerable question



An Example

- TELL: John is a male **person**
- TELL: Emily is a female **person**
- TELL: person is either male or female
- TELL: James is a parent of John
- TELL: James is a parent of Emily
- TELL: Siblings have the same parent

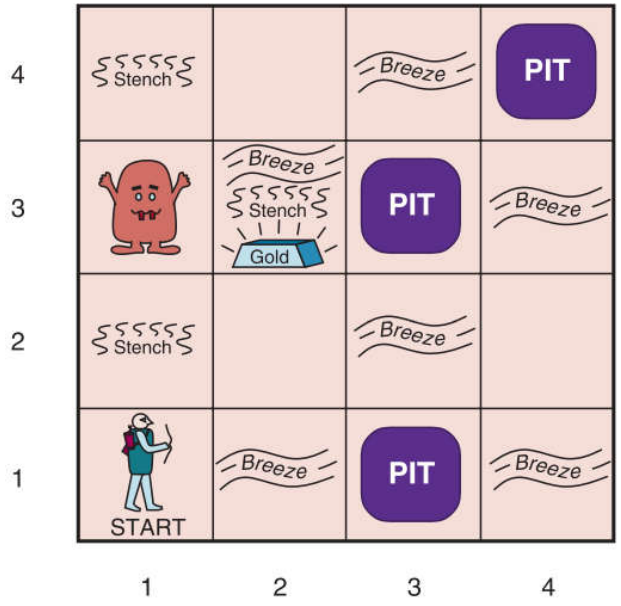
…

- ASK: Is Emily a sibling of John?

[40-417, SUT]

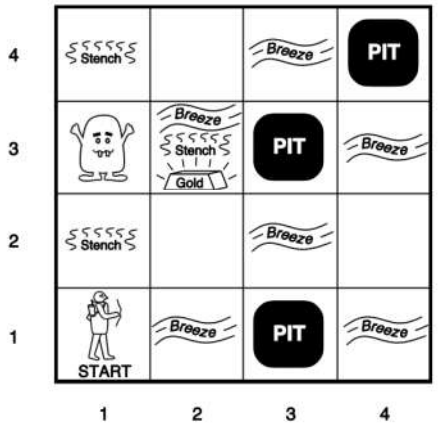
wumpus world

- Wumpus
- Pitts
- Gold
- Agent



wumpus world

- Performance measure
  - +1000 for garbing gold
  - -1000 for death
  - -1 for each action
  - -10 for using up the arrow

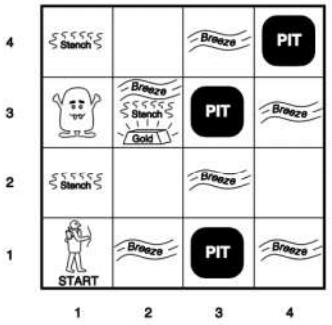


  - Game ends when the agent dies or climbs out of the cave

- Environment
  - 4×4 grid
  - Agent starts in [1,1] while facing to the right
  - Gold and wumpus are located randomly in the squares except to [1,1]
  - Each square other than [1,1] can be a pit with probability 0.2

wumpus world

- Sensors: Stench, Breeze, Glitter, Bump, Scream
  - In the squares adjacent to wumpus, agent perceives a Stench
  - It the squares adjacent to a pit, agent perceives a Breeze
  - In the gold square, agent perceives a Glitter
  - When walking into a wall, agent perceives a Bump
  - When Wumpus is killed, agent perceives a Scream



- Actuators: Forward, TurnLeft, TurnRight, Shoot, Grab, Climb
  - Forward, TurnLeft, TurnRight: moving and rotating face actions.
    - Moving to a square containing a pit or a live wumpus causes death.
    - If an agent tries to move forward and bumps into a wall then it does not move.
  - Shoot: to fire an arrow in a straight line in the facing direction of the agent
    - Shooting kills wumpus if the agent is facing it (o.w. the arrow hits a wall)
    - The first shoot action has any effect (the agent has only one arrow)
  - Grab: to pick up the gold if it is in the same square as the agent.
  - Climb: climb out of the cave but only from [1,1]

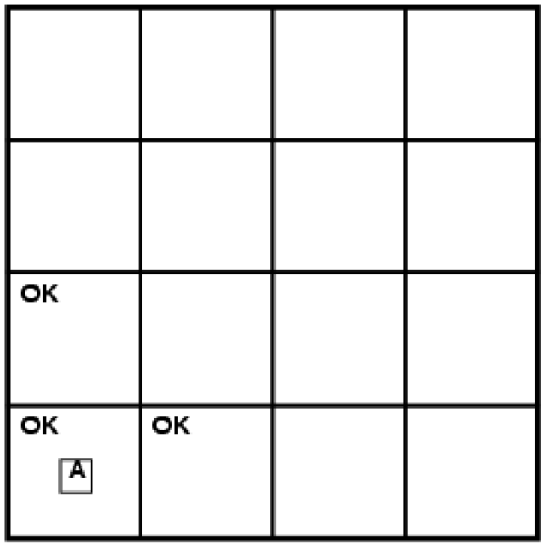
wumpus world characterization

- Fully_Observable? No – many aspects are not directly perceivable
- Episodic? No – sequential at the level of actions
- Static? Yes
- Discrete? Yes
- Single-agent? Yes

wumpus Agent

A: agent

OK: safe square

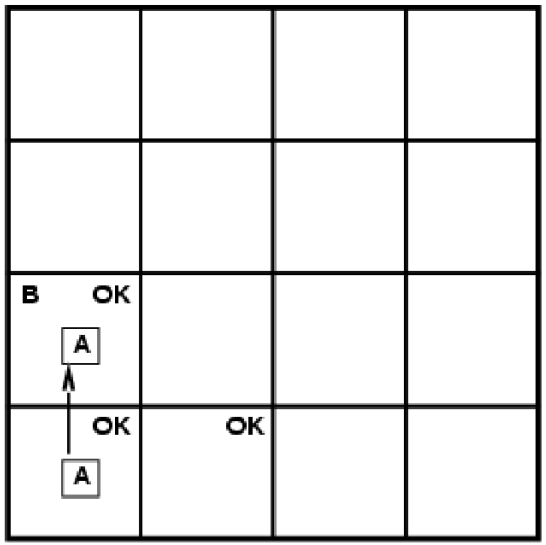


wumpus Agent

A: agent

OK: safe square

B: Breeze



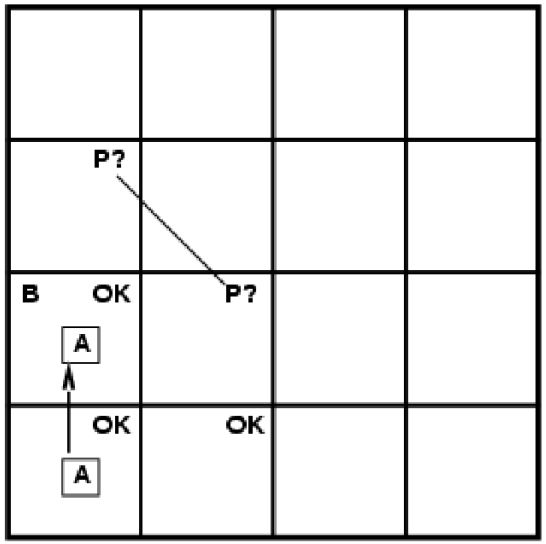
wumpus Agent

A: agent

OK: safe square

B: Breeze

P: Pit



wumpus Agent

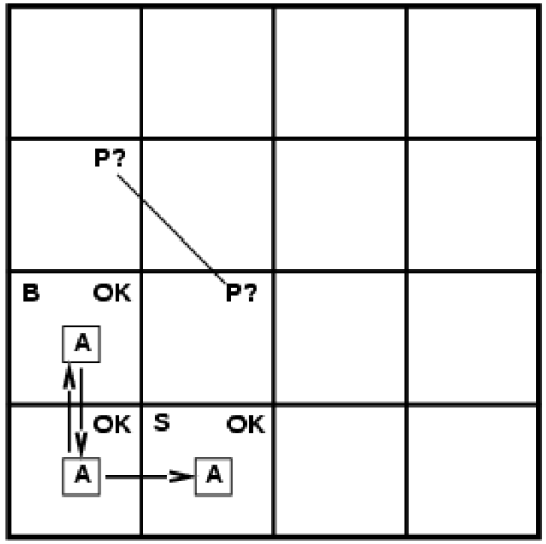
A: agent

OK: safe square

B: Breeze

P: Pit

S: Stench



wumpus Agent

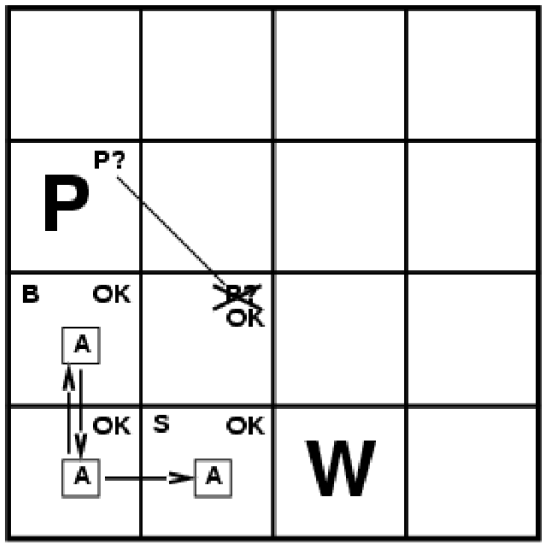
A: agent

OK: safe square

B: Breeze

P: Pit

S: Stench



wumpus Agent

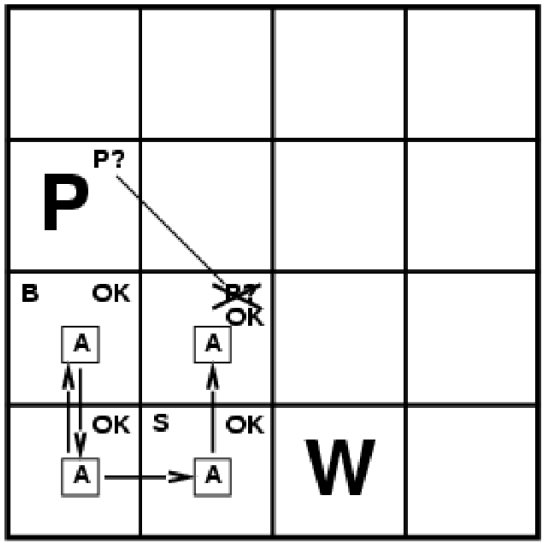
A: agent

OK: safe square

B: Breeze

P: Pit

S: Stench



wumpus Agent

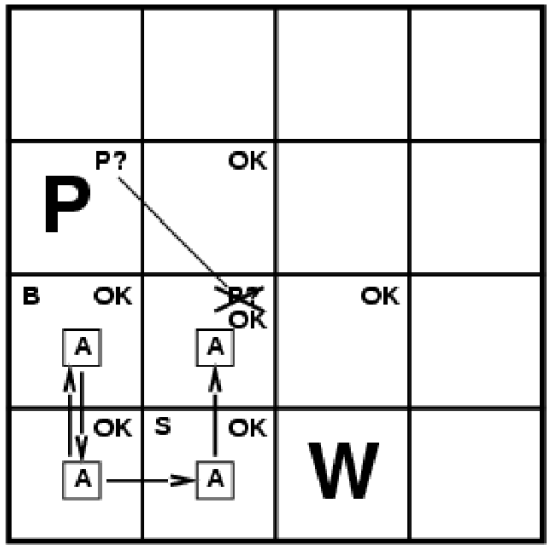
A: agent

OK: safe square

B: Breeze

P: Pit

S: Stench



wumpus Agent

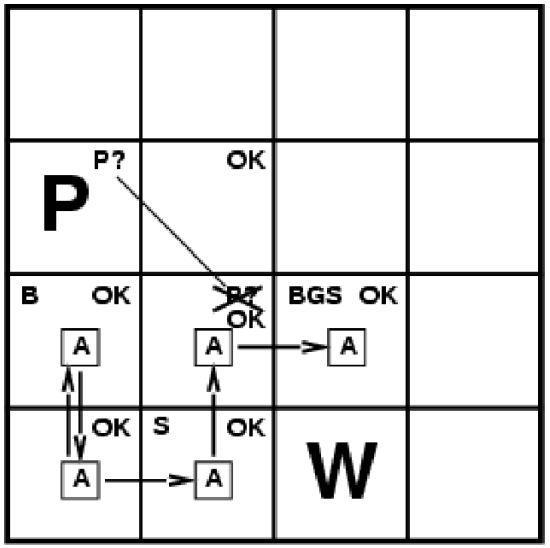
A: agent

OK: safe square

B: Breeze

P: Pit

S: Stench



- How Logic Agent do?

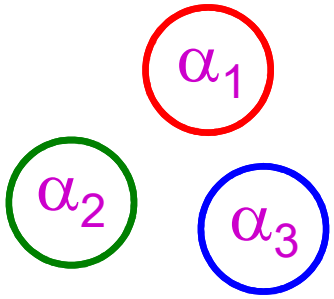
Lets Say some basic info

Logic

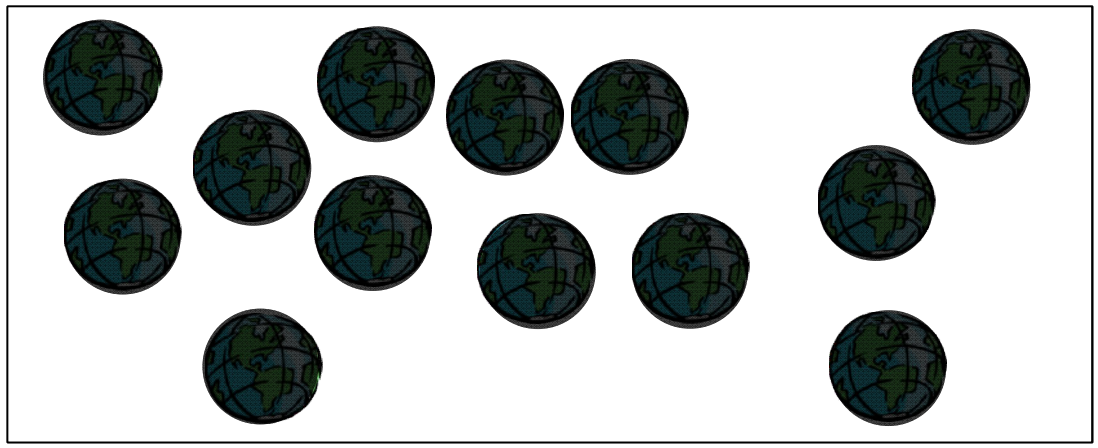
1. ***Syntax***: What sentences are allowed?
2. ***Semantics***:
    1. What are the ***possible worlds***?
    - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)
3. **Inference**

Logic

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***?
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)



Syntaxland



Semanticsland

Logic

- ***Syntax***: What sentences are allowed?
- ***Semantics***:
  - What are the ***possible worlds***?
  - Which sentences are ***true*** in which worlds? (i.e., ***definition*** of truth)
- E.g., the language of arithmetic
  - x+2 ≥ y is a sentence; x2+y > {} is not a sentence
  - x+2 ≥ y is true iff the number x+2 is no less than the number y
  - x+2 ≥ y is true in **a world** where x = 7, y = 1
  - x+2 ≥ y is false in **a world** where x = 0, y = 6

Different kinds of logic

- **Propositional logic (this slide)**
  - Syntax: $P \vee (\neg Q \wedge R)$; $X_1 \Leftrightarrow (\text{Raining} \Rightarrow \neg \text{Sunny})$
  - Possible world: {P=true,Q=true,R=false,S=true} or 1101
  - Semantics: $\alpha \wedge \beta$ is true in a world iff is α true and β is true (etc.)
- First-order logic
  - Syntax: $\forall x \exists y\ P(x,y) \wedge \neg Q(\text{Joe}, f(x)) \Rightarrow f(x)=f(y)$
  - Possible world: Objects o_1, o_2, o_3; P holds for $<o_1,o_2>$; Q holds for $<o_3>$; $f(o_1)=o_1$; $\text{Joe}=o_3$; etc.
  - Semantics: $\phi(\sigma)$ is true in a world if $\sigma=o_j$ and ϕ holds for o_j; etc.

  Ontology

  Other Logic: Fuzzy Logic, Probabilistic Logic

Propositional logic syntax

- Given: a set of proposition symbols $\{X_1, X_2, \ldots, X_n\}$
  - (we often add True and False for convenience)
  - Symbols, some ralation
- X_i is a sentence
- 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

And p.s. there are no other sentences!

- Create Complex Sentence
-

Propositional logic syntax

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \mid ComplexSentence \\
AtomicSentence &\rightarrow True \mid False \mid P \mid Q \mid R \mid \ldots \\
ComplexSentence &\rightarrow (Sentence) \\
&\quad \mid \neg Sentence \\
&\quad \mid Sentence \wedge Sentence \\
&\quad \mid Sentence \vee Sentence \\
&\quad \mid Sentence \Rightarrow Sentence \\
&\quad \mid Sentence \Leftrightarrow Sentence
\end{aligned}
$$

$Precedence: \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Truth tables for connectives

- Sentence are true or false

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*

Propositional logic semantics

- Let *m* be a **model** assigning true or false to {X_1,X_2,…, X_n}

  - e.g., Propistion symbols: $P_{1,2}, P_{2,2}, P_{3,1}$
    - 8 possible models
    - $m_1 = \{P_{1,2} = false, P_{2,2} = false, P_{3,1} = true\}$

Propositional logic semantics

- Let *m* be a **model** assigning true or false to $\{X_1, X_2, \ldots, X_n\}$
- Semantics of a complex sentence in any model m

If α is a symbol then its truth value is given in *m*

$\neg\alpha$ is true in *m* iff α is false in *m*

$\alpha \wedge \beta$ is true in *m* iff α is true in *m* and β is true in *m*

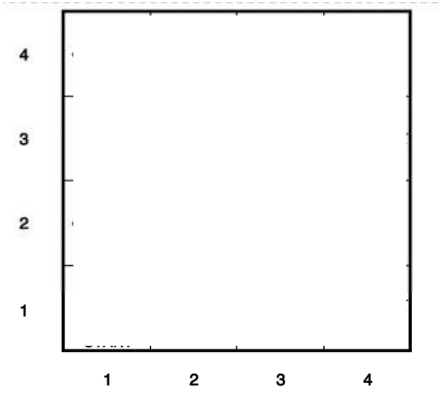
$\alpha \vee \beta$ is true in *m* iff α is true in *m* or β is true in *m*

$\alpha \Rightarrow \beta$ is true in *m* iff α is false in *m* or β is true in *m*

$\alpha \Leftrightarrow \beta$ is true in *m* iff $\alpha \Rightarrow \beta$ is true in *m* and $\beta \Rightarrow \alpha$ is true in *m*

Wumpus world sentences

- $P_{i,j}$ is true if there is a pit in $[i, j]$.
- $W_{i,j}$ is true if there is a wumpus in $[i, j]$, dead or alive.
- $B_{i,j}$ is true if the agent perceives a breeze in $[i, j]$.
- $S_{i,j}$ is true if the agent perceives a stench in $[i, j]$.



- General rules (only related ones to the current agent position)
  - R_1: $\neg P_{1,1}$ (no pit in [1,1])
  - R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ (Pits cause breezes in adjacent squares)
  - R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$ (Pits cause breezes in adjacent squares)

- Perception (After each Action)
  - R_4: $\neg B_{1,1}$
  - R_5: $B_{2,1}$

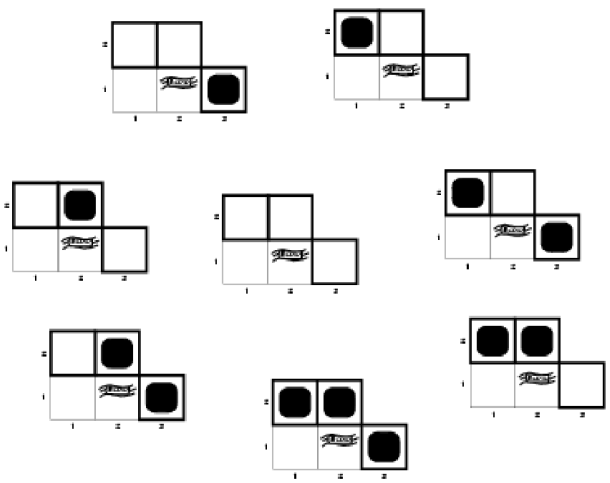
Model

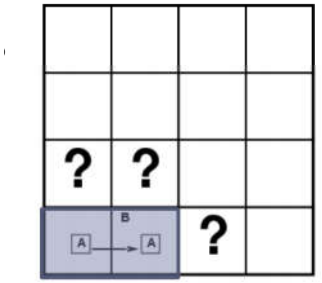
- Models are mathematical abstraction of **possible worlds(Sets)**
- Each model fixes the truth or falsehood of every relevant sentence.
- We can consider a set for each logical sentence that specifies all possible worlds in which α is true

Wumpus models

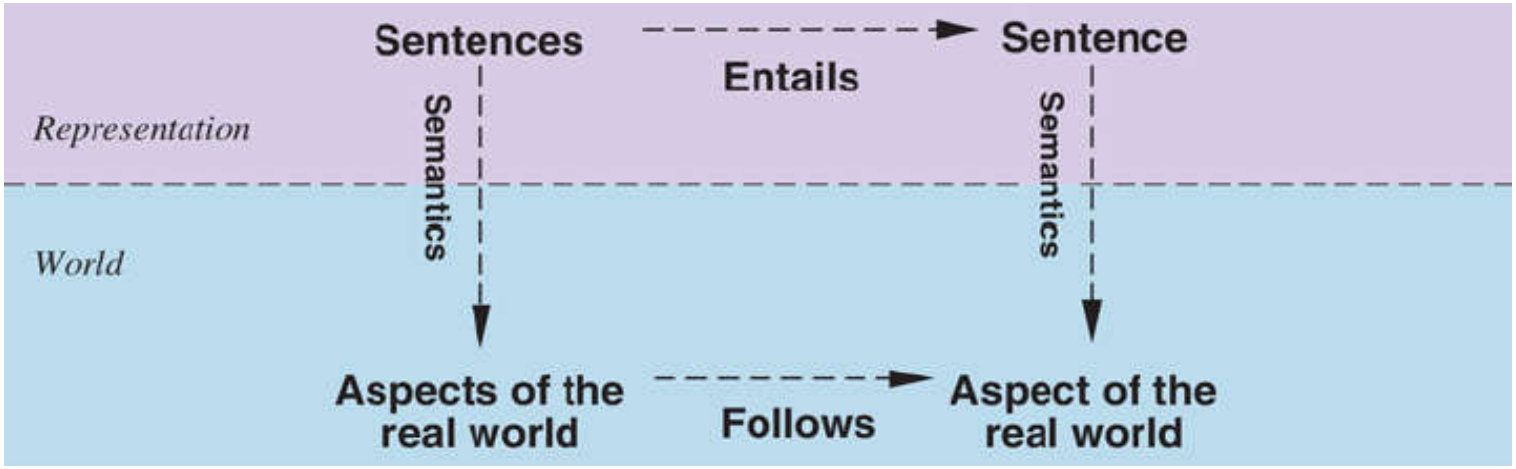
- Agents starts at [1,1] with no active sensor,
  then moves to [2,1] and senses Breeze in it

Possible models for pits in [1,2], [2,2], [3,1]:





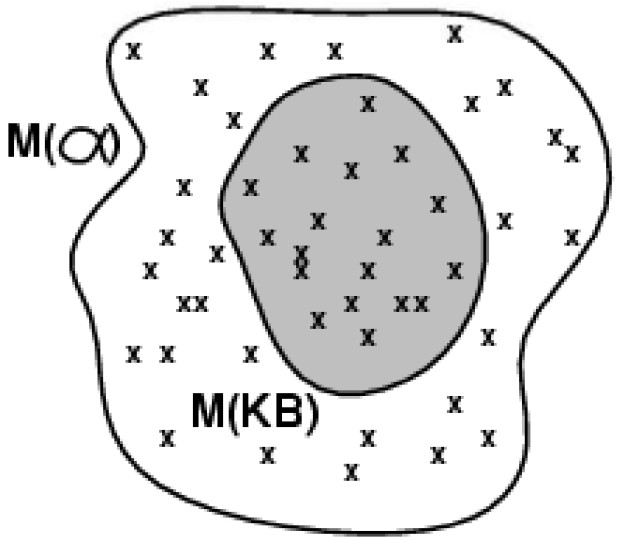
Inference: entailment



Inference: entailment

- $KB \models \alpha$ iff $M(KB) \subseteq M(\alpha)$

- Example:
  - *KB*="A is red" & "B is blue"
  - α = "A is red"



Inference: entailment

- ***Entailment***: $\alpha \models \beta$ ("α entails β" or "β follows from α")

  iff in every world where α is true, β is also true

  - I.e., the α-worlds are a subset of the β-worlds [***models***$(\alpha) \subseteq$ ***models***(β)]
- X+y = 4 entails 4=x+y
- Entails is relation between **syntax of sentences** based on **semantic of sentences**

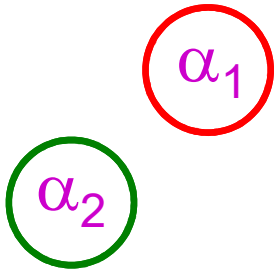
Inference: entailment

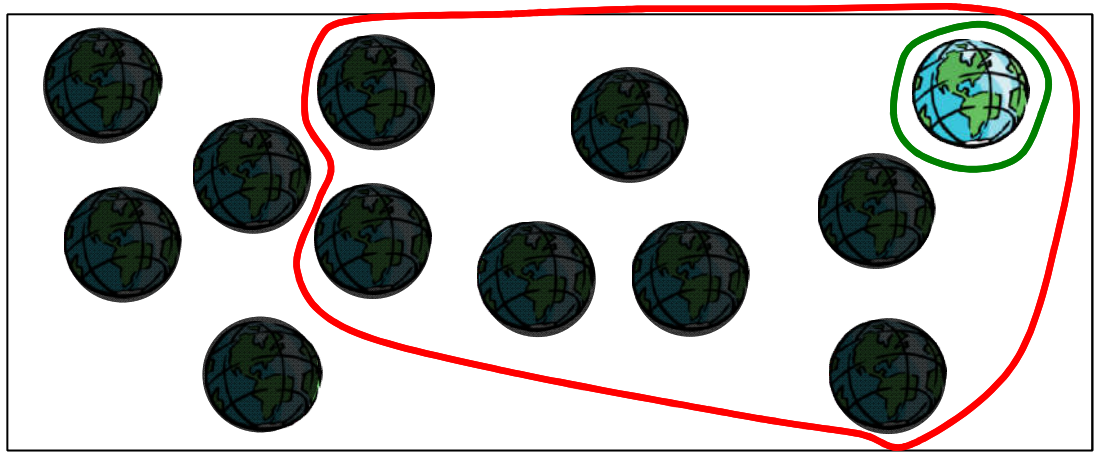
- ***Entailment***: $\alpha \models \beta$ ("α entails β" or "β follows from α")

  iff in every world where α is true, β is also true

  - I.e., the α-worlds are a subset of the β-worlds [***models***(α) $\subseteq$ ***models***(β)]
- In the example, $\alpha_2 \models \alpha_1$
- (Say α_2 is $\neg Q \wedge R \wedge S \wedge W$

  α_1 is $\neg Q$)





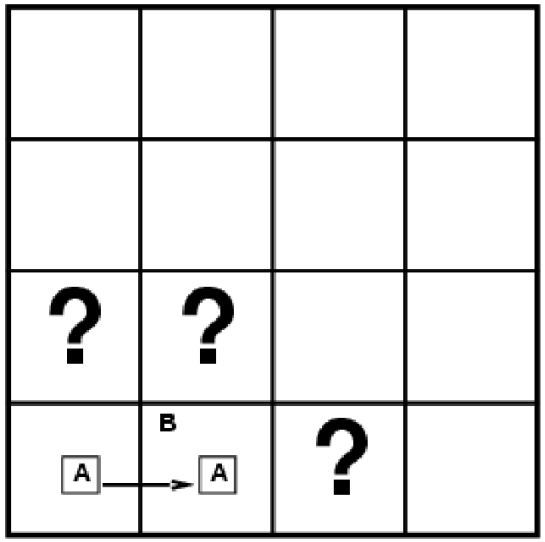
Inference: entailment

- $\alpha \models \beta$ (entailment): α entails β
  - $\alpha \Rightarrow \beta$ **is a tautology or valid**
    - A sentence is **valid** or **tautology** if it is $True$ in all models (e.g., $P \vee \neg P$, $(P \wedge (P \Rightarrow Q)) \Rightarrow Q$)
  - β logically follows from α or α is stronger than β

- $\alpha \models \beta$ iff $M(\alpha) \subseteq M(\beta)$
  - α entails β iff β is true in all worlds where α is true

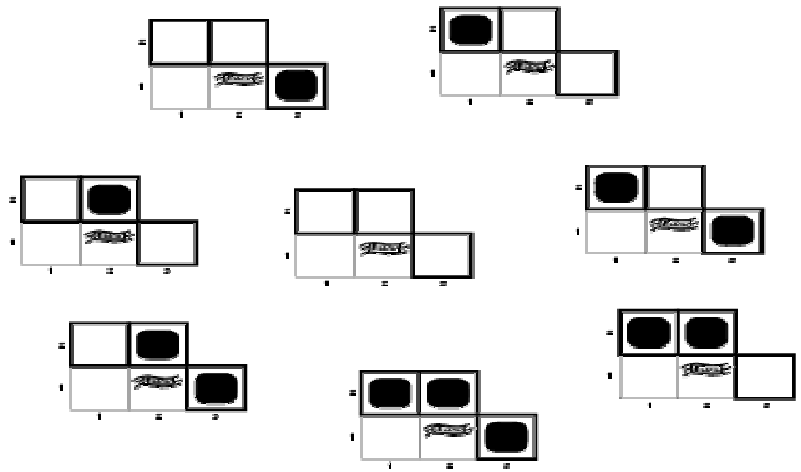
Entailment in the wumpus world

- KB (before perception): rules of the wumpus world
- **Perception**: After detecting nothing in [1,1], moving right, a breeze in [1,2]



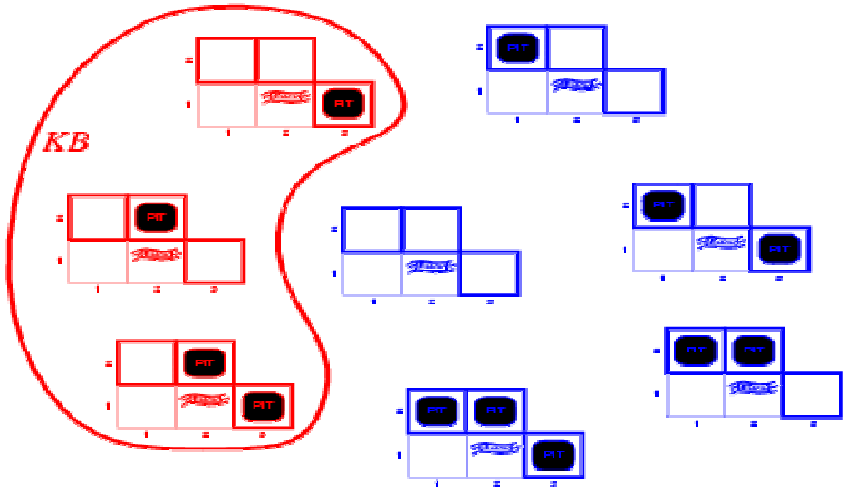
Wumpus model

Possible models for pits in [1,2], [2,2], [3,1]



Consider possible models for only pits in neighboring squares 23 = 8 possible models

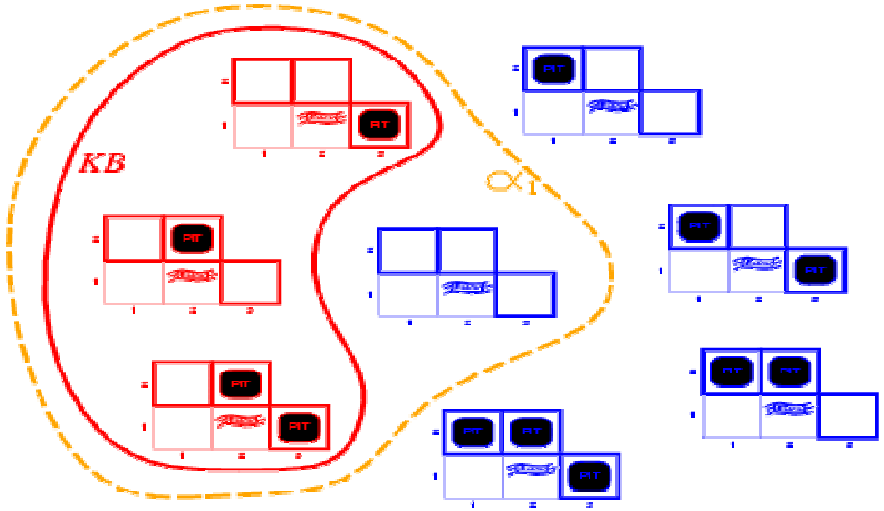
Wumpus model



KB = wumpus-world rules + observations (AND)

(Q) is α_1 = "[1,2] is safe(is not pit)",?

Wumpus model

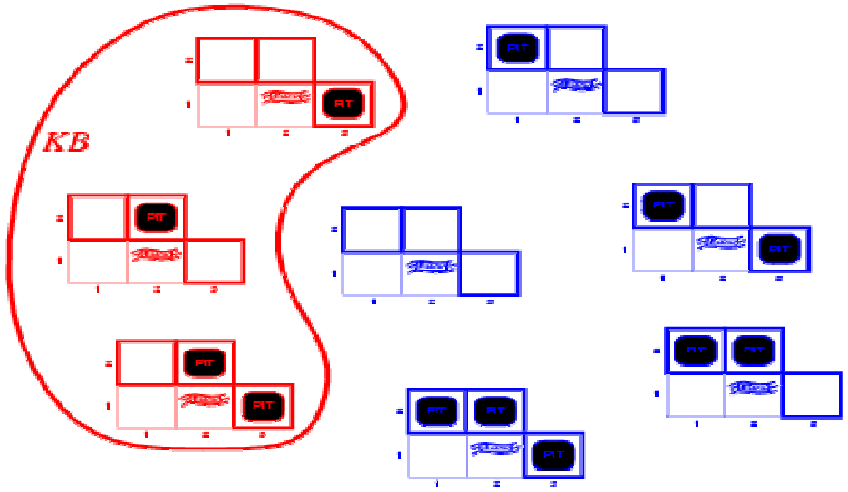


KB = wumpus-world rules + observations

α_1 = "[1,2] is safe(is not pit)",

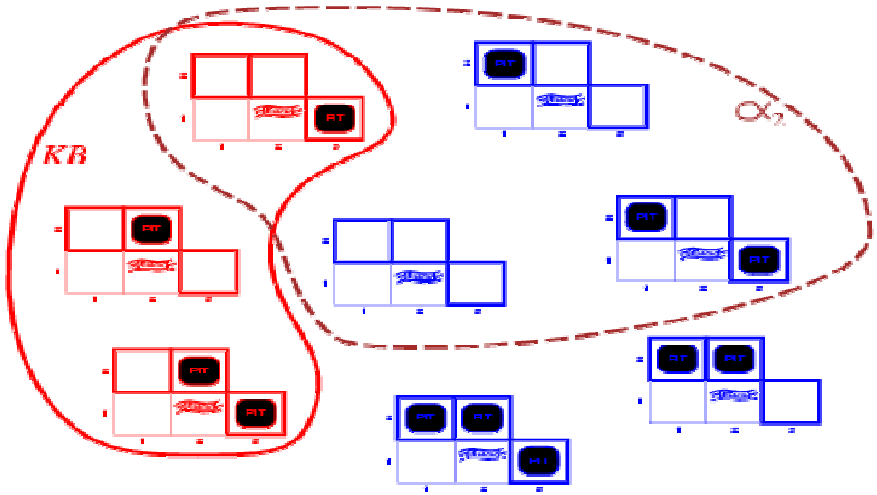
$KB \models \alpha_1$, $M(KB) \subseteq M(\alpha 1)$

Wumpus model



KB = wumpus-world rules + observations

Wumpus model



KB = wumpus-world rules + observations

α_2 = "[2,2] is safe", $KB \not\models \alpha_2$

Inference

- $KB \vdash_i \alpha$: α can be derived from KB by an inference algorithm i

- An inference algorithm i is:
  - sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
    - all provable statements by this algorithm are true
  - complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$
    - all true statements are provable by this algorithm

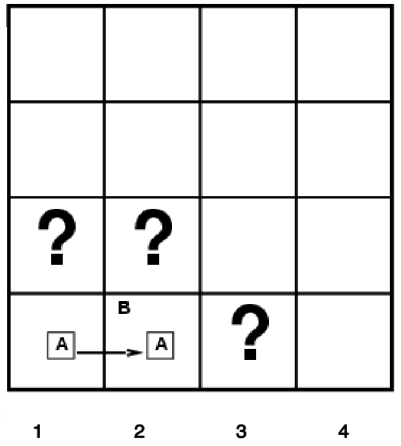
Inference: proofs

- Method 1: ***model-checking***
  - For every possible world, if α is true make sure that is β true too
  - OK for propositional logic (finitely many worlds); not easy for first-order logic
- Method 2: ***theorem-proving***
  - Search for a sequence of proof steps (applications of ***inference rules***) leading from α to β
  - E.g., from $P \wedge (P \Rightarrow Q)$, infer Q by ***Modus Ponens***

METHOD 1: *MODEL-CHECKING*

Wumpus world sentences

- $P_{i,j}$ is true if there is a pit in $[i, j]$.
- $W_{i,j}$ is true if there is a wumpus in $[i, j]$, dead or alive.
- $B_{i,j}$ is true if the agent perceives a breeze in $[i, j]$.
- $S_{i,j}$ is true if the agent perceives a stench in $[i, j]$.

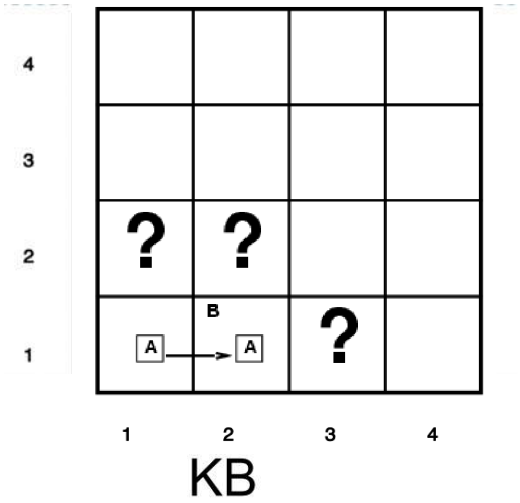


- General rules (only related ones to the current agent position)
  - R_1: $\neg P_{1,1}$ (no pit in [1,1])
  - R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$ (Pits cause breezes in adjacent squares)
  - R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$ (Pits cause breezes in adjacent squares)

- Perception (After each Action)
  - R_4: $\neg B_{1,1}$
  - R_5: $B_{2,1}$

Inference the sentence?

- (Q) is α_1 = "[1,2] is safe(is not pit)",?
- ~$P_{2,1}$



- $KB \models \alpha$?
  - Enumerate the models
  - "Is α true in every model in which KB is true?"

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

Inference by truth table checking

- $KB \models \alpha$?
  - Enumerate the models
  - "Is α true in every model in which KB is true?"

- KB is AND of All sentence

- Algorithm?

KB

R_1: $\neg P_{1,1}$
R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
R_4: $\neg B_{1,1}$
R_5: $B_{2,1}$

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

Entailment by enumeration (model checking)

function $TT\_ENTAILS?(KB, \alpha)$ **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 $TT\_CHECK\_ALL(KB, \alpha, symbols, \{\})$

---

function $TT\_CHECK\_ALL(KB, \alpha, symbols, model)$ **returns** $true$ or $false$
if $EMPTY?(symbols)$ **then**
if $PL\_TRUE?(KB, model)$ **then return** $PL\_TRUE?(\alpha, model)$
else return $true$
else
$P \leftarrow FIRST(symbols)$
$rest \leftarrow REST(symbols)$
return $TT\_CHECK\_ALL(KB, \alpha, rest, model \cup \{P = true\})$ **and**
$TT\_CHECK\_ALL(KB, \alpha, rest, model \cup \{P = false\})$

Entailment by enumeration: properties

- Depth-first enumeration of all models is **sound and complete** (when finite models).
- For n symbols,

  time complexity is $O(2\text{^}n)$, space complexity is $O(n)$.

Validity and satisfiability

A sentence is valid if it is true in all models,

e.g., *True*, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

Validity is connected to the inference via:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ 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 the inference via:

$KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable

DPLL algorithm

```
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 false
  P,value ←FIND-PURE-SYMBOL(symbols,clauses,model)
  if P is non-null then return DPLL(clauses, symbols–P, modelU{P=value})
  P,value ←FIND-UNIT-CLAUSE(clauses,model)
  if P is non-null then return DPLL(clauses, symbols–P, modelU{P=value})
  P ← First(symbols); rest ← Rest(symbols)
  return or(DPLL(clauses,rest,modelU{P=true}),
            DPLL(clauses,rest,modelU{P=false}))
```

The DPLL (Davis, Putnam, Logemann, Loveland) algorithm

Satisfiability is connected to the inference via:

$KB \models \alpha$ if and only if $(KB \wedge \neg \alpha)$ is unsatisfiable

Idea: What is meaning of we found solution for

$(KB \wedge \neg \alpha)$ be true

-> $(KB \wedge \neg \alpha)$ is satisfiable

-> $KB \models \alpha$ is not true

Create CSP with several Constraints → CNF form

CNF Grammar

$$CNFSentence \rightarrow Clause_1 \wedge \cdots \wedge Clause_n$$

$$Clause \rightarrow Literal_1 \vee \cdots \vee Literal_m$$

$$Literal \rightarrow Symbol \mid \neg Symbol$$

$$Symbol \rightarrow P \mid Q \mid R \mid \ldots$$

Efficient SAT solvers

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers
- Recursive depth-first search over models with some extras:
  - ***Early termination***: stop if
    1. all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}
    2. any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=false,B=false}
  - ***Pure literals***: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign(***Pure***), then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true
  - ***Unit clauses***: if a clause is left with a single literal, set symbol to satisfy clause
    - E.g., if A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
    - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.

METHOD 2: *THEOREM-PROVING*

Rules of Inference

$$\frac{P \wedge Q}{\therefore P}$$

And Elimination ($-\wedge$)

$$\frac{P \vee Q \quad \neg P}{\therefore Q}$$

Or Elimination ($-\vee$)

$$\frac{P \quad Q}{\therefore P \wedge Q}$$

And Introduction ($+\wedge$)

$$\frac{P}{\therefore P \vee Q}$$

Or Introduction ($+\vee$)

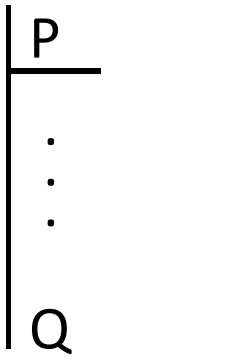
Rules of Inference (cont.)

$P \Rightarrow Q$
P
$\therefore Q$

Modus Ponens
$(-\Rightarrow)$

$\neg\neg P$
$\therefore P$

Double
Negation
$(-\neg)$



$\therefore P \Rightarrow Q$

Conditional
Proof
$(+\Rightarrow)$

$\neg Q$
$\therefore \neg P$

Reductio Ad
Absurdum
$(+\neg)$

Natural deduction example

Prove $((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$

1.	$(P \Rightarrow Q) \wedge (Q \Rightarrow R)$	Assumption
2.	$(P \Rightarrow Q)$	$-\wedge$, 1
3.	$(Q \Rightarrow R)$	$-\wedge$, 1
4.	P	Assumption
5.	Q	$-\Rightarrow$ 2, 4
6.	R	$-\Rightarrow$ 3, 4
7.	$P \Rightarrow R$	$+\Rightarrow$ 4, 6
8.	$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$	$+\Rightarrow$ 1, 7

Logical equivalence

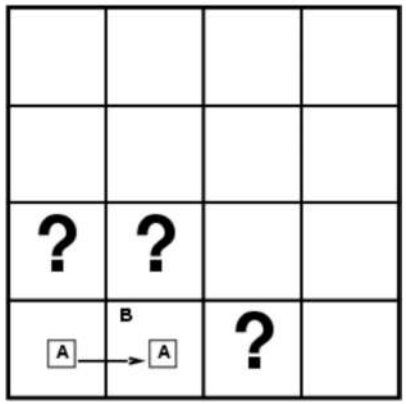
- Two sentences are logically equivalent} iff true in same models: α ≡ ß iff α ⊨ β and β ⊨ α

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

- α, ß and γ stand for arbitrary statements in propositional logic.

Example of inference

- R_1: $\neg P_{1,1}$
- R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- R_4: $\neg B_{1,1}$
- R_5: $B_{2,1}$



- Can we infer from R_1 through R_5 to prove $\neg P_{1,2}$?
  - R_6: $\left(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})\right) \wedge \left((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}\right)$ [biconditional elim. to R_2]
  - R_7: $\left((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}\right)$ [and elim. to R_6]
  - R_8: $\left(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})\right)$ $[(P \Rightarrow Q) \equiv (\neg Q \Rightarrow \neg P)]$
  - R_9: $\neg(P_{1,2} \vee P_{2,1})$ [Modus Ponens with R_8 and R_4]
  - R_{10}: $\neg P_{1,2} \wedge \neg P_{2,1}$ [De Morgan' rule]

Finding a proof as a search problem

- $INITIAL\_STATE$: the initial knowledge base
- $ACTIONS$: all inference rules applied to all the sentences that match their top line
- $RESULT$: add the sentence in the bottom line of the used inference rule to the current ones
- $GOAL$: a state containing the sentence we are trying to prove.

Finding a proof as a search problem

- Inference rules are sound.
- Is a set of rules complete?
  - Are the available inference rules adequate?

FORWARD CHAINING

Simple theorem proving: Forward chaining

- Forward chaining applies

  Modus Ponens to generate new facts:

  Given $X_1 \wedge X_2 \wedge \ldots X_n \Rightarrow Y$ and X_1, X_2, …, X_n, ***infer*** Y

- Forward chaining keeps applying this rule, adding new facts, until nothing more can be added

Horn clause

- Some real-world KBs satisfy restrictions on the form of sentences
  - We can use more restricted and efficient inference algorithms
- Definite clause: a disjunction of literals of which exactly one is positive literal.

$$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_k \vee P_{k+1} \Leftrightarrow (P_1 \wedge P_2 \wedge \ldots \wedge P_k) \Rightarrow P_{k+1}$$

- Horn clause: a disjunction of literals of which at most one is positive literal.
  - Closed under resolution

Simple theorem proving: Forward chaining

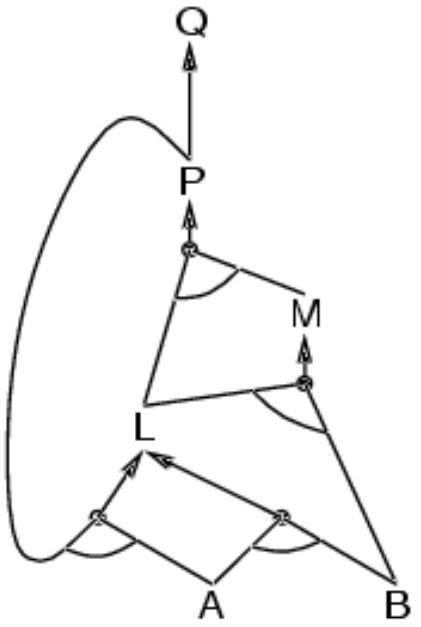
- Requires KB to contain only ***definite clauses***:
  - (Conjunction of symbols) $\Rightarrow$ symbol; or
  - A single symbol (note that X is equivalent to True $\Rightarrow$ X)

- Runs in ***linear*** time using two simple tricks:
  - Each symbol X_i knows which rules it appears in
  - Each rule keeps count of how many of its premises are not yet satisfied

Forward chaining example

- Idea: fire any rule whose premises(symbols) are satisfied in the *KB*,
  - add its conclusion to the *KB*, until query is found (or there is no inference node)

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

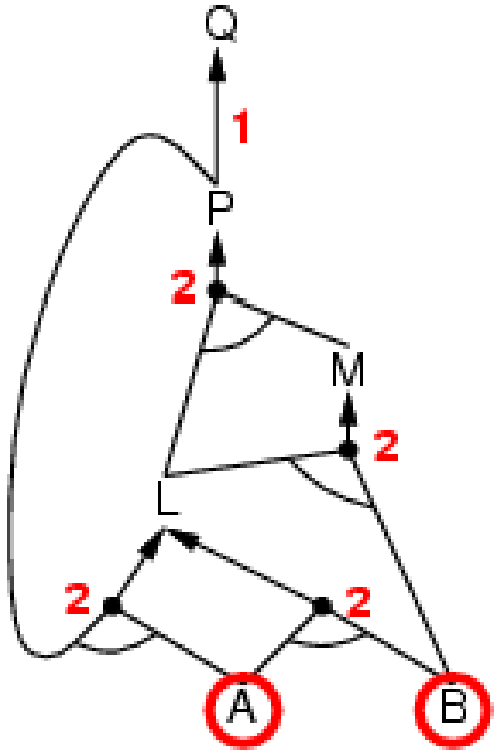


Forward chaining algorithm: Details

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  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 s
  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
```

Forward chaining example

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



Inferred					
A	B	L	M	P	Q

Agenda: {A, B}

Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

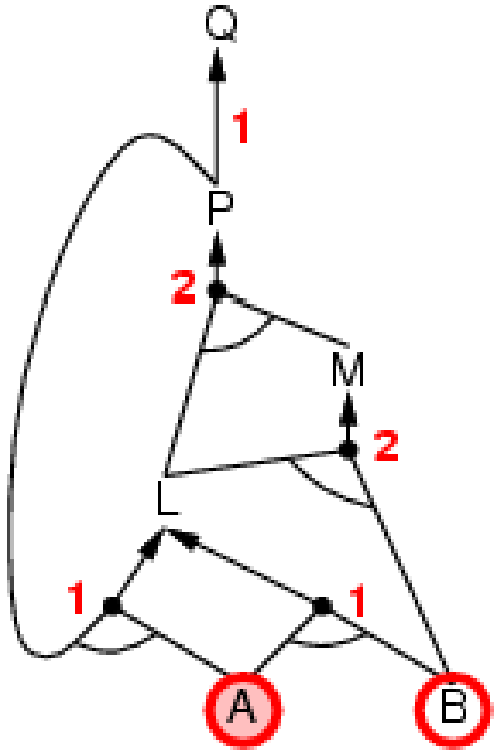
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

√ A

B



Inferred					
A	B	L	M	P	Q
√					

Agenda: {B}

[40-417, SUT]

Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

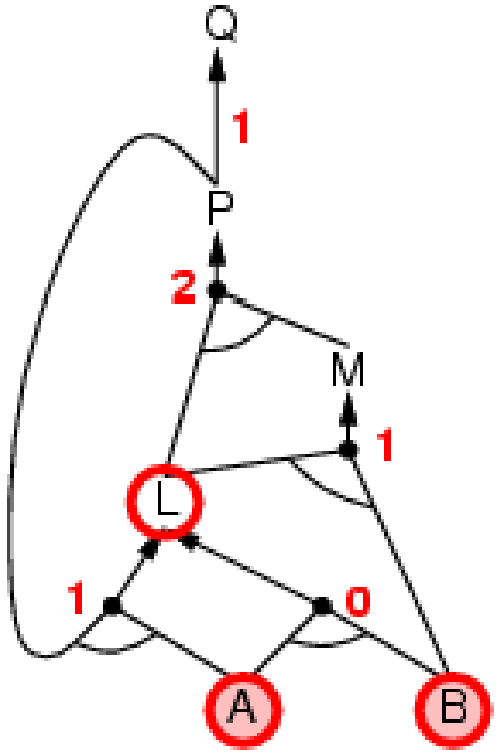
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

√ B



Inferred					
A	B	L	M	P	Q
√	√				

Agenda: {L}

Forward chaining example

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

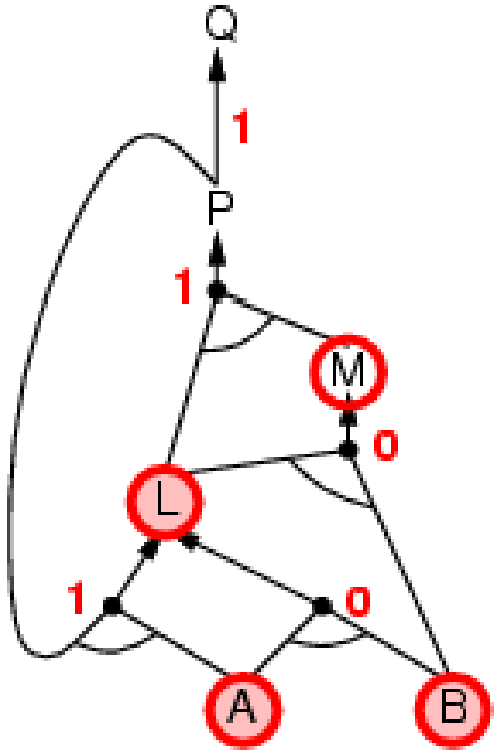
√ $B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

√ B



Inferred					
A	B	L	M	P	Q
√	√	√			

Agenda: {M}

Forward chaining example

$P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

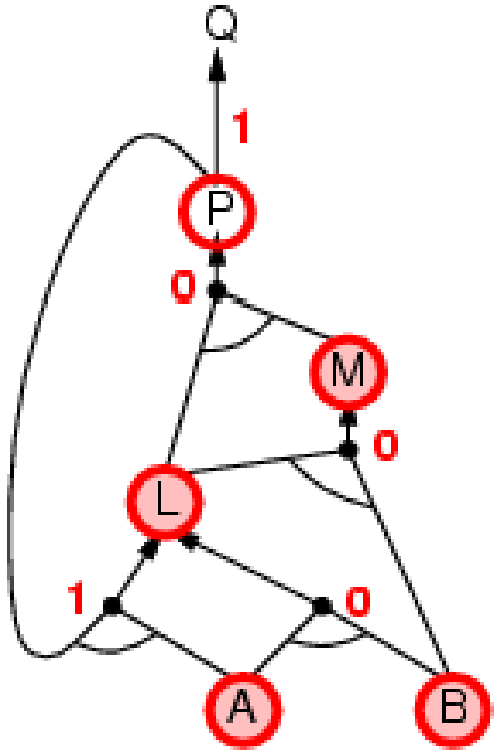
√ $B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

√ B

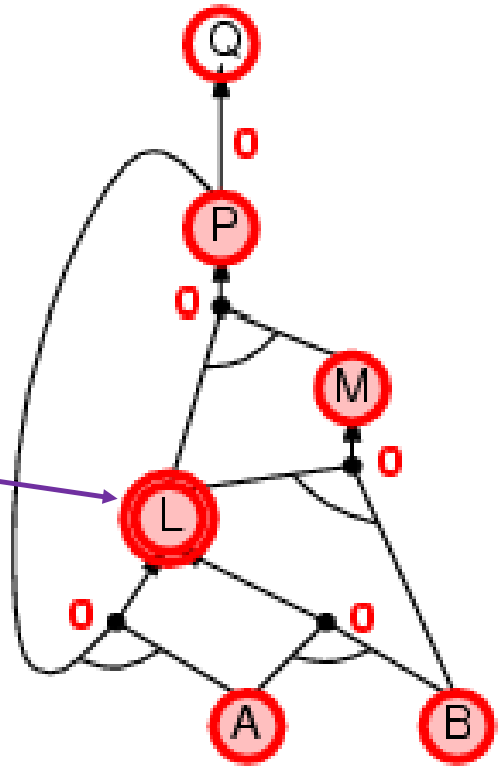


Inferred					
A	B	L	M	P	Q
√	√	√	√		

Agenda: {P}

Forward chaining example

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



- L is not added to the agenda; because it has already been inferred.

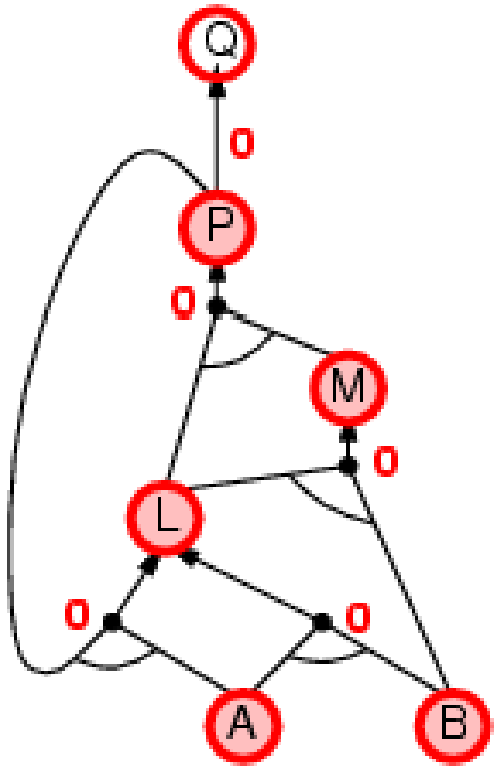
Inferred					
A	B	L	M	P	Q
√	√	√	√	√	

Agenda: {Q}

- This avoids redundant work and prevents loops caused by implications such as X ⇒ Y and Y ⇒ X

Forward chaining example

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

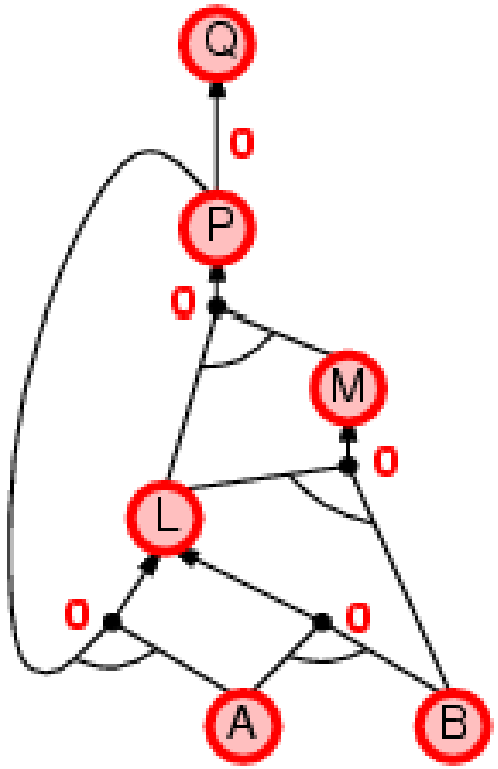


Inferred					
A	B	L	M	P	Q
√	√	√	√	√	

Agenda: {Q}

Forward chaining example

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



Inferred					
A	B	L	M	P	Q
√	√	√	√	√	√

Agenda: {}

Properties of forward chaining

- Theorem: FC is sound and complete for definite-clause KBs
- Soundness:

  follows from soundness of Modus Ponens (easy to check)
- Complete: each atomic entailment can be found

Properties of forward chaining

- Completeness proof:

1. FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final set of known-to-be-true symbols as a model ***m*** (other ones false)
3. (Every clause in the original KB) is true in the model ***m***

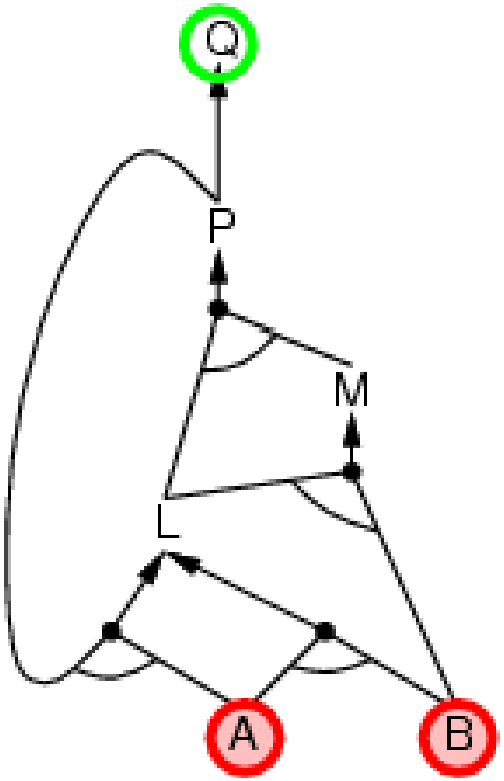
   Proof: Suppose a (Horn)clause $a_1 \wedge \ldots \wedge a_k \Rightarrow b$ is false in ***m***
   Then $a_1 \wedge \ldots \wedge a_k$ is true in ***m*** and b is false in ***m***
   Therefore the algorithm has not reached a fixed point!
4. Hence ***m*** is also a model of KB
5. If KB $\models$ q, q is true in every model of KB, including ***m***

Backward chaining algorithm

- Idea: work backwards from the query *q*:
  - to prove *q* by BC,
    1. check if *q* is known (before), or
    2. 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
  - has already been proved true, or
  - has already failed

Backward chaining example

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



Ex. Goals					
A	B	L	M	P	Q
					√

Inferred					
A	B	L	M	P	Q

Goal stack: {Q}

Backward chaining example

√ $P \Rightarrow Q$

$L \wedge M \Rightarrow P$

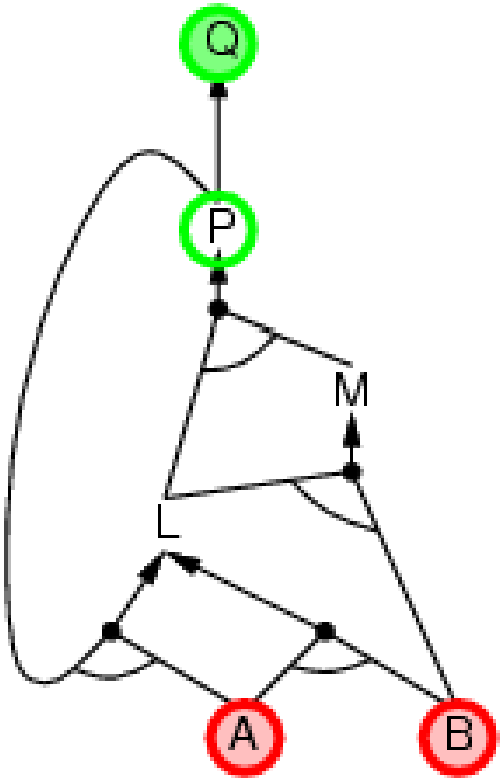
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Ex. Goals					
A	B	L	M	P	Q
				√	√

Inferred					
A	B	L	M	P	Q

Goal stack: {P}

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

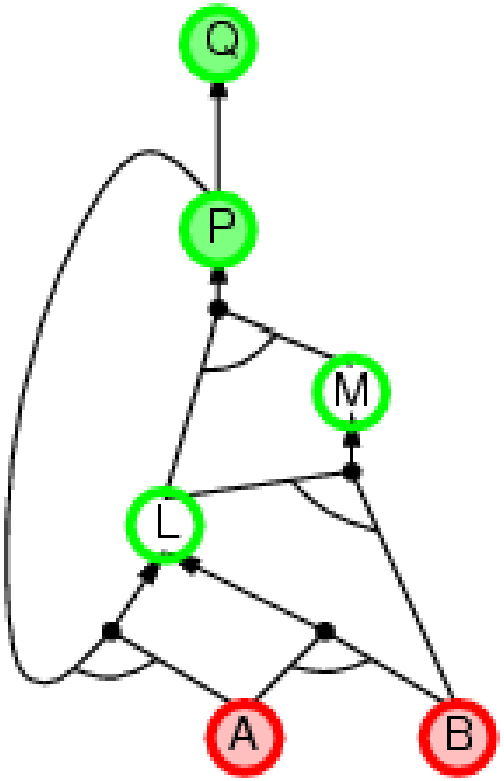
$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Ex. Goals					
A	B	L	M	P	Q
		√	√	√	√

Inferred					
A	B	L	M	P	Q

Goal stack: {L, M}

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

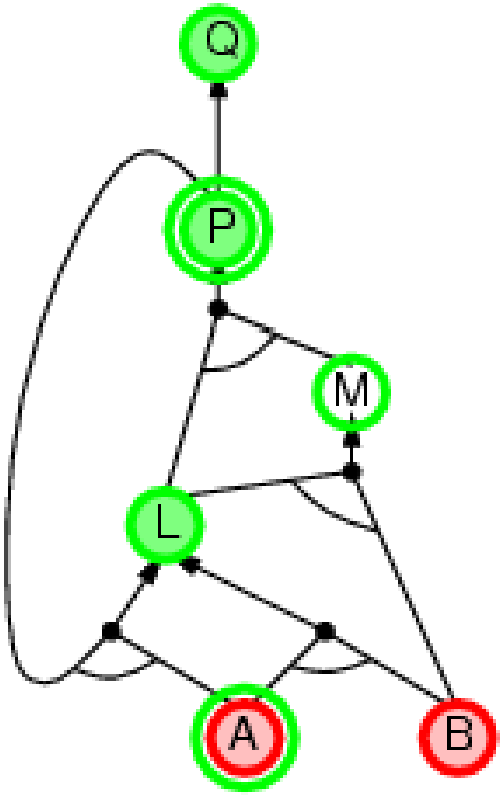
$B \wedge L \Rightarrow M$

√ $A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B



Ex. Goals					
A	B	L	M	P	Q
√		√	√	√	√

Inferred					
A	B	L	M	P	Q

Goal stack: {A, M}

- P is not added to the G.S. again; because it was there once before.

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

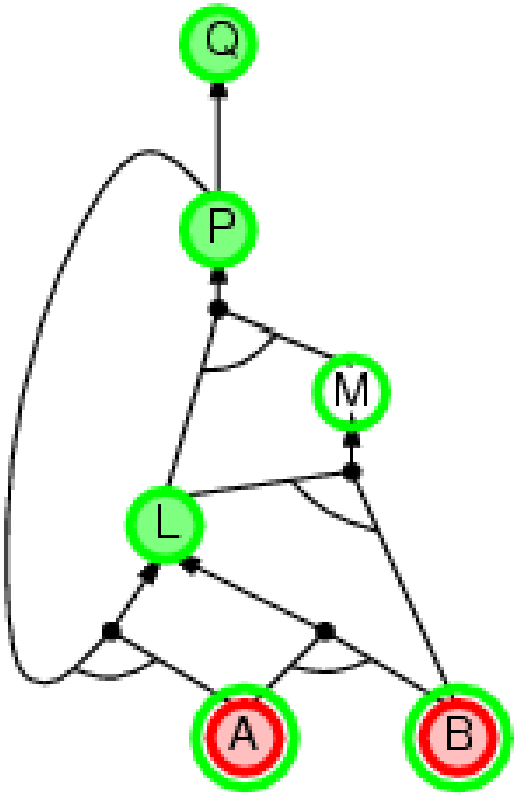
$B \wedge L \Rightarrow M$

√ $A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

A

B



Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q

Goal stack: {A, B, M}

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

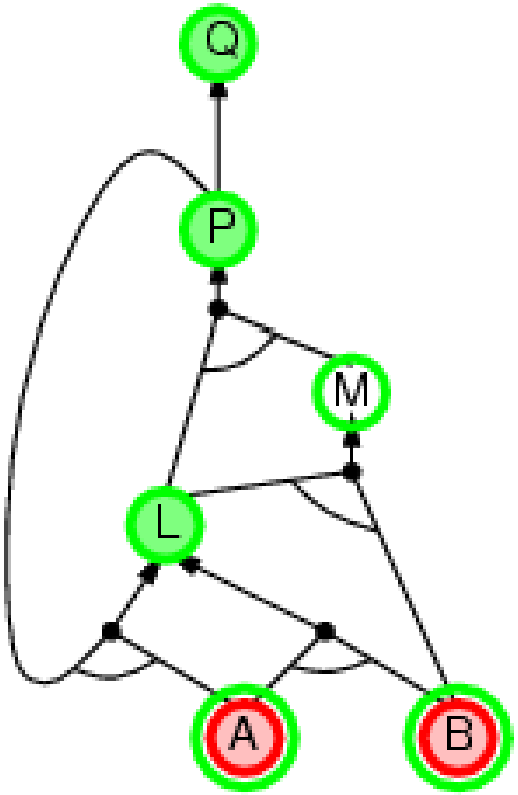
$B \wedge L \Rightarrow M$

√ $A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

B



Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q
√					

Goal stack: {B, M}

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

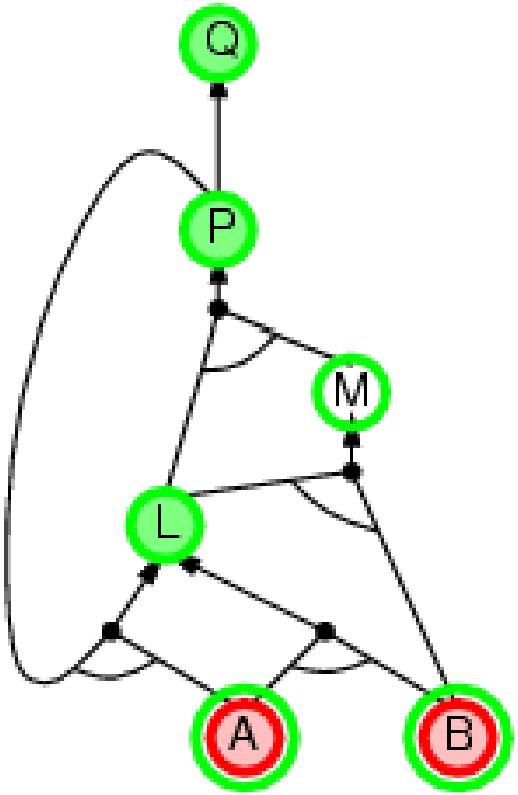
$B \wedge L \Rightarrow M$

√ $A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

√ B



Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q
√	√				

Goal stack: {M}

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

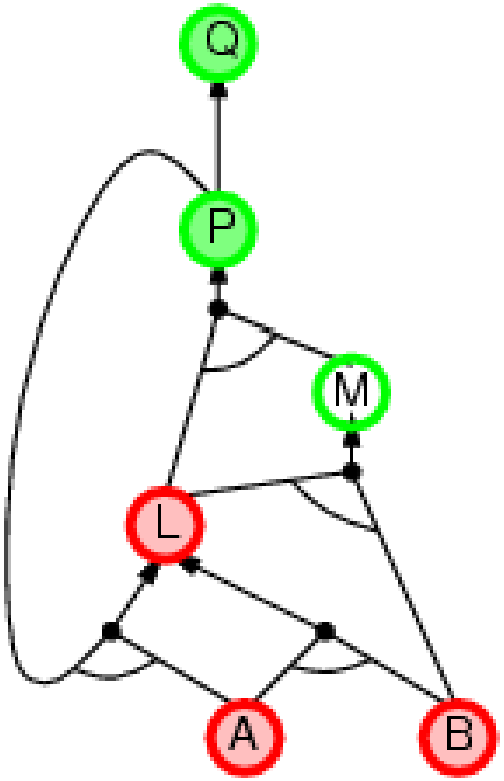
$B \wedge L \Rightarrow M$

√ $A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

√ B



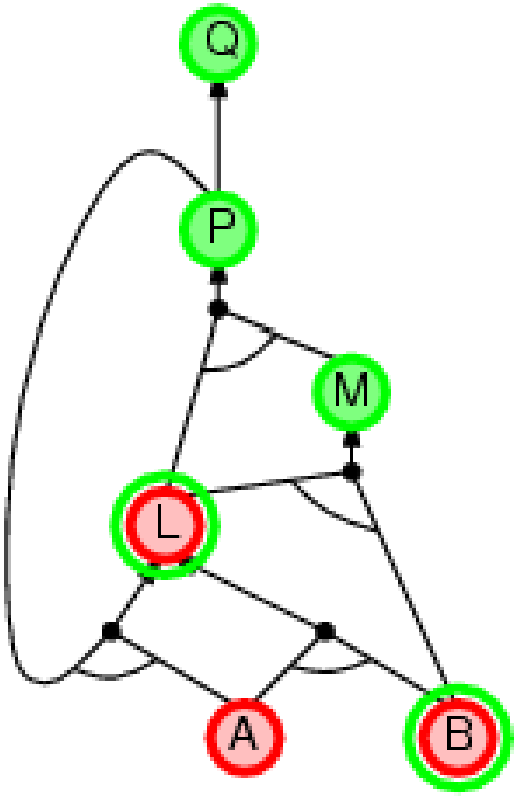
Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q
√	√	√			

Goal stack: {M}

Backward chaining example

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



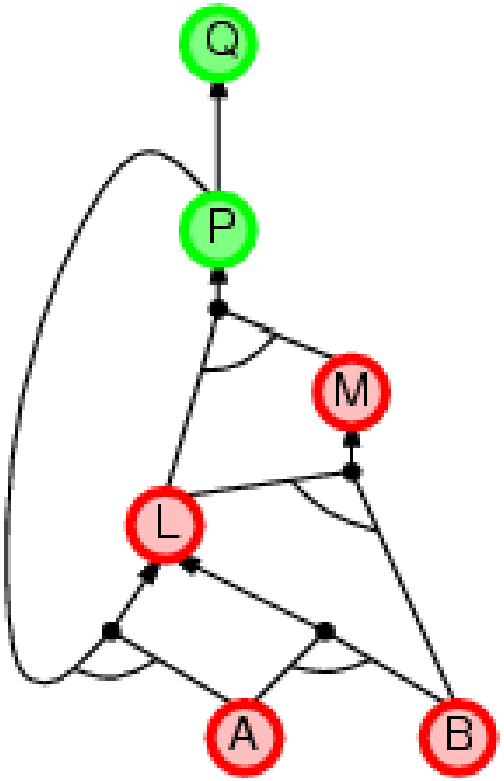
Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q
√	√	√			

Goal stack: {}

Backward chaining example

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



Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

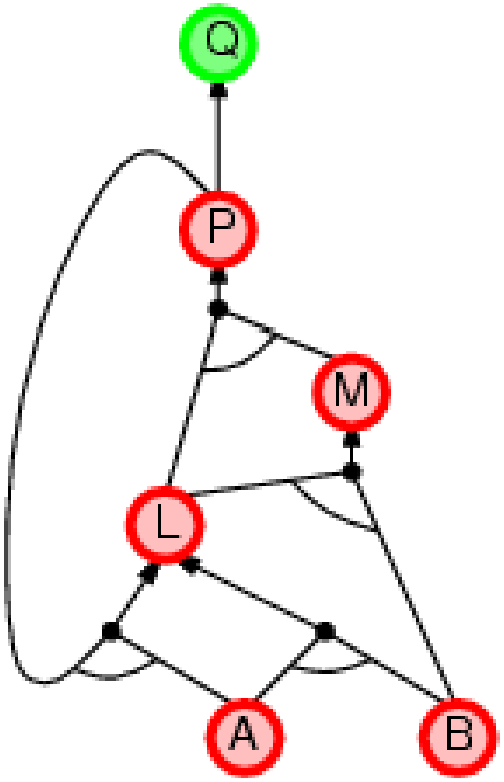
Inferred					
A	B	L	M	P	Q
√	√	√	√		

Goal stack: {}

Backward chaining example

- Here L is inferred for the second 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



Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q
√	√	√	√	√	

Goal stack: {}

Backward chaining example

√ $P \Rightarrow Q$

√ $L \wedge M \Rightarrow P$

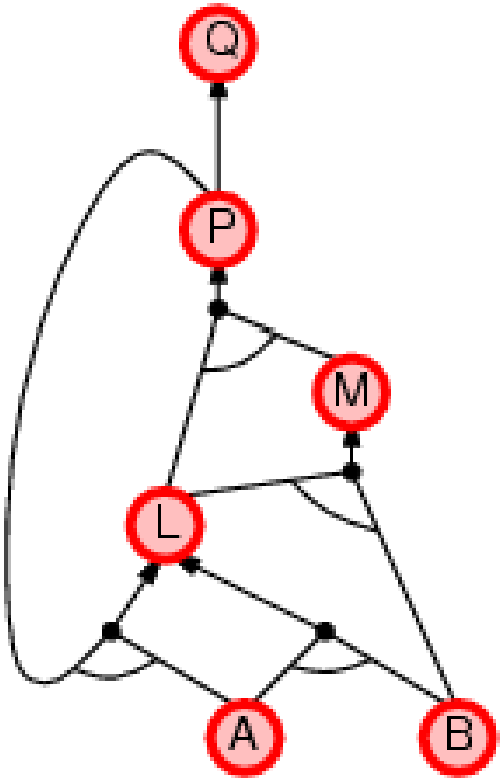
√ $B \wedge L \Rightarrow M$

√ $A \wedge P \Rightarrow L$

√ $A \wedge B \Rightarrow L$

√ A

√ B



Ex. Goals					
A	B	L	M	P	Q
√	√	√	√	√	√

Inferred					
A	B	L	M	P	Q
√	√	√	√	√	√

Goal stack: {}

Satisfiability and entailment

- A sentence is ***satisfiable*** if it is true in at least one world
- Suppose we have a hyper-efficient SAT solver (**WARNING: NP-COMPLETE** 😈😈😈); how can we use it to test entailment?
  - $\alpha \models \beta$
  - iff $\alpha \Rightarrow \beta$ is true in all worlds
  - iff $\neg(\alpha \Rightarrow \beta)$ is false in all worlds
  - iff $\alpha \wedge \neg\beta$ is false in all worlds, i.e., unsatisfiable
- So, add the ***negated*** conclusion to what you know, test for (un)satisfiability; also known as reductio ad absurdum
- Efficient SAT solvers operate on ***conjunctive normal form***

Conjunctive normal form (CNF)

- Every sentence can be expr[illegible] ***clauses***
- Each clause is a ***disjunction*** o[illegible]
- Each literal is a symbol or a [illegible]ymb[illegible]
- Conversion to CNF by a se[illegible]qu[illegible]ce of standard [illegible]sformations:
  - At_1,1_0 $\Rightarrow$ (Wall_0,1 $\Leftrightarrow$ B[illegible]ocked_W_0)
  - At_1,1_0 $\Rightarrow$ ((Wall_0,1 $\Rightarrow$ Blocked_W_0) $\wedge$ ([illegible]ocked_W_0 $\Rightarrow$Wall_0,1))
  - $\neg$At_1,1_0 v (($\neg$Wall_0,1 v Blocked_W_0) $\wedge$ ($\neg$Blocked_W_0 v Wall_0,1))
  - ($\neg$At_1,1_0 v $\neg$Wall_0,1 v Blocked_W_0) $\wedge$
    ($\neg$At_1,1_0 v $\neg$Blocked_W_0 v Wall_0,1)

Replace biconditional by two implications

Replace $\alpha \Rightarrow \beta$ by $\neg\alpha$ v β

Distribute v over $\wedge$

Monotonicity property

- The set of entailed sentences can only grow as information is added to KB.
  - Inference rules can be applied where premises are found in KB (regardless of what else is in KB)
- If $KB \models \alpha$ then $KB \wedge \beta \models \alpha$ (for any α and β)
  - The assertion β can not invalidate any conclusion α already inferred.

Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing
- May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving
- Complexity of BC can be much less than linear in the size of KB

RESOLUTION

Resolution

- The resolution inference rule takes two implication sentences (of a particular form) and infers a new implication sentence:

Unit resolution: l_i and m are complementary literals

$$\frac{l_1 \vee l_2 \vee \cdots \vee l_k, \quad m}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k}$$

- Resolution
  - Work With CNF Form
  - a single inference rule that yields a complete inference algorithm
  - complete for propositional logic
  - Exponential time in the worst case

Conversion to CNF

- $B \Leftrightarrow (P \vee Q)$
- $(B \Rightarrow (P \vee Q)) \wedge ((P \vee Q) \Rightarrow B)$
- $(\neg B \vee P \vee Q) \wedge (\neg (P \vee Q) \vee B)$
- $(\neg B \vee P \vee Q) \wedge ((\neg P \wedge \neg Q) \vee B)$
- $(\neg B \vee P \vee Q) \wedge (\neg P \vee B) \wedge (\neg Q \vee B)$

1) Replace $\Leftrightarrow$ by two $\Rightarrow$

2) Replace $\alpha \Rightarrow \beta$ by $\neg\alpha \vee \beta$

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

4) Distribute $\vee$ over $\wedge$

Resolution

- Resolution inference rule (for CNF):

$$\frac{l_1 \vee \ldots \vee l_k \qquad m_1 \vee \ldots \vee m_n}{l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n}$$

  where l_i and m_j are complementary literals.

- E.g.,

$$\frac{P \vee Q \qquad \neg Q}{P}$$

- Resolution is sound and complete for propositional logic.

Resolution algorithm

- Proves entailment by *Refutation*, i.e., show ($KB \wedge \neg\alpha$) is unsatisfiable

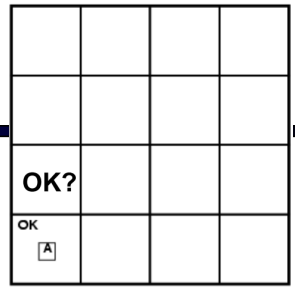
function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses $\leftarrow$ the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new $\leftarrow \{\}$
  while *true* **do**

Each pair containing complementary literals is resolved and the resulted clause is added to the set if it is not already present.

The process continues until one of these happens:
- No new clauses ($KB \not\models \alpha$)
- Two clauses resolve to yield the empty clause ($KB \models \alpha$)

Resolution example



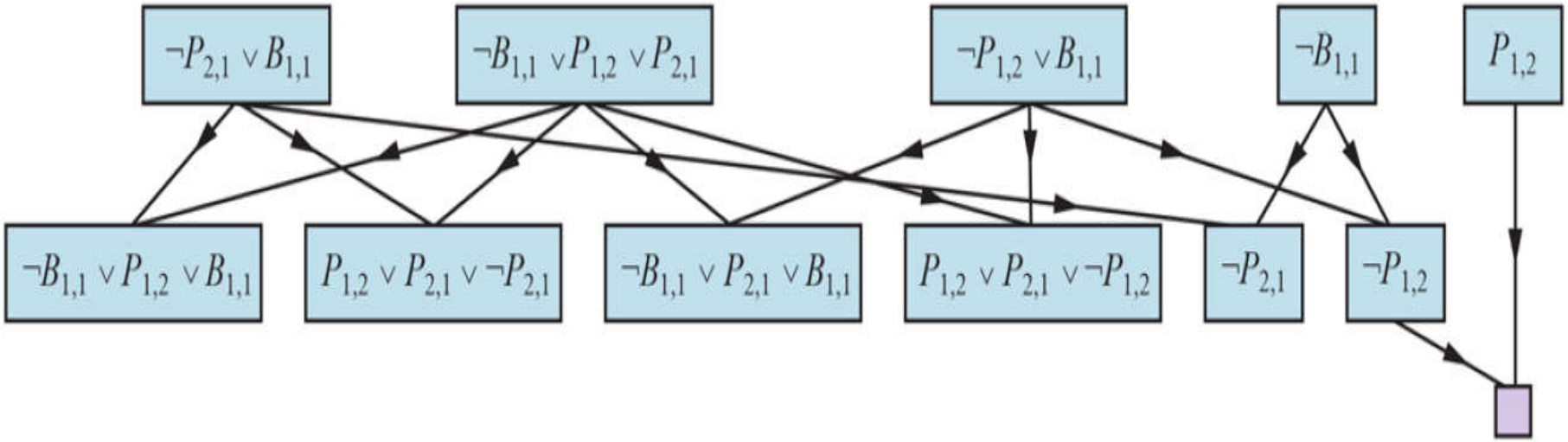
$KB = \left(B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})\right) \wedge \neg B_{1,1}$

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

Prove:

KB = { $\neg P_{2,1} \vee B_{1,1}$, $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$, $\neg P_{1,2} \vee B_{1,1}$, $\neg B_{1,1}$ }

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



Propositional resolution - example

KB:

P

$(P \wedge Q) \Rightarrow R$

$(S \vee T) \Rightarrow Q$

T

KB (in CNF):

P

$\neg P \vee \neg Q \vee R$

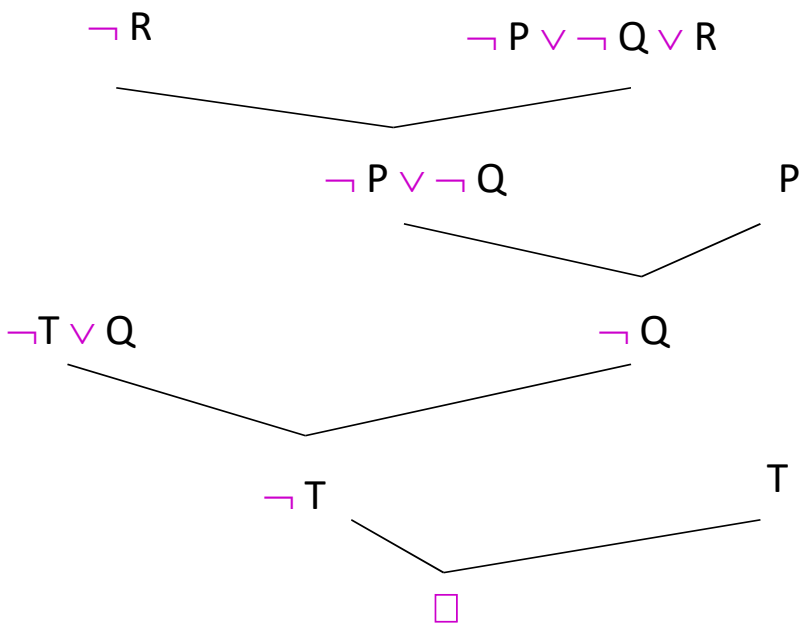
$\neg S \vee Q$

$\neg T \vee Q$

T

Prove

$KB \models R$



[40-417, SUT]

Only select one pair to resolve

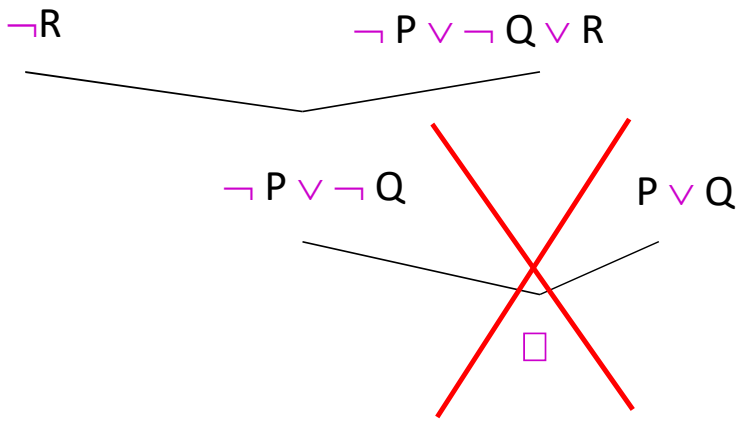
KB (in CNF):

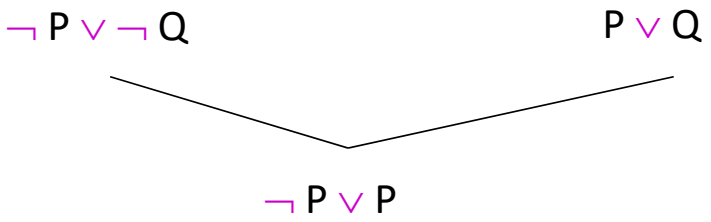
$\neg P \vee \neg Q \vee R$

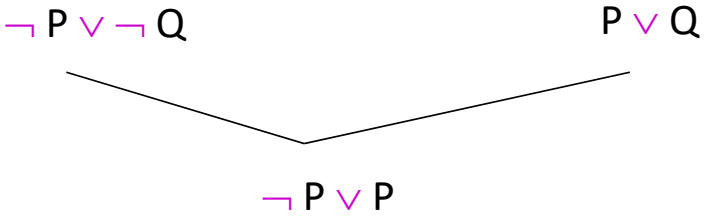
$P \vee Q$

Prove

$KB \models R$







- But, is R entailed by the two facts we have been given? NO!

KB in the standard form:

$P \wedge Q \Rightarrow R$

$P \vee Q$

[40-417, SUT]

Efficient Propositional Inference

Efficient algorithms for general propositional inference:

- We introduced DPLL algorithm as a systematic search method
  Backtracking search algorithms
- we will introduce (non systematic search method)
  - WalkSAT algorithm as a local search algorithm
  - Incomplete

Efficient SAT solvers

- DPLL (Davis-Putnam-Logemann-Loveland) is the core of modern solvers
- Recursive depth-first search over models with some extras:
  - ***Early termination***: stop if
    - all clauses are satisfied; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=true}
    - any clause is falsified; e.g., $(A \vee B) \wedge (A \vee \neg C)$ is satisfied by {A=false,B=false}
  - ***Pure literals***: if all occurrences of a symbol in as-yet-unsatisfied clauses have the same sign, then give the symbol that value
    - E.g., A is pure and positive in $(A \vee B) \wedge (A \vee \neg C) \wedge (C \vee \neg B)$ so set it to true
  - ***Unit clauses***: if a clause is left with a single literal, set symbol to satisfy clause
    - E.g., if A=false, $(A \vee B) \wedge (A \vee \neg C)$ becomes $(\text{false} \vee B) \wedge (\text{false} \vee \neg C)$, i.e. $(B) \wedge (\neg C)$
    - Satisfying the unit clauses often leads to further propagation, new unit clauses, etc.

DPLL algorithm

```
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 false
  P,value ←FIND-PURE-SYMBOL(symbols,clauses,model)
  if P is non-null then return DPLL(clauses, symbols–P, modelU{P=value})
  P,value ←FIND-UNIT-CLAUSE(clauses,model)
  if P is non-null then return DPLL(clauses, symbols–P, modelU{P=value})
  P ← First(symbols); rest ← Rest(symbols)
  return or(DPLL(clauses,rest,modelU{P=true}),
            DPLL(clauses,rest,modelU{P=false}))
```

Efficiency

- Naïve implementation of DPLL: solve ~100 variables
- Extras:
  - Smart variable and value ordering
  - Divide and conquer
  - Caching unsolvable subcases as extra clauses to avoid redoing them
  - Cool indexing and incremental recomputation tricks so that every step of the DPLL algorithm is efficient (typically O(1))
    - Index of clauses in which each variable appears +ve/-ve
    - Keep track number of satisfied clauses, update when variables assigned
    - Keep track of number of remaining literals in each clause
- Real implementation of DPLL: solve ~100000000 variables

The *WalkSAT* algorithm

- It tests entailment $KB \models \alpha$ by testing unsatisfiability of $KB \wedge \neg\alpha$.
  - Local search for (satisfy solution of $KB \wedge \neg\alpha$)
- Local search algorithms (hill climbing, SA,…)
  - state space: complete assignments
  - valuation function: number of unsatisfied clauses
- WalkSat
  - Simple & effective
  - Balance between greediness and randomness (To escape from local minima)
  - Evaluation function:
    - The min-conflict heuristic of minimizing the number of unsatisfied clauses
  - Balance between greediness and randomness
  - Incomplete, local search algorithm

The *WalkSAT* algorithm

Local search for (satisfy solution of $KB \wedge \neg\alpha$)

Each iteration

- Select an unsatisfied clause
  - Select a symbol to reveres it
    - P :random select symbol
    - (1-p):symbol with maximize satisfieble clause

The *WalkSAT* algorithm

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 value flips allowed before giving up

model ← a random assignment of *true/false* to the symbols in *clauses*

for each $i = 1$ **to** *max_flips* **do**

if *model* satisfies *clauses* **then return** *model*

clause ← a randomly selected clause from *clauses* that is false in *model*

if RANDOM(0, 1) $\leq p$ **then**

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*

Good Reference for logic

- https://milnepublishing.geneseo.edu/concise-introduction-to-logic